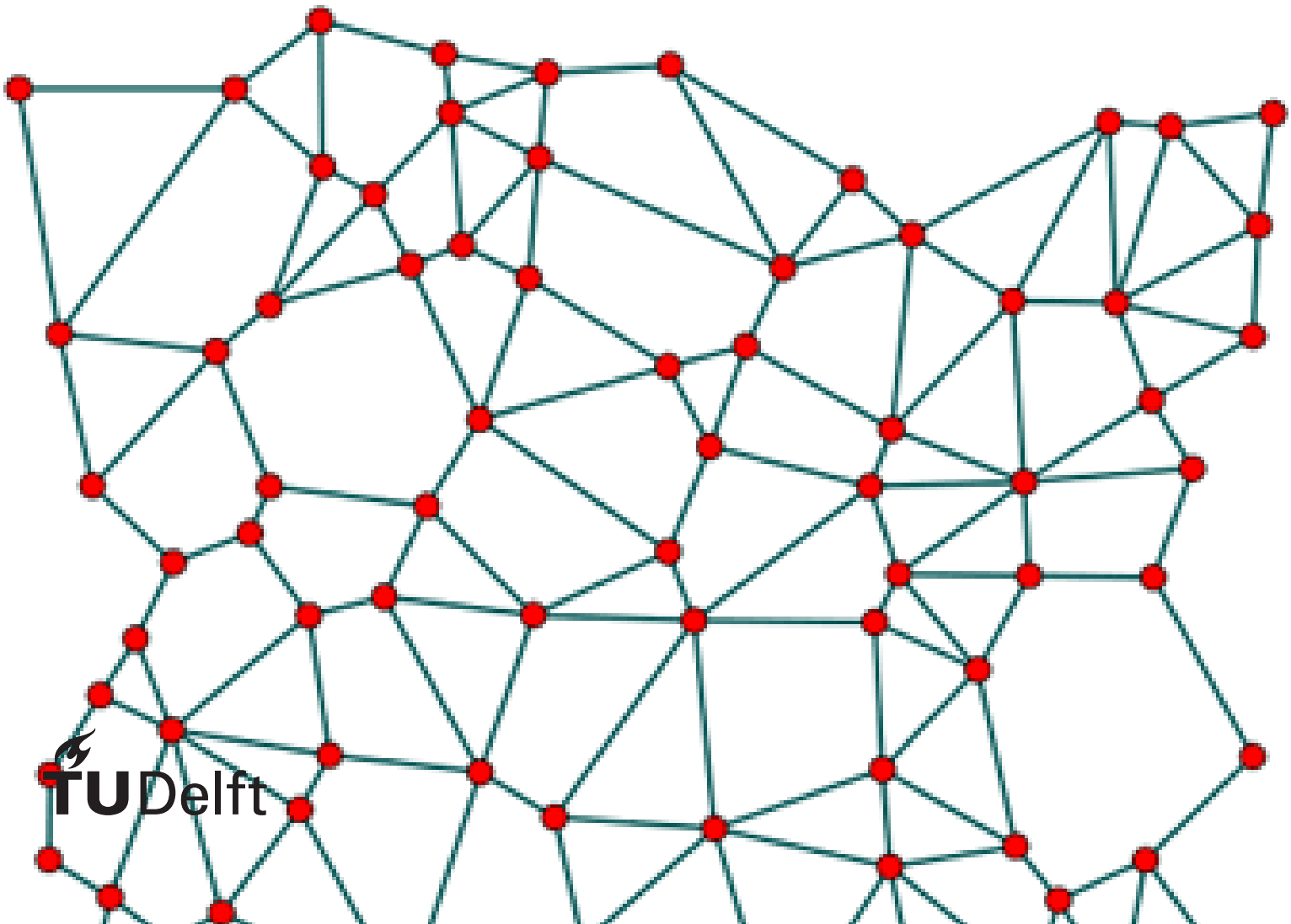
Maximum Independent Set Algorithms

The Effect of Graph Structure on Greedy, Simulated Annealing, and Lotka–Volterra Methods

Nina Forstner



Maximum Independent Set Algorithms

The Effect of Graph Structure on Greedy, Simulated Annealing, and Lotka–Volterra Methods

by

Nina Forstner

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Friday July 17, 2026 at 10:00 AM.

Student number:	6020313	
Project duration:	Feb 17, 2026 – July 17, 2026	
Thesis committee:	Dr. Yves van Gennip,	TU Delft, supervisor
	Dr. Sebastian Feld,	TU Delft, supervisor
	Dr. Marlies Goorden,	TU Delft
	Dr. Dion Gijswijt,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`. Cover image: adapted from a Gabriel graph by David Eppstein, public domain, via Wikimedia Commons [11].

TU Delft

Abstract

The maximum-independent-set problem is a fundamental graph problem with applications in situations where one wants to select as many mutually compatible objects as possible, such as non-conflicting tasks or choices. This problem is computationally difficult to solve exactly on large graphs. This motivates the use of heuristic algorithms. However, heuristic performance can depend strongly on the structure of the input graph, so it is not enough to ask which algorithm performs best on average. It is also important to understand when and why a heuristic fails.

This Bachelor End Project studies heuristic algorithms for the maximum-independent-set problem from a structural point of view. We compare three methods: a minimum-degree Greedy algorithm, Lotka–Volterra dynamics, and simulated annealing, using an exact solver as a benchmark on small graphs. The goal is not only to compare solution sizes, but also to identify the graph structures and parameter choices that cause each method to perform poorly.

The results show that the algorithms fail for different structural reasons. Greedy can perform poorly when the minimum-degree rule is locally attractive but globally misleading. In the tested bad examples, Greedy selects vertices that look good by degree, but this choice blocks access to a much larger independent set. Lotka–Volterra dynamics can converge to maximal independent sets that are not maximum. This behavior is influenced by the initial condition and becomes more important when the graph contains many competing maximal independent sets. Simulated annealing is affected by the structure of the energy landscape. In particular, the experiments suggest that failure is not caused only by a single large energy barrier, but also by the presence of many competing trap states.

The parameter experiments support these interpretations. Increasing the Lotka–Volterra competition parameter τ strengthens early suppression between neighboring vertices, but often leads to smaller independent sets. For simulated annealing, increasing the penalty parameter α makes edge conflicts more expensive and helps the algorithm reach feasible states earlier, but it can also reduce exploration through temporary conflict states. Longer cooling schedules improve performance, especially on larger or more crowded instances, but require more computation.

Overall, the thesis shows that the performance of maximum-independent-set heuristics cannot be explained by graph size alone. The relevant difficulty depends on graph structure, the number of competing maximal independent sets, the initial condition, and the algorithm parameters. This gives a more diagnostic view of heuristic performance: instead of treating Greedy, Lotka–Volterra, and simulated annealing as black-box methods, the thesis identifies structural warning signs that indicate when each method is likely to struggle.

Layman's Summary

Many real-world problems can be described as networks, where points represent objects and lines represent connections between them. This thesis studies the problem of finding the largest possible group of points where no two chosen points are connected. This is called the maximum-independent-set problem.

Although the problem is easy to describe, it is difficult to solve exactly for large networks. Therefore, this thesis compares three methods that try to find good solutions: a greedy algorithm, simulated annealing, and Lotka–Volterra dynamics. The greedy algorithm makes simple choices step by step. Simulated annealing searches for good solutions by sometimes accepting temporary worse choices. Lotka–Volterra dynamics use equations inspired by competition between species, where connected points suppress each other.

The goal is not only to compare the methods, but also to understand why they sometimes fail. The results show that each method has its own weakness. Greedy can make an early choice that blocks a better solution. Lotka–Volterra can settle into a solution that is good locally but not globally. Simulated annealing can struggle when there are many competing near-solutions or when the time to settle is too short.

Overall, the results show that performance depends not only on the size of the network, but also on its structure and on the chosen algorithm parameters. There is therefore no single best method for all graph types.

Preface

This thesis marks the completion of my Bachelor End Project, and I would like to express my sincere gratitude to everyone who supported me throughout this process. I am especially thankful to my family, friends and boyfriend for their immense support, encouragement, and patience along the way. I would also like to thank my study advisor, GertJan Broekman, for his continued support and guidance during my studies. Finally, I am very grateful to my BEP supervisors for their valuable guidance, feedback, and time throughout this project.

Nina Forstner
Delft, July 2026

Contents

Abstract ii

Layman's Summary iii

1 Introduction 1
- 1.1 Research Question . . . 3
- 1.2 Reading Guide . . . 3

2 Theoretical Background 4
- 2.1 Basic Graph Theory . . . 4
- 2.2 Graph Types . . . 6
- 2.3 Complexity Theory . . . 8

3 Algorithms and Experimental Methodology 10
- 3.1 Exhaustive Enumeration . . . 10
- 3.2 Minimum-Degree Greedy Algorithm . . . 11
- 3.3 Lotka–Volterra Dynamics . . . 11
  - 3.3.1 Equilibria and Stability . . . 13
  - 3.3.2 Why the Condition $\tau > 1$ Matters . . . 13
  - 3.3.3 Lyapunov Function and Energy Landscape . . . 15
  - 3.3.4 The Lyapunov Function Decreases Along LV Trajectories . . . 16
  - 3.3.5 Maximal Independent Sets and Local Minima . . . 17
  - 3.3.6 Maximum Independent Sets and Ground States . . . 17
  - 3.3.7 Theoretical Justification for the LV Method . . . 17
  - 3.3.8 Numerical Implementation Used in This Thesis . . . 18
  - 3.3.9 Thresholding and Discretization Effects . . . 20
  - 3.3.10 Preventing Bias from the Threshold θ . . . 20
- 3.4 Simulated Annealing . . . 21
  - 3.4.1 Energy Formulation Using a QUBO . . . 21
  - 3.4.2 Simulated Annealing Algorithm . . . 21
  - 3.4.3 Metropolis Acceptance Rule . . . 22
  - 3.4.4 Cooling Schedule . . . 23
  - 3.4.5 Other Cooling Mechanisms . . . 23
  - 3.4.6 Temperature Parameters Used in This Thesis . . . 24
  - 3.4.7 Why $\alpha > 1$ . . . 25
  - 3.4.8 How $H(x)$ Connects to Independent Sets . . . 26
  - 3.4.9 Final Simulated Annealing Procedure Used in This Thesis . . . 28
  - 3.4.10 Running Time . . . 29
- 3.5 Experimental Methodology . . . 29

4 Evaluation of the Minimum-Degree Greedy Algorithm 31
- 4.1 Structural Stress Tests for Greedy . . . 31
- 4.2 Running Time of Greedy . . . 36

5 Evaluation of Lotka–Volterra Dynamics 37
- 5.1 Clustering Hypothesis . . . 37
- 5.2 Clustering Coefficient Theory . . . 38
- 5.3 Testing the Clustering Hypothesis . . . 39
- 5.4 Initial Conditions and Competing Maximal Sets . . . 43

6 Evaluation of Simulated Annealing **51**

7 Effect of Parameter Choices **58**
7.1 Changing τ . . . 58
7.1.1 Convergence Time . . . 59
7.2 Changing the Cooling Mechanism . . . 63
7.3 Changing the Penalty Parameter α . . . 65

8 Conclusion **71**
8.1 Main Findings . . . 71
8.2 Future Work . . . 72

Bibliography **74**

A Declaration of AI Usage **76**

B Proof of the correspondence between maximal independent sets and local minima **77**

C Proof of the correspondence between maximum independent sets and ground states **83**

D Additional Algorithm Results **85**
D.1 Greedy Additional Results . . . 85
D.2 Lotka–Volterra Additional Results . . . 87
D.3 Simulated Annealing Additional Results . . . 92

E Parameter Settings **99**

1 Introduction

Many problems in mathematics and computer science can be represented using graphs. A graph consists of vertices and edges, where the vertices represent objects and the edges represent relationships between them. Depending on the problem, one may be interested in finding a matching, a clique, a coloring, or an independent set. This thesis focuses on the maximum-independent-set problem. An independent set is a set of vertices in which no two vertices are connected by an edge, and the goal is to find an independent set of maximum size.

The maximum-independent-set problem is difficult to solve exactly on large graphs because the number of possible vertex subsets grows exponentially with the number of vertices. A graph with n vertices has 2^n possible vertex subsets, so a direct exhaustive search quickly becomes impractical as n increases. The decision version of the problem asks whether a graph contains an independent set of size at least k; this decision problem is one of Karp's classical NP-complete problems [18]. Therefore, for general graphs, we do not expect exact methods to scale well to large instances. Nevertheless, the problem is relevant in many settings where one wants to choose as many mutually compatible objects as possible. For example, the vertices could represent tasks, transmitters, people, or choices, and an edge could represent a conflict between two of them. An independent set then gives a collection of objects that can be chosen without conflicts.

The maximum-independent-set problem is also closely related to the maximum-clique problem. A clique in a graph G is an independent set in the complement graph $\overline{G}$, and conversely an independent set in G is a clique in $\overline{G}$ [10]. For this reason, work on maximum-clique algorithms is also relevant for independent-set problems [4]. In this thesis, the main focus remains on the maximum-independent-set problem, but clique instances can be handled by first taking the complement graph.

Because exact methods quickly become impractical for large graphs, heuristic algorithms are often used to find large independent sets. A heuristic does not guarantee that it finds a maximum independent set, but it can often find a good solution much faster than an exact method. This makes heuristics useful in practice, especially when the graph is too large for exhaustive search. However, the quality of a heuristic solution can depend strongly on the structure of the graph. For this reason, it is not enough to ask only which algorithm performs best on average. It is also important to understand when and why an algorithm fails.

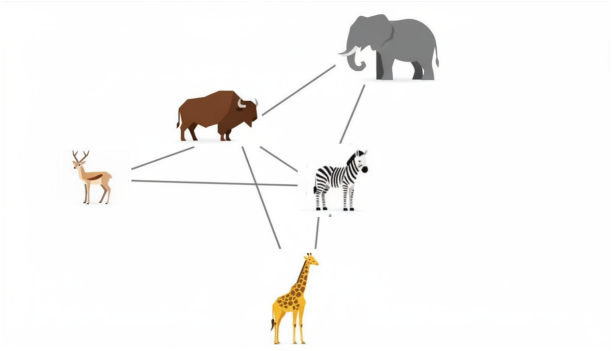
Greedy algorithms are a natural first approach. They build a solution step by step by making local choices, for example by selecting a vertex with small degree. Such methods are fast and easy to implement, which makes them useful as baseline algorithms [7, 17]. However, their simplicity also has a drawback: an early local choice can prevent the algorithm from finding a better solution later.

Simulated annealing is a different type of heuristic. It is inspired by physical cooling processes and was introduced as a general method for optimization problems with large search spaces [19]. The method is based on the Metropolis acceptance rule, which allows the algorithm to sometimes accept a move that temporarily worsens the objective value [25]. For the maximum-independent-set problem, this means that the algorithm searches through possible vertex selections using an energy function. Accepting temporary conflicts can help the algorithm move between different candidate independent sets, but the success of this search depends on the cooling schedule and on how strongly conflicts are penalized.

The third method studied in this thesis is based on Lotka–Volterra dynamics. The Lotka–Volterra equations originally come from mathematical biology, where they describe interactions between species, such as predator-prey dynamics or competition between species [21, 34]. This ecological interpretation gives a useful way to think about independent sets. In a competitive ecological system, species that rely on the same limited resource inhibit each other's growth, and strong competition may prevent them from coexisting at high population levels. Similarly, in a graph, two vertices connected by an edge cannot both be chosen in the same independent set. Figure 1.1 illustrates this analogy by showing how ecological competition can be represented as a graph. Figure 1.2 then shows an independent set in this simplified competition graph. In the Lotka–Volterra approach to the maximum-independent-set problem, each vertex is assigned a value that evolves over time, while neighboring vertices suppress each other through competitive interactions. Recent work has shown that such competitive dynamics can be used to find large independent sets in networks [27].



(a) Animals sharing the same environment.



(b) A simplified competition graph.

Figure 1.1: Ecological competition represented as a graph. In the photograph on the left, several animal species share the same environment and may compete for limited resources such as food, water, or territory. The diagram on the right (created by the author using Canva AI Image Generator) gives a simplified and idealized graph model of such competition, intended only for explanatory purposes. In this model, animals are represented by vertices, and an edge indicates direct competition between two animals. In this setting, an independent set represents a group of animals with no direct competition between them. The graph is not intended to be an accurate ecological model of the specific animals shown in the photograph. Image (a) adapted from Pixabay [30]; image (b) created by the author.

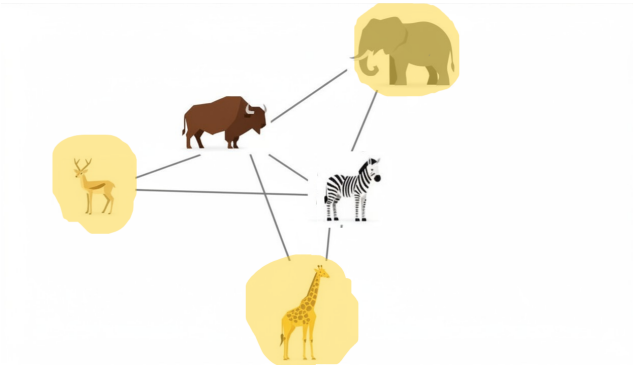


Figure 1.2: Image created by the author using Canva AI Image Generator. Example of an independent set in the simplified ecological competition graph. The highlighted animals form an independent set, because no two of them are connected by an edge. In ecological terms, this means that these selected animals do not directly compete with each other in the simplified model.

The motivation for this thesis is to study heuristic algorithms for the maximum-independent-set problem in a more diagnostic way. Instead of only comparing the final solution sizes, we also want to understand which graph structures and parameter choices make an algorithm unreliable. This is useful because two algorithms can return similar solution sizes for one graph family but fail for very different reasons on another. Identifying these failure mechanisms gives more information than a simple benchmark: it helps indicate when a heuristic should be trusted, when it should be used with caution, and which parameter choices may improve or worsen its behavior.

1.1. Research Question

In this thesis, performance mainly refers to the size and quality of the independent set returned by an algorithm. When the exact optimum is known, the returned solution can be compared with the true maximum independent set size. We also consider whether the output is maximum, maximal but not maximum, or non-maximal. For the parameter experiments, performance also includes how parameter choices affect convergence behavior and the ability of the algorithm to avoid poor local outcomes.

The main research question is:

> **RQ.** How do graph structure and algorithmic parameter choices affect the performance of greedy, simulated annealing, and Lotka–Volterra algorithms for the maximum-independent-set problem?

To answer this question, we consider the following subquestions:

SQ1. Which graph families cause each algorithm to perform significantly worse than the others, and how can this behavior be explained from the structure of the graph?

SQ2. How and why do parameter choices, such as the competition parameter τ, the penalty parameter α, and the cooling schedule, affect the performance of the algorithms, especially on graph families where the algorithms perform poorly?

1.2. Reading Guide

The aim of this thesis is to be self-contained and to answer the research question stated in Section 1.1.

- In Chapter 1, we introduce the maximum-independent-set problem, motivate the use of heuristic algorithms, and state the research question.
- In Chapter 2, we give the theoretical background needed for the project. This includes basic graph theory, the different graph types used in the simulations, and the relevant complexity theory.
- In Chapter 3, we describe the algorithms studied in this thesis. We first introduce the exact solver and the greedy algorithm. We then discuss the Lotka–Volterra dynamics and simulated annealing, including their theoretical motivation, implementation, and running time. This chapter also explains the experimental methodology used in the numerical comparisons.
- In Chapter 4, we investigate cases where the greedy algorithm performs poorly. In particular, we study how the performance of the greedy algorithm depends on the graph structure.
- In Chapter 5, we study cases where Lotka–Volterra dynamics perform poorly. We focus on the role of clustering and explain when the dynamics can converge to maximal independent sets that are not maximum.
- In Chapter 6, we discuss cases where simulated annealing performs poorly and analyze why the algorithm may fail to find good solutions.
- In Chapter 7, we study how parameter choices affect the algorithms. In particular, we consider the competition strength in the Lotka–Volterra method, as well as the cooling mechanism and penalty term in simulated annealing.
- In Chapter 8, we summarize the main conclusions of the thesis and explain how the results answer RQ, SQ1, and SQ2. We also discuss possible improvements.

2

Theoretical Background

Before studying algorithms for the maximum-independent-set problem, we introduce the graph theory language and graph families used throughout this thesis. The goal of this chapter is not to give a complete overview of graph theory, but to collect the definitions that are needed for the algorithms and experiments later on.

2.1. Basic Graph Theory

A graph is a pair

$$G = (V, E),$$

where V is a set of vertices and E is a set of edges [10, 35]. In this thesis, all graphs are finite, simple, and undirected. This means that the graph has finitely many vertices, contains no self-loops or multiple edges between two vertices, and that edges have no direction. Thus, an edge between two distinct vertices $u, v \in V$ is written as

$$\{u, v\} \in E,$$

where the order does not matter.

Two vertices u and v are called adjacent if they are connected by an edge. The neighbors of a vertex v are all vertices adjacent to v, and the neighborhood of v is denoted by

$$N(v) = \{u \in V : \{u, v\} \in E\}.$$

The degree of v, denoted by $\deg(v)$, is the number of neighbors of v:

$$\deg(v) = |N(v)|.$$

If the vertices are labeled as

$$V = \{1, \dots, n\},$$

then the adjacency matrix $A \in \mathbb{R}^{n \times n}$ is defined by

$$A_{ij} = \begin{cases} 1, & \text{if } \{i, j\} \in E, \\ 0, & \text{otherwise.} \end{cases}$$

For a simple graph, the diagonal entries are zero, because vertices are not connected to themselves. For an undirected graph, the adjacency matrix is symmetric, because $\{i, j\} \in E$ if and only if $\{j, i\} \in E$ [5, 35].

A walk in a graph is a sequence of vertices

$$v_0, v_1, \dots, v_k$$

such that every consecutive pair is connected by an edge:

$$\{v_{i-1}, v_i\} \in E \quad \text{for all } i = 1, \dots, k.$$

A path is a walk in which all vertices are distinct, and the length of a path is the number of edges it contains. A cycle is a closed path, meaning that it starts and ends at the same vertex, while all other vertices in the sequence are distinct. A graph is connected if there exists a path between any two distinct vertices in the graph [10, 35].

A clique is a set of vertices in which every pair of vertices is connected by an edge [10]. An example is shown in Figure 2.1.

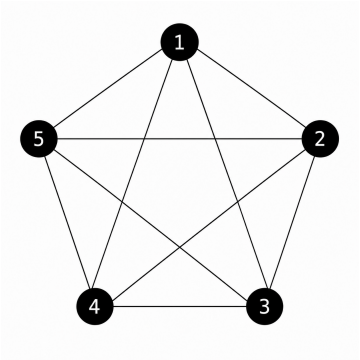


Figure 2.1: Example of a clique. Every pair of vertices in the graph is connected by an edge, so the graph forms a clique.

An independent set is a set of vertices in which no two vertices are connected by an edge [10, 35]. Thus, a set $S \subseteq V$ is independent if

$$\{u, v\} \notin E \quad \text{for all distinct } u, v \in S.$$

Figure 2.2 shows an example.

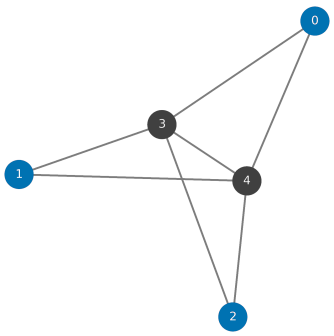


Figure 2.2: Example of an independent set. The highlighted blue vertices form an independent set because no two selected vertices are adjacent.

An independent set $S \subseteq V$ is called maximal if it cannot be extended by adding another vertex while remaining independent [10, 35]. Equivalently, every vertex outside S has at least one neighbor in S:

$$\forall v \in V \setminus S, \quad N(v) \cap S \neq \emptyset.$$

A maximal independent set is not necessarily the largest independent set in the graph.

A maximum independent set is an independent set of largest possible size. In other words, among all independent sets in the graph, it has maximum cardinality $|S|$. This largest possible value of $|S|$ is called the independence number of the graph. Figure 2.3 illustrates the difference between a maximal independent set and a maximum independent set.

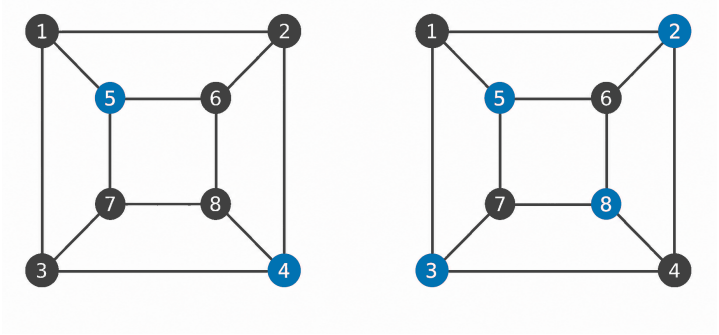


Figure 2.3: Comparison between a maximal and a maximum independent set. The highlighted blue vertices on the left form a maximal independent set: no additional vertex can be added while preserving independence. However, it is not the largest possible independent set. The highlighted vertices on the right form a maximum independent set, which has the largest possible cardinality among all independent sets in the graph.

The maximum-independent-set problem asks for a maximum independent set of a given graph. In this thesis, we study algorithms that try to find such sets, either exactly for small graphs or heuristically for larger graphs.

2.2. Graph Types

The experiments in this thesis use several graph families. These families were chosen because they represent different structural situations that can affect the behavior of the algorithms. Some graphs are useful as simple reference cases, while others isolate specific properties such as density, degree variation, complementarity, regularity, or local clustering.

A complete graph is a graph in which every pair of distinct vertices is connected by an edge [10, 35]. In a complete graph, the largest independent set has size 1, since every pair of vertices is adjacent. An example is shown in Figure 2.4.

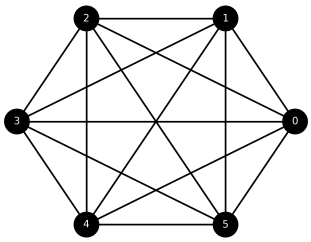


Figure 2.4: Example of a complete graph. Every pair of distinct vertices is connected by an edge, so any independent set can contain at most one vertex.

A tree is a connected graph that contains no cycles [10, 35]. Equivalently, a tree on n vertices has exactly $n - 1$ edges and is connected. Figure 2.5 shows an example.

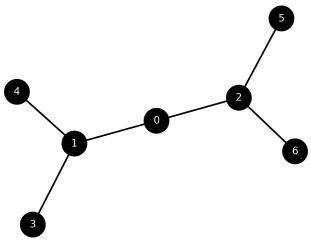


Figure 2.5: Example of a tree graph. The graph is connected and contains no cycles.

A graph is bipartite if its vertex set can be divided into two disjoint parts such that every edge has

one endpoint in each part [10]. In other words, there are no edges between vertices within the same part. An example is shown in Figure 2.6.

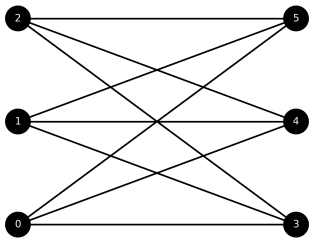


Figure 2.6: Example of a bipartite graph. The vertices can be split into two parts, with edges only running between the two parts and not within either part.

The complement of a graph $G = (V, E)$, denoted by $\overline{G}$, is the graph with the same vertex set V, where two distinct vertices are adjacent in $\overline{G}$ exactly when they are not adjacent in G [10]. Thus,

$$\{u, v\} \in E(\overline{G}) \quad \Leftrightarrow \quad u \neq v \text{ and } \{u, v\} \notin E(G).$$

This construction is useful because a clique in G corresponds to an independent set in $\overline{G}$. Figure 2.7 gives an example of a graph and its complement.

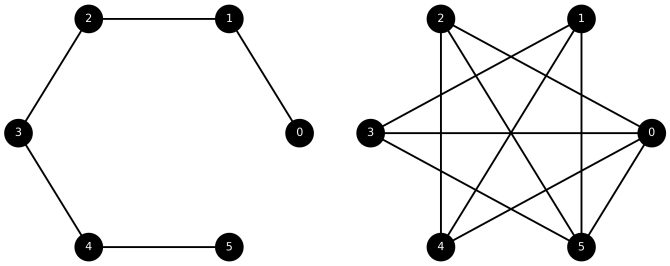


Figure 2.7: Example of a graph and its complement. Edges that are present in the original graph are absent in the complement graph, and edges that are absent in the original graph are present in the complement graph.

There are two common random graph models associated with Erdős–Rényi graphs. In this thesis, we use the $G(n, p)$ model: a graph on n vertices in which each possible edge is included independently with probability p [2, 12]. This is different from the $G(n, m)$ model, where exactly m edges are chosen uniformly at random. Erdős–Rényi graphs are useful as a random baseline because they do not impose geometric or regular structure.

A d-regular graph is a graph in which every vertex has degree d. A random regular graph is chosen uniformly at random from the set of all d-regular graphs on n vertices [2, 36]. Random regular graphs are useful because all vertices have the same degree, so they remove simple degree variation from the input. This is especially relevant when testing algorithms whose choices depend on degree. Figure 2.8 shows an example.

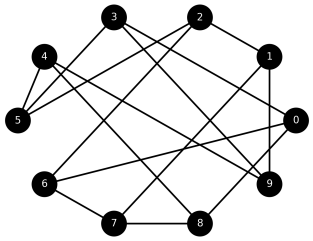


Figure 2.8: Example of a random regular graph. Every vertex has the same degree, so the graph does not contain low-degree or high-degree vertices.

A random geometric graph is constructed by placing vertices independently and uniformly at random in the unit square $[0, 1]^2$. Two vertices are connected by an edge if their Euclidean distance is at most a given radius r [29]. Random geometric graphs are useful because they contain spatial structure and local neighborhoods. This often creates clustering, which plays an important role in the Lotka–Volterra experiments. An example is shown in Figure 2.9.

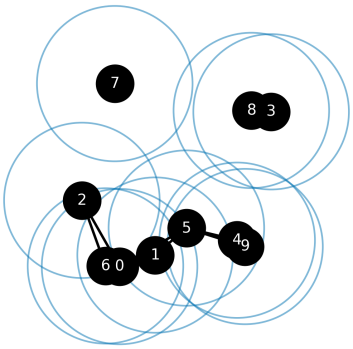


Figure 2.9: Example of a random geometric graph with connection radius $r = 0.35$. Vertices are placed in the plane, and an edge is drawn between two vertices whenever their Euclidean distance is at most r. The blue circles show the connection radius around each vertex, illustrating the local neighborhoods that determine the edge set.

Complete graphs and bipartite graphs are useful because their independent-set structure is easy to interpret. Trees give sparse connected examples without cycles. Complement graphs are included because they connect the maximum-independent-set problem to the maximum-clique problem. Erdős–Rényi graphs provide a standard random baseline. Random regular graphs remove degree variation, which is useful when studying algorithms that rely on degree information. Random geometric graphs introduce spatial structure and local clustering, which are important in several of the later experiments.

The more specialized graph constructions used to test particular failure mechanisms are introduced later, in the chapters where they are used. This keeps the present section focused on standard graph families and background terminology.

2.3. Complexity Theory

To understand why heuristic algorithms are useful for the maximum-independent-set problem, we briefly introduce some ideas from complexity theory. Complexity theory studies how the computational effort needed to solve a problem grows as the input size increases. In this thesis, the input is usually a graph, so the input size is related to the number of vertices and edges.

The maximum-independent-set problem is easy to state, but difficult to solve exactly on large graphs. A graph with n vertices has 2^n possible vertex subsets. An exhaustive search could check each subset and keep the largest one that is independent, but the number of subsets doubles whenever one vertex is added. This exponential growth quickly makes exhaustive search impractical.

Although this thesis studies the optimization version of the maximum-independent-set problem, its formal computational difficulty is usually described through the corresponding decision problem. A decision problem is a problem whose answer is either “yes” or “no” [13, 31]. Instead of asking for a largest independent set, the decision version asks:

$$\text{Does } G \text{ contain an independent set of size at least } k?$$

This formulation is useful because P, NP, and NP-complete are usually defined for decision problems, while the optimization version is described through its corresponding decision version. Before we define these classes, we must explain the meaning of the running time of an algorithm first.

The running time of an algorithm is described as a function of the input size. In complexity theory, we usually count the number of elementary computational steps rather than the actual time measured in seconds, since the measured time also depends on hardware, programming language, and implementation details. An algorithm runs in polynomial time if the number of steps is bounded by a polynomial

in the input size [13, 20]. Examples of polynomial running times are

$$O(n), \qquad O(n^2), \qquad O(n^3).$$

Here, O-notation describes an upper bound on the growth rate of a function. We write

$$f(n) = O(g(n))$$

if there exist constants $C > 0$ and n_0 such that

$$f(n) \leq Cg(n) \quad \text{for all } n \geq n_0.$$

The class P consists of all decision problems that can be solved in polynomial time [31]. In complexity theory, polynomial time is often used as a formal notion of efficient computation. This does not mean that every polynomial-time algorithm is practical, but it gives a way to distinguish algorithms whose running time grows moderately from algorithms with exponential growth.

The class NP consists of decision problems for which a "yes" answer can be verified in polynomial time, given a suitable certificate [13, 31]. A certificate is extra information that helps verify a proposed solution. For the independent-set decision problem, a certificate is a set of vertices S. To check the certificate, we verify that

$$|S| \geq k$$

and that no two vertices in S are adjacent. This can be done in polynomial time, so the independent-set decision problem belongs to NP.

A polynomial-time reduction is a way to transform one decision problem into another decision problem in polynomial time [13, 31]. If problem A can be reduced to problem B, then an efficient algorithm for B would also give an efficient algorithm for A. Reductions are used to compare the difficulty of computational problems.

A problem is called NP-hard if every problem in NP can be reduced to it in polynomial time. Informally, an NP-hard problem is at least as hard as the hardest problems in NP. A decision problem is called NP-complete if it is both in NP and NP-hard [13, 18]. Thus, NP-complete problems are the hardest problems inside NP.

The decision version of the maximum-independent-set problem is one of Karp's classical NP-complete problems [18]. The optimization version asks for an independent set of largest possible size. If we could solve this optimization problem exactly in polynomial time, then we could also solve the decision version in polynomial time by checking whether the maximum size found is at least k. Therefore, the optimization version is NP-hard.

The maximum-independent-set problem is also closely related to the maximum-clique problem. The maximum-clique problem asks for a clique of largest possible size. For a graph $G = (V, E)$, the complement graph $\overline{G}$ has the same vertex set, and two distinct vertices are adjacent in $\overline{G}$ exactly when they are not adjacent in G. A set $S \subseteq V$ is a clique in G if and only if it is an independent set in $\overline{G}$ [10]. Therefore, maximum-clique instances can be handled by taking the complement graph and then applying algorithms for the maximum-independent-set problem.

This complexity-theoretic background explains why this thesis focuses on heuristic algorithms. Exact algorithms can be used as benchmarks on small graphs, but they do not scale well to large instances. Heuristic algorithms do not guarantee that they find a maximum independent set on every graph, but they can often find large independent sets much faster. The central question of this thesis is therefore not only whether these algorithms work, but also on which graph structures they work well or fail.

3

Algorithms and Experimental Methodology

This chapter introduces the algorithms and implementation choices used in this thesis. The methods are chosen to play different roles in the study of the maximum-independent-set problem. Exhaustive enumeration is used as an exact benchmark on small graphs. The minimum-degree greedy algorithm is used as a fast and simple baseline. Simulated annealing is used as a stochastic energy-based heuristic, while Lotka–Volterra dynamics provide a continuous-dynamical approach to the same discrete problem. Comparing these methods gives a way to study not only which algorithm finds large independent sets, but also which structural features of a graph make each method fail.

The chapter separates standard algorithmic background from choices made specifically for this thesis. General definitions and known theoretical results are introduced as background. Sections and subsections that describe numerical parameters, thresholding, post-processing, and experimental organization describe the implementation used in this thesis.

3.1. Exhaustive Enumeration

To obtain the true maximum independent set on small graphs, we use exhaustive enumeration. This is the most direct exact method: it checks all possible subsets of vertices and returns one of largest size that is independent.

Concretely, for a graph with n vertices, there are 2^n possible vertex subsets. For each subset, we check whether it forms an independent set by verifying that no two vertices in the subset are adjacent. This feasibility check can be performed in $O(n^2)$. Among all feasible independent subsets, we then select one with the largest size.

The total running time is

$$O(2^n n^2),$$

which grows exponentially with the number of vertices. Table 3.1 illustrates how quickly the number of subsets increases.

Number of vertices n	**Number of subsets** 2^n
10	1,024
15	32,768
20	1,048,576
25	33,554,432
30	1,073,741,824

Table 3.1: Growth of the number of subsets checked by exhaustive enumeration. Even for moderately sized graphs, the number of possible subsets becomes very large.

This exponential growth reflects the fact that the maximum-independent-set problem is NP-hard, and no polynomial-time algorithm is known for solving it exactly on general graphs [13]. Therefore,

exhaustive enumeration is only used as a benchmark on small graphs. In this thesis, we use it for graphs with $n \leq 20$.

3.2. Minimum-Degree Greedy Algorithm

The greedy method used in this thesis is a minimum-degree greedy algorithm. It constructs an independent set step by step by repeatedly selecting a vertex of minimum degree in the remaining graph. Greedy approaches for independent-set problems are widely used as baseline methods because they are simple and efficient [17, 20]. The particular greedy rule used here is the following.

1. Start with the full graph $G = (V, E)$ and initialize the solution set $S = \emptyset$.
2. While the vertex set of the current graph is not empty:
   (a) Select a vertex $v \in V$ with minimum degree.
   (b) Add v to the solution set:
   $$S = S \cup \{v\}.$$
   (c) Remove v and all of its neighbors from the graph:
   $$V = V \setminus \big(\{v\} \cup N(v)\big).$$
   (d) Update the graph by removing all edges incident to the removed vertices.
3. Return the set S.

Once a vertex is selected, both the vertex itself and all of its neighboring vertices are removed from the current graph. This guarantees that no two selected vertices are adjacent in the original graph. In case of a tie, where several vertices have the same minimum degree, the implementation chooses the vertex with the smallest label, where the label is the numerical index assigned to the vertex.

By construction, the algorithm always returns an independent set. Moreover, it returns a maximal independent set, since every unselected vertex is either selected or removed as a neighbor of a selected vertex. However, a maximal independent set is not necessarily a maximum independent set, so the minimum-degree greedy algorithm does not guarantee optimality [35].

The greedy algorithm runs in $O(n^2)$ time. Checking whether a returned set of size k is independent takes $O(k^2)$ [20].

3.3. Lotka–Volterra Dynamics

In contrast to exhaustive enumeration and the greedy algorithm, which operate directly on the graph, the Lotka–Volterra approach reformulates the problem as a continuous dynamical system.

More precisely, each vertex $i \in V$ is assigned a time-dependent variable $x_i(t)$. In this thesis, the initial conditions are chosen nonnegative, so $x_i(0) \geq 0$ for all i. For the Lotka–Volterra system used here, nonnegativity is preserved over time: if $x_i(t) = 0$, then the factor $x_i(t)$ in the differential equation makes $\dot{x}_i(t) = 0$, so the variable cannot cross below zero.

The value $x_i(t)$ can be interpreted as a continuous score for vertex i. Large values indicate that the vertex is strongly present in the current configuration, while values close to 0 indicate that the vertex is suppressed. The method therefore does not make binary decisions immediately. Instead, the continuous variables evolve over time, and an independent set is extracted from the final state.

The dynamics are governed by the generalized Lotka–Volterra equations

$$\frac{dx_i(t)}{dt} = x_i(t)(r_i - (Mx(t))_i), \tag{3.1}$$

which originate from ecological models describing competing species [26]. Here, r_i represents intrinsic growth and the matrix M encodes interactions between vertices. Expanding the interaction term gives

$$\frac{dx_i(t)}{dt} = x_i(t)\left(r_i - \sum_{j=1}^{n} M_{ij}x_j(t)\right).$$

This structure reflects a balance between growth and competition. In the classical Lotka–Volterra competition model, each variable represents the population size of a species in an ecosystem. A species can grow when resources are available, but its growth is limited by competition with itself and with other species.

In the graph setting, we use the same idea: each vertex is treated like a competing species, and an edge between two vertices means that the corresponding species compete directly. Therefore, two adjacent vertices should not both survive with large values if the final set is to represent an independent set.

To incorporate the graph structure, the interaction matrix is defined as

$$M = I + \tau A, \tag{3.2}$$

where A is the adjacency matrix. Then

$$M_{ij} = \begin{cases} 1, & \text{if } i = j, \\ \tau, & \text{if } i \neq j \text{ and } \{i,j\} \in E, \\ 0, & \text{otherwise.} \end{cases}$$

Here, $\tau > 0$ controls the strength of competition between adjacent vertices. Setting $r_i = 1$, the system simplifies to

$$\frac{dx_i(t)}{dt} = x_i(t)\left(1 - x_i(t) - \tau \sum_{j \sim i} x_j(t)\right). \tag{3.3}$$

For nonnegative initial conditions, the solution remains nonnegative. Indeed, the equation has the form

$$\dot{x}_i(t) = x_i(t) g_i(t),$$

where

$$g_i(t) = 1 - x_i(t) - \tau \sum_{j \sim i} x_j(t).$$

Therefore,

$$x_i(t) = x_i(0) \exp\left(\int_0^t g_i(s)\, ds\right).$$

The exponential factor is always positive, regardless of the sign of $g_i(t)$. Therefore, if $x_i(0) > 0$, then $x_i(t)$ remains positive for all later times for which the solution exists. If $x_i(0) = 0$, then the factor $x_i(t)$ in the differential equation gives $\dot{x}_i(t) = 0$, so $x_i(t)$ remains equal to zero. Hence a trajectory that starts in $\mathbb{R}^n_{\geq 0}$ cannot leave it, and the nonnegative orthant is forward invariant.

Again, $x_i(t)$ represents the value assigned to vertex i at time t. Here $j \sim i$ means that j is a neighbor of i, or equivalently that $\{i,j\} \in E$. The term 1 represents the intrinsic growth rate of $x_i(t)$, the term $-x_i(t)$ limits its own growth, and the term

$$-\tau \sum_{j \sim i} x_j(t)$$

represents suppression from neighboring vertices. Thus, if neighboring vertices have large values, the value of $x_i(t)$ is pushed downward. Specifically, a vertex value decreases when

$$x_i(t) + \tau \sum_{j \sim i} x_j(t) > 1.$$

Thus, neighboring vertices can suppress each other when their values become sufficiently large. Later, we use the condition $\tau > 1$ to connect this competition to the independence condition.

3.3.1. Equilibria and Stability

To understand why the Lotka–Volterra dynamics can be used for the maximum-independent-set problem, we study the long-term behavior of the system. In particular, we look at equilibria, also called fixed points. These are points where the variables no longer change over time.

A fixed point x^* satisfies

$$\frac{dx_i}{dt} = 0 \quad \text{for all } i.$$

Using the Lotka–Volterra equation, this means that

$$x_i^* \left(1 - x_i^* - \tau \sum_{j \sim i} x_j^* \right) = 0 \quad \text{for all } i.$$

Therefore, for each vertex i, either

$$x_i^* = 0,$$

or

$$1 - x_i^* - \tau \sum_{j \sim i} x_j^* = 0.$$

In the first case, the value assigned to vertex i is zero, so the vertex is not selected in the final set. In the second case, the value of vertex i is balanced by its own value and by the values of its neighboring vertices.

This suggests a connection to independent sets. During the evolution of the Lotka–Volterra system, adjacent vertices inhibit each other's growth through the competition term

$$-\tau \sum_{j \sim i} x_j.$$

Therefore, configurations in which neighboring vertices are both large are discouraged. If the system reaches a stable final state, vertices with values close to 1 are interpreted as selected, while vertices with values close to 0 are interpreted as excluded.

However, not every fixed point describes the behavior we expect to observe in the long term. Some fixed points are unstable, meaning that a small perturbation moves the system away from them. Stable fixed points, on the other hand, persist under small perturbations. Therefore, the connection between Lotka–Volterra dynamics and independent sets is not based only on fixed points, but on stable fixed points.

3.3.2. Why the Condition $\tau > 1$ Matters

The connection between the Lotka–Volterra system and independent sets is not only heuristic. It is based on a theorem by Mooij and Kryven [27]. For the graph-based Lotka–Volterra system with interaction matrix

$$M = I + \tau A,$$

where A is the adjacency matrix of the graph and $\tau > 1$, the stable states selected by the dynamics correspond to maximal independent sets of the graph. More precisely, the set of vertices with positive limiting value forms a maximal independent set. This result is the theoretical reason why Lotka–Volterra dynamics can be used as a method for finding independent sets.

The parameter τ controls the strength of competition between adjacent vertices. If τ is small, neighboring vertices only weakly suppress each other, so it may still be possible for adjacent vertices to coexist in a stable equilibrium. If τ is large, competition between neighbors becomes stronger.

The threshold $\tau > 1$ can be understood by considering the simplest possible graph: two adjacent vertices. The following two-vertex calculation does not prove the general theorem, but it illustrates the local mechanism behind the condition $\tau > 1$. It shows that, when two vertices are adjacent, a state in which both vertices coexist with positive value is unstable for $\tau > 1$. The system becomes

$$\frac{dx_1}{dt} = x_1(1 - x_1 - \tau x_2),$$

$$\frac{dx_2}{dt} = x_2(1 - x_2 - \tau x_1).$$

This system has a coexistence equilibrium where both adjacent vertices have the same positive value:

$$x_1 = x_2 = \frac{1}{1+\tau}.$$

At this point, both adjacent vertices are present. For the independent set problem, this is not the type of stable outcome we want, because adjacent vertices should not both be selected.

Figure 3.1 illustrates the phase portrait for the case $\tau = 2$. The figure is included only as an illustration of the two-dimensional system. It shows the nullclines, the coexistence equilibrium, and the direction in which trajectories move. The general graph result is not proved by this picture, but by the theorem of Mooij and Kryven stated later.

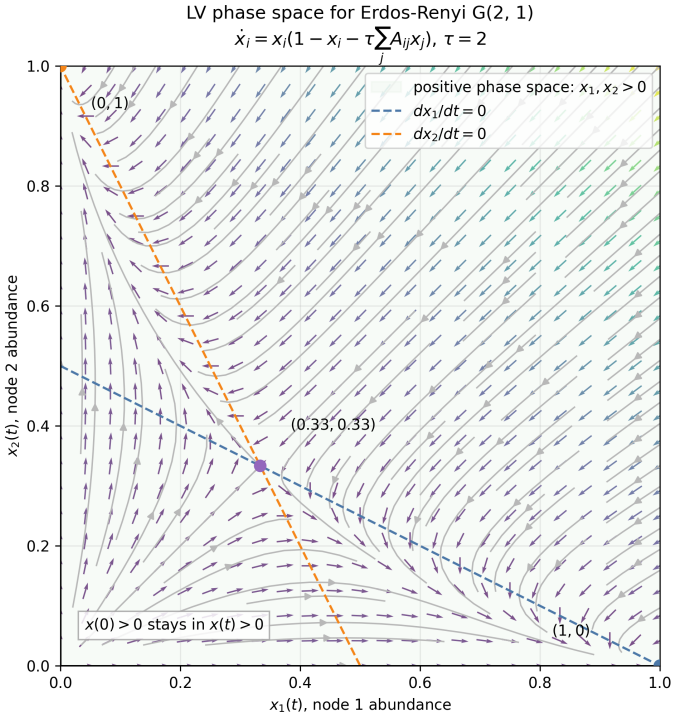


Figure 3.1: Phase portrait of the Lotka–Volterra system for two adjacent vertices with $\tau = 2$. The dashed lines indicate the nontrivial nullclines $\dot{x}_1 = 0$ and $\dot{x}_2 = 0$. Their intersection at $(1/3, 1/3)$ is the coexistence equilibrium. The arrows indicate the direction of the vector field. This figure is meant as an illustration of the two-dimensional case, not as a proof of the general graph result.

We now check the stability of the coexistence equilibrium by linearizing the system at this point. Let

$$f_1(x_1, x_2) = x_1(1 - x_1 - \tau x_2), \qquad f_2(x_1, x_2) = x_2(1 - \tau x_1 - x_2).$$

The Jacobian matrix is

$$J(x_1, x_2) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 1 - 2x_1 - \tau x_2 & -\tau x_1 \\ -\tau x_2 & 1 - \tau x_1 - 2x_2 \end{pmatrix}.$$

At the coexistence equilibrium

$$x_1 = x_2 = \frac{1}{1+\tau},$$

this becomes

$$J = \begin{pmatrix} 1 - \frac{2}{1+\tau} - \frac{\tau}{1+\tau} & -\frac{\tau}{1+\tau} \\ -\frac{\tau}{1+\tau} & 1 - \frac{\tau}{1+\tau} - \frac{2}{1+\tau} \end{pmatrix}.$$

Since

$$1 - \frac{2+\tau}{1+\tau} = \frac{1+\tau-2-\tau}{1+\tau} = -\frac{1}{1+\tau},$$

we obtain

$$J = \begin{pmatrix} -\frac{1}{1+\tau} & -\frac{\tau}{1+\tau} \\ -\frac{\tau}{1+\tau} & -\frac{1}{1+\tau} \end{pmatrix}.$$

This matrix has the form

$$\begin{pmatrix} a & b \\ b & a \end{pmatrix},$$

whose eigenvalues are $a+b$ and $a-b$. Here

$$a = -\frac{1}{1+\tau}, \qquad b = -\frac{\tau}{1+\tau}.$$

Therefore,

$$\lambda_1 = a + b = -\frac{1}{1+\tau} - \frac{\tau}{1+\tau} = -1,$$

and

$$\lambda_2 = a - b = -\frac{1}{1+\tau} + \frac{\tau}{1+\tau} = \frac{\tau-1}{1+\tau}.$$

For $\tau > 1$, we have $\lambda_2 > 0$. Hence, the coexistence equilibrium of this two-vertex graph is unstable. Thus, in this local example, two adjacent vertices cannot both remain positive in a stable state when $\tau > 1$. This gives intuition for the general Mooij–Kryven theorem stated in Theorem 1.

3.3.3. Lyapunov Function and Energy Landscape

The previous subsections explained the LV dynamics by looking at fixed points and stability. We can also understand the same system from an energy perspective. Instead of only asking where the differential equation stops, we can ask whether the system is moving downhill in some landscape. This is where the Lyapunov function becomes useful.

A Lyapunov function assigns a scalar value to every state x. This value can be interpreted as an energy-like quantity, but it is not necessarily a physical energy. Its key property is that it never increases along trajectories of the Lotka–Volterra system. Therefore, the Lyapunov function can be used to study the long-term behavior and stability of the dynamics. In other words, as the variables $x_i(t)$ evolve over time, the system behaves as if it is moving downhill until it reaches a point where the energy can no longer decrease. This gives another way to understand why the LV dynamics settle into stable configurations.

Let $G = (V, E)$ be a finite, simple, undirected graph with

$$V = \{1, \dots, n\}.$$

Let A denote the adjacency matrix of G, and define

$$M = I + \tau A,$$

as in Equation 3.2, with $\tau > 1$. We consider the function

$$V(x) = \frac{1}{2} x^T M x - \mathbf{1}^T x \tag{3.4}$$

on the domain

$$D = \mathbb{R}^n_{\geq 0}.$$

Thus all coordinates satisfy $x_i \geq 0$.

For a subset $S \subseteq V$, we denote by $\chi_S \in \{0,1\}^n$ its characteristic vector, defined by

$$(\chi_S)_i = \begin{cases} 1, & i \in S, \\ 0, & i \notin S. \end{cases}$$

We first show why V is a Lyapunov function for the LV system. This means that, if $x(t)$ follows the LV dynamics, then the value $V(x(t))$ is non-increasing over time. In simple terms, the dynamics move downhill in the landscape defined by V. After that, we explain why this landscape is useful for the maximum-independent-set problem: local minima of V correspond to maximal independent sets, while global minima, also called ground states, correspond to maximum independent sets.

3.3.4. The Lyapunov Function Decreases Along LV Trajectories

Recall that the LV system with $r = \mathbf{1}$ is

$$\frac{dx_i}{dt} = x_i \left(1 - (Mx)_i\right),$$

as in Equation 3.3.

Proof. Since G is undirected, the adjacency matrix A in Equation 3.2 is symmetric. Therefore M is also symmetric.

First, we compute the gradient of V. Since

$$V(x) = \frac{1}{2} x^T M x - \mathbf{1}^T x,$$

we have

$$\nabla\left(\frac{1}{2} x^T M x\right) = \frac{1}{2}(M + M^T)x.$$

For an undirected graph, the adjacency matrix A is symmetric. Hence $M = I + \tau A$ is also symmetric, and therefore

$$\nabla V(x) = Mx - \mathbf{1}.$$

Thus the i-th component of the gradient is

$$(\nabla V(x))_i = (Mx)_i - 1.$$

We now look at how $V(x(t))$ changes over time. By the chain rule,

$$\frac{d}{dt} V(x(t)) = \nabla V(x(t))^T \frac{dx(t)}{dt}.$$

Substituting the expression for the gradient gives

$$\frac{d}{dt} V(x(t)) = \sum_{i=1}^{n} \left((Mx(t))_i - 1\right) \frac{dx_i(t)}{dt}.$$

Using the LV equation,

$$\frac{dx_i(t)}{dt} = x_i(t) \left(1 - (Mx(t))_i\right),$$

we obtain

$$\frac{d}{dt} V(x(t)) = \sum_{i=1}^{n} \left((Mx(t))_i - 1\right) x_i(t) \left(1 - (Mx(t))_i\right).$$

This becomes

$$\frac{d}{dt} V(x(t)) = -\sum_{i=1}^{n} x_i(t) \left(1 - (Mx(t))_i\right)^2.$$

Because $x_i(t) \geq 0$ for all i, and squares are always nonnegative, every term in the sum is nonnegative before the minus sign. Therefore

$$\frac{d}{dt} V(x(t)) \leq 0.$$

This proves that $V(x(t))$ is non-increasing along LV trajectories. The energy can decrease, or it can stay constant, but it cannot increase. The energy stops decreasing only when

$$x_i(t)\,(1 - (Mx(t))_i) = 0 \quad \text{for all } i,$$

which is exactly the fixed point condition of the LV system.

This is why V is a Lyapunov function for the LV dynamics. It shows that the system follows a downhill direction in the energy landscape until it reaches an equilibrium. □

3.3.5. Maximal Independent Sets and Local Minima

The next question is what the low values of this landscape represent in terms of the graph. The Lyapunov function V is defined on the continuous space $\mathbb{R}^n_{\geq 0}$, but subsets of vertices can be represented inside this space by binary indicator vectors. For a subset $S \subseteq V$, let $\chi_S \in \{0,1\}^n$ denote the vector whose i-th entry is 1 if $i \in S$ and 0 otherwise. In this way, χ_S represents the choice of selecting exactly the vertices in S.

We are interested in which of these binary indicator vectors are local minima of the continuous function V. Here, a local minimum means that V cannot be decreased by making sufficiently small perturbations of the vector inside $\mathbb{R}^n_{\geq 0}$. Thus, although the independent-set problem is discrete, the notion of local minimum is taken with respect to the continuous domain of the Lotka–Volterra system.

In Appendix B, we prove, using the notation of this thesis, that for every subset $S \subseteq V$,

$$S \text{ is a maximal independent set} \quad \Leftrightarrow \quad \chi_S \text{ is a local minimum of } V \text{ on } \mathbb{R}^n_{\geq 0}.$$

3.3.6. Maximum Independent Sets and Ground States

We have shown that local minima of the Lyapunov function correspond to maximal independent sets. However, the maximum-independent-set problem asks for more than maximality: it asks for an independent set of largest possible size.

To capture optimality, we consider the global minima of V. A global minimum of V is also called a ground state. While local minima at binary indicator vectors correspond to maximal independent sets, ground states at binary indicator vectors correspond to maximum independent sets. In Appendix C, we prove, using the notation of this thesis, that

$$S \text{ is a maximum independent set} \quad \Leftrightarrow \quad \chi_S \text{ is a ground state of } V \text{ on } \mathbb{R}^n_{\geq 0}.$$

3.3.7. Theoretical Justification for the LV Method

The previous subsections introduced the Lyapunov function associated with the graph-based Lotka–Volterra system. In particular, we have seen that V is non-increasing along trajectories, and that its local minima at binary indicator vectors correspond to maximal independent sets. These results explain the connection between the continuous Lotka–Volterra dynamics and the discrete maximum-independent-set problem.

The central convergence result used in this thesis is due to Mooij and Kryven [26]. It states that, for $\tau > 1$, the Lotka–Volterra dynamics do not merely have a formal connection to independent sets: the limiting state of the dynamics encodes a maximal independent set.

Theorem 1 (Mooij–Kryven theorem, [26])**.** *Let $G = (V, E)$ be a simple graph with n vertices and adjacency matrix A. Let $\tau > 1$ and define*

$$M = I + \tau A.$$

Consider the Lotka–Volterra system with $r = \mathbf{1}$ and initial condition $x_0 \in (0,1)^n$. Then the limit

$$x^* := \lim_{t \to \infty} x(t)$$

exists, and the set

$$S = \{v_i \in V : x_i^* = 1\}$$

is a maximal independent set of G.

The proof of this theorem is nontrivial and is not reproduced here. Its main ingredients are the Lyapunov function introduced above and an analysis of the limiting equilibria of the Lotka–Volterra system. The Lyapunov function gives a quantity that decreases along trajectories, which helps control the long-term behavior of the system. The fixed-point analysis then shows that, when $\tau > 1$, adjacent vertices cannot both be selected in a stable limiting state, while every unselected vertex must be blocked by at least one selected neighbor. Together, these arguments imply that the limiting selected set is independent and maximal.

For this thesis, Theorem 1 is the theoretical justification for using Lotka–Volterra dynamics as a heuristic for the maximum-independent-set problem. The theorem guarantees convergence to a maximal independent set, but not necessarily to a maximum independent set. The experiments in later chapters therefore investigate when this maximal independent set is also close to optimal, and when the dynamics become trapped in a suboptimal maximal independent set.

3.3.8. Numerical Implementation Used in This Thesis

To simulate the Lotka–Volterra dynamics, we solve the ODE numerically. In this project, we use the forward Euler method, which is one of the simplest time-stepping schemes for approximating the solution of an initial value problem [15].

Starting from the Lotka–Volterra ODE in Equation 3.1, we discretize time as

$$t_k = k\Delta t,$$

where Δt is the chosen time step. The forward Euler update is then given by

$$x^{k+1} = x^k + \Delta t \left[x^k \odot \left(r - Mx^k\right)\right],$$

where $\odot$ denotes component-wise multiplication. Equivalently, using the graph form of the equation with $r_i = 1$, this can be written component-wise as

$$x_i^{k+1} = x_i^k + \Delta t\, x_i^k \left(1 - x_i^k - \tau \sum_{j \sim i} x_j^k\right).$$

The initial condition is chosen as

$$x_i(0) \in (0, 1)$$

for every vertex i. The reason we avoid choosing $x_i(0) = 0$ is that the LV equation contains the multiplicative factor $x_i(t)$. Indeed, if $x_i(t) = 0$, then

$$\frac{dx_i(t)}{dt} = 0.$$

Therefore, once a variable reaches zero exactly, it stays zero forever. In ecological language, a species that is completely extinct cannot regrow in this model. In graph-theoretic language, starting a vertex value at zero would remove it from the competition before the algorithm has had a chance to decide whether it should belong to the independent set. Mooij also assumes initial conditions $x_0 \in (0, 1)^n$ for the main convergence result, so all coordinates initially lie strictly between 0 and 1. Moreover, the closed unit hypercube $[0, 1]^n$ is forward invariant for the LV system, meaning that trajectories starting with coordinates in $[0, 1]$ remain in $[0, 1]$ for all later times [26].

In our implementation, we choose

$$\Delta t = 0.01, \qquad t_{\max} = 50.$$

This means that the number of Euler steps is

$$N_{\text{steps}} = \frac{t_{\max}}{\Delta t} = \frac{50}{0.01} = 5000.$$

The choice $\Delta t = 0.01$ is small enough to follow the dynamics smoothly, while still being computationally manageable. However, because forward Euler is an explicit method, the time step cannot be chosen arbitrarily large. The component-wise update can be written as

$$x_i^{k+1} = x_i^k \left[1 + \Delta t \left(1 - x_i^k - \tau \sum_{j \sim i} x_j^k\right)\right].$$

Thus, even though the continuous Lotka–Volterra system preserves nonnegativity, the numerical Euler approximation may produce negative values if the factor in brackets becomes negative.

A sufficient condition for preserving nonnegativity at each Euler step is

$$1 + \Delta t \left(1 - x_i^k - \tau \sum_{j \sim i} x_j^k \right) \geq 0 \quad \text{for all } i.$$

If $x^k \in [0,1]^n$, then

$$x_i^k + \tau \sum_{j \sim i} x_j^k \leq 1 + \tau d_i,$$

where d_i is the degree of vertex i. Therefore, a conservative sufficient condition is

$$\Delta t \leq \frac{1}{\tau d_{\max}},$$

where $d_{\max}$ is the maximum degree of the graph. This is a CFL-type stability restriction for the explicit Euler discretization. In practice, the bound can be restrictive for dense graphs or large values of τ, so $\Delta t = 0.01$ should be understood as a numerical choice that must be checked for stability in the experiments. The value $t_{\max} = 50$ gives the system enough time to approach a stable equilibrium in the experiments. These parameters therefore balance accuracy, convergence, and running time.

The growth parameter is chosen as

$$r_i = 1$$

for all vertices i, following the setting used in Mooij's theorem for unweighted graphs [26]. This choice means that every vertex value has the same intrinsic growth rate, so all variables x_i are treated equally before graph competition is introduced. This is appropriate for the unweighted maximum-independent-set problem, where no vertex is preferred over another.

The running time of the LV method depends strongly on the numerical integration method. With forward Euler, the total cost is determined by the number of time steps and the cost of one update. If the adjacency matrix is stored densely, computing Ax^k costs $O(n^2)$ per time step. Thus, the total running time is

$$O\left(N_{\text{steps}} n^2\right) = O\left(\frac{t_{\max}}{\Delta t} n^2\right).$$

This shows an important difference between the LV method and the minimum-degree greedy algorithm. The greedy algorithm works directly on the graph through discrete graph operations, such as selecting a vertex and removing vertices from the current graph. The LV method also has discrete computational steps in the implementation, because the continuous differential equation is approximated by a numerical time-stepping method. However, these steps are not graph operations in the same sense. They are Euler steps used to follow an approximate trajectory of the Lotka–Volterra system.

Therefore, for fixed numerical parameters Δt and $t_{\max}$, the forward Euler implementation with a dense adjacency matrix has asymptotic running time $O(n^2)$ in the number of vertices. However, the hidden constant in this bound depends on the number of Euler steps,

$$N_{\text{steps}} = \frac{t_{\max}}{\Delta t}.$$

Thus, although Δt and $t_{\max}$ do not change the asymptotic dependence on n when they are fixed, they can have a large effect on the practical running time. A smaller time step gives more Euler steps and may give a more accurate trajectory, while a larger $t_{\max}$ gives the system more time to approach equilibrium. These numerical choices therefore affect the accuracy and practical cost of the LV method, even though they are hidden as constants in the O-notation with respect to n.

After the numerical integration has finished, the continuous output must be converted back into a discrete independent set. Since the theorem states that, for $\tau > 1$, the limit values satisfy $x_i^* \in \{0,1\}$ and the set $\{v_i \in V : x_i^* = 1\}$ is a maximal independent set, we interpret vertices with large final values as selected vertices [26]. In practice, the values may not be exactly zero or one, so a threshold is used to decide which vertices are included in the final solution.

3.3.9. Thresholding and Discretization Effects

In theory, the LV dynamics converge to a limit state x^* in which each component satisfies $x_i^* \in \{0, 1\}$ when $\tau > 1$, and the set $\{v_i \in V : x_i^* = 1\}$ forms a maximal independent set [26]. In practice, however, we only compute a numerical approximation of this limit. There are several sources of error. First, the simulation is stopped at a finite time $t_{\max}$, so the system may not yet have reached its exact equilibrium. Second, the forward Euler method introduces discretization error because it approximates the continuous trajectory using finite time steps. Finally, floating-point computations can introduce small rounding errors.

For these reasons, the final numerical values $x_i(t_{\max})$ are generally not exactly equal to 0 or 1. Instead, vertices that belong to the limiting independent set are expected to have values close to 1, while the remaining vertices are expected to have values close to 0. A threshold is therefore needed to convert the continuous numerical output into a discrete independent set.

To extract a discrete independent set from the continuous solution, we apply a thresholding procedure. In our implementation, we choose a small threshold $\theta = 0.001$, fixed throughout this thesis, and define

$$S_\theta = \{v_i \in V : x_i(t_{\max}) \geq \theta\}.$$

This choice reflects the idea that any vertex that has not been effectively suppressed by the dynamics, meaning that it remains significantly above zero, is considered part of the solution.

This differs from more symmetric threshold choices such as $\theta = 0.5$, where one explicitly separates values close to 1 from those close to 0. Instead, our approach is more permissive: we primarily distinguish between vertices that have been driven to near extinction and those that survive. This aligns with the ecological interpretation of the model, where species either die out or persist at a positive population level.

Besides the finite simulation time preventing the system from fully converging to its asymptotic equilibrium, the numerical integration method also introduces error. In particular, forward Euler is a first-order method, so the numerical trajectory only approximates the continuous LV trajectory [15, Chapter II].

Additionally, convergence speed depends on both the graph structure and the parameter τ. In some cases, the system converges rapidly, while in others it may evolve slowly near unstable or nearly degenerate configurations. As a result, some vertices may still have intermediate values at time $t_{\max}$, even though they would eventually converge to 0 or 1 for larger times.

These observations highlight an important distinction between theory and practice. While the theoretical result guarantees convergence to a maximal independent set as $t \to \infty$, in practice we only approximate this limit. Therefore, the final solution depends not only on the graph and the parameter τ, but also on numerical choices such as Δt, $t_{\max}$, and the threshold θ.

Despite these limitations, the chosen threshold $\theta = 0.001$ works effectively in practice, as the dynamics tend to clearly separate vertices into those that are strongly suppressed and those that remain active.

3.3.10. Preventing Bias from the Threshold θ

Since the threshold value $\theta = 0.001$ chosen in Section 3.3.9 is quite low, two adjacent vertices may, in rare cases, both still have values slightly above θ, even though they should not both be included in an independent set. To avoid this problem, we add a final filtering step after the Lotka–Volterra dynamics have finished. First, we collect all vertices with

$$x_i > \theta.$$

Then we order these vertices from largest to smallest value of x_i. We start with an empty set S, which will become the final independent set. Going through the ordered list one vertex at a time, we add a vertex to S only if it is not adjacent to any vertex already in S.

In this way, vertices with larger final Lotka–Volterra values are given priority, while the independence condition is still enforced. This post-processing step ensures that the final output is always an independent set, even when the threshold is low or when the dynamics have not fully converged.

3.4. Simulated Annealing

Simulated annealing also uses the idea of an energy landscape, but in a different way from the Lotka–Volterra method. Instead of a continuous Lyapunov function, simulated annealing uses an energy function defined on binary vectors. Each binary variable indicates whether a vertex is selected, so the maximum-independent-set problem is rewritten as an optimization problem over binary variables.

3.4.1. Energy Formulation Using a QUBO

For each vertex $i \in V$, we assign a binary variable

$$x_i \in \{0, 1\}.$$

The interpretation is

$$x_i = \begin{cases} 1, & \text{if vertex } i \text{ is selected,} \\ 0, & \text{if vertex } i \text{ is not selected.} \end{cases}$$

Unlike in the Lotka–Volterra method, these variables are not continuous. They are either 0 or 1. The goal is to choose as many vertices as possible, while avoiding the selection of two adjacent vertices.

This can be written as a QUBO problem, which stands for Quadratic Unconstrained Binary Optimization. A QUBO problem is an optimization problem where the variables are binary and the objective function contains linear and quadratic terms. The word “unconstrained” means that the independence condition is not imposed as a separate constraint. Instead, violations of the constraint are included as penalty terms in the objective function. This energy function is called the Hamiltonian.

For the maximum-independent-set problem, we use the Hamiltonian

$$H(x) = -\sum_{i \in V} x_i + \alpha \sum_{\{i,j\} \in E} x_i x_j, \tag{3.5}$$

where $\alpha > 0$ is a penalty parameter.

The first term,

$$-\sum_{i \in V} x_i,$$

rewards selecting many vertices. Since we minimize $H(x)$, selecting an extra vertex lowers the energy by 1. The second term,

$$\alpha \sum_{\{i,j\} \in E} x_i x_j,$$

adds a penalty whenever two adjacent vertices are both selected. Indeed, if $x_i = x_j = 1$ and $\{i, j\} \in E$, then the product $x_i x_j$ equals 1, so the energy increases by α.

Thus, minimizing $H(x)$ means balancing two goals: selecting many vertices, while avoiding edges between selected vertices. If α is chosen large enough, configurations that select adjacent vertices become energetically unfavorable. In this way, low-energy configurations correspond to large independent sets. This QUBO formulation of the maximum-independent-set problem is commonly used in energy-based optimization methods [8, 22].

The energy landscape is the collection of all possible binary configurations $x \in \{0, 1\}^n$, together with their energy values $H(x)$. Each configuration represents a possible choice of selected vertices. Simulated annealing moves through this landscape by making random changes to the current configuration and deciding whether to accept or reject them. The aim is to find a configuration with very low energy, which corresponds to a large independent set.

3.4.2. Simulated Annealing Algorithm

While the QUBO formulation gives an energy function for the maximum-independent-set problem, it does not yet tell us how to find a low-energy configuration.

A simple way to search this landscape would be to only accept moves that decrease the energy. For example, we could flip one variable x_i, check whether the energy improves, and keep the change only if the new configuration has lower energy.

However, this can cause a problem. The algorithm may reach a point where all nearby moves, meaning all configurations obtained by flipping one variable x_i, make the energy worse, even though a much better solution exists somewhere else in the landscape. Such a point is called a local minimum. This means that the algorithm has found a configuration that cannot easily be improved by a small local change, but that is still not the best possible solution.

Simulated annealing is designed to reduce this problem. Instead of only accepting moves that lower the energy, it sometimes also accepts moves that increase the energy. At the beginning of the algorithm, these uphill moves are accepted more often, so the algorithm can explore the landscape and escape local minima. Later, uphill moves become less likely, and the algorithm gradually focuses on low-energy configurations [19, 32].

In the energy-landscape picture, the difficulty of escaping a local minimum depends not only on the height of an energy barrier, but also on its width. The height describes how much the energy must temporarily increase, while the width describes how many unfavorable moves may be needed before the algorithm reaches a better region. Here, a better region means a part of the search space with access to lower-energy configurations, which usually correspond to larger independent sets or configurations with fewer edge conflicts. This distinction is useful later when studying why simulated annealing can fail on some graph families.

This idea is illustrated in Figure 3.2. A descent method can get stuck in the first local minimum it reaches, because it only moves downhill. Simulated annealing can temporarily move uphill, which may allow it to leave a local minimum and continue searching for a better solution.

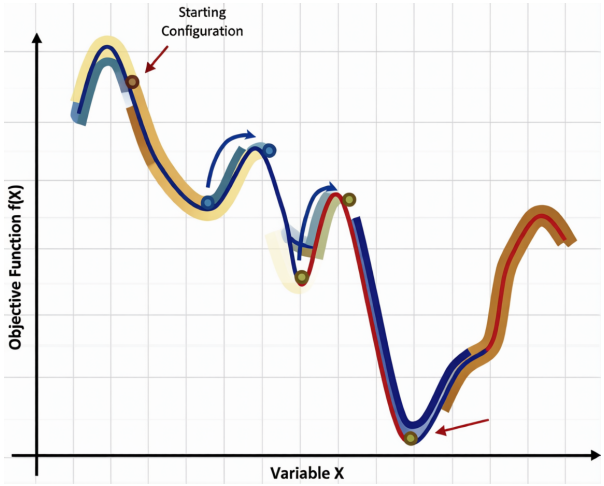


Figure 3.2: Created using AI image generator, adapted from Nini (2019) [28]. Illustration of simulated annealing on an energy landscape. A descent method only accepts moves that decrease the energy, so it can become stuck in a local minimum. Simulated annealing can sometimes accept moves to higher-energy states, especially at high temperature, which allows it to escape local minima and continue searching for lower-energy configurations.

3.4.3. Metropolis Acceptance Rule

Simulated annealing uses the Hamiltonian $H(x)$ as the energy of a binary state x. Here, $x \in \{0,1\}^n$, where $x_i = 1$ means that vertex i is selected and $x_i = 0$ means that it is not selected. The Hamiltonian used in this thesis is defined in Equation 3.5. Low-energy states correspond to configurations with many selected vertices and few, or no, edge conflicts.

At each iteration, the algorithm generates a new candidate state x' from the current state x. In our implementation, this is done by flipping a single bit, meaning that one randomly chosen binary variable x_i is changed from 0 to 1, or from 1 to 0. The change in energy is then

$$\Delta H = H(x') - H(x).$$

The accept-or-reject step is based on the Metropolis rule [24]. If the candidate state has lower or equal energy, so $\Delta H \leq 0$, it is always accepted. If the candidate state has higher energy, so $\Delta H > 0$, it is accepted only with a certain probability.

This probability is motivated by the Boltzmann distribution from statistical mechanics. At temperature T, lower-energy states are more likely than higher-energy states, with probabilities proportional

to

$$e^{-E/(kT)},$$

where k is the Boltzmann constant [19]. In simulated annealing, the constant k is absorbed into the temperature parameter, so T is treated as a computational temperature. The Metropolis acceptance probability is then

$$P(\text{accept}) = \begin{cases} 1, & \text{if } \Delta H \leq 0, \\ \exp\left(-\frac{\Delta H}{T}\right), & \text{if } \Delta H > 0. \end{cases} \tag{3.6}$$

Thus, moves that improve the solution are always accepted, while worse moves can still be accepted with a probability depending on the energy increase and the current temperature. Small energy increases are more likely to be accepted than large ones. Also, when the temperature is high, worse moves are accepted more often; when the temperature is low, they are rarely accepted. This allows simulated annealing to explore the search space early on and gradually become more selective as the temperature decreases [19, 24].

3.4.4. Cooling Schedule

It remains to specify how the temperature T changes during the algorithm. In simulated annealing, T is gradually decreased according to a cooling schedule. In this thesis, we use an exponential cooling schedule. The motivation for this choice, as well as a discussion of alternative cooling mechanisms, is given in Section 3.4.5.

Let N denote the total number of simulated annealing iterations. We write T_k for the temperature at iteration k, where $k = 0, 1, \ldots, N-1$. The initial temperature is denoted by T_{start}, and the final temperature after the last iteration is denoted by T_{end}. For an exponential cooling schedule, the temperature is multiplied by a constant decay factor ρ at each iteration, so

$$T_k = T_{\text{start}} \rho^k.$$

This already gives $T_0 = T_{\text{start}}$. We choose the decay factor ρ so that the final iteration reaches the prescribed final temperature T_{end}, that is,

$$T_{N-1} = T_{\text{end}}.$$

Substituting $k = N - 1$ into the exponential schedule gives

$$T_{\text{end}} = T_{\text{start}} \rho^{N-1}.$$

Solving for ρ gives

$$\rho = \left(\frac{T_{\text{end}}}{T_{\text{start}}}\right)^{1/(N-1)}.$$

Therefore, the temperature at iteration k is

$$T_k = T_{\text{start}} \left(\frac{T_{\text{end}}}{T_{\text{start}}}\right)^{k/(N-1)}. \tag{3.7}$$

3.4.5. Other Cooling Mechanisms

The cooling schedule determines how the temperature T decreases during simulated annealing. This is important because the temperature controls how likely the algorithm is to accept moves that increase the energy. At high temperature, the algorithm explores the energy landscape more freely. At low temperature, it becomes more selective and mostly accepts moves that lower the energy.

One theoretically important cooling schedule is the logarithmic schedule,

$$T_k = \frac{T_0}{\log(1 + k)},$$

where k denotes the iteration number. The advantage of this schedule is that, under suitable conditions, logarithmic cooling gives convergence guarantees to a global minimum [14, 16]. However, this

schedule cools extremely slowly. In practice, this means that many iterations are needed before the temperature becomes small, making the method computationally expensive.

For simulations, a faster schedule is often preferred. A common choice is geometric cooling, also called exponential cooling, where

$$T_k = T_0 \rho^k,$$

with $0 < \rho < 1$, as defined in Equation 3.7. This is the schedule used in this thesis. It is much faster than logarithmic cooling and is therefore more suitable for numerical experiments, although it does not give the same general convergence guarantee to the global minimum [23, 33].

Another simple option is linear cooling, where the temperature decreases by the same amount at every step:

$$T_k = T_{\text{start}} - k\left(\frac{T_{\text{start}} - T_{\text{end}}}{N-1}\right).$$

Linear cooling is easy to implement, but it is less flexible. Depending on the problem, it may cool too quickly in some parts of the search and too slowly in others. For this reason, geometric cooling is often a more convenient practical choice.

3.4.6. Temperature Parameters Used in This Thesis

The choice of the starting and ending temperatures depends on the scale of the energy function. Simulated annealing does not depend on the temperature T alone, but on the ratio

$$\frac{\Delta H}{T},$$

where ΔH is the change in energy caused by a proposed move. This means that the temperature must be chosen relative to the typical size of the energy changes in the problem.

In the original work of Kirkpatrick et al. [19], large temperature values are used because the objective function values are on a large scale. For example, if

$$\Delta H \approx 1000 \quad \text{and} \quad T = 10{,}000,$$

then the probability of accepting a worse move is

$$P(\text{accept}) = e^{-1000/10000} = e^{-0.1} \approx 0.90.$$

So, even though the temperature looks very large, it is large because the energy differences are also large. In our case, the energy function is the QUBO Hamiltonian

$$H(x) = -\sum_{i \in V} x_i + \alpha \sum_{\{i,j\} \in E} x_i x_j,$$

as introduced in Equation 3.5. When flipping one variable x_i, the energy change is roughly of the order

$$\Delta H \approx 1 + \alpha d_i,$$

where d_i is the degree of vertex i. The term 1 comes from either adding or removing one selected vertex, while the term αd_i comes from the possible penalty terms involving the neighbors of i. In the smaller graphs considered in this BEP, the degrees are usually not very large. With our choice $\alpha = 1.1$, this means that typical energy changes are often on the scale of about 1 to 10, rather than thousands.

Because the acceptance probability depends on the ratio $\Delta H/T$, our temperature values should be chosen on this smaller scale. At the start of the algorithm, we want the temperature to be high enough that worse moves are still accepted reasonably often. For example, if

$$\Delta H \approx 1 \quad \text{and} \quad T_{\text{start}} = 2,$$

then

$$P(\text{accept}) = \exp\left(-\frac{1}{2}\right) \approx 0.61.$$

So a move that increases the energy by 1 is still accepted with probability about 61%. This allows the algorithm to explore the energy landscape instead of immediately behaving like a greedy method.

At the end of the algorithm, we want the temperature to be very low, so that moves increasing the energy are almost never accepted. With

$$T_{\text{end}} = 0.001,$$

a move with

$$\Delta H \approx 1$$

is accepted with probability

$$P(\text{accept}) = \exp\left(-\frac{1}{0.001}\right) = e^{-1000} \approx 0.$$

Thus, near the end of the cooling process, simulated annealing behaves almost like a local descent method: it mainly accepts moves that lower the energy and almost never accepts moves that increase it.

We do not set the final temperature equal to exactly zero, because the acceptance probability contains division by T. Instead, we use a very small positive value. In this thesis, the simulated annealing schedule therefore starts at

$$T_{\text{start}} = 2$$

and ends at

$$T_{\text{end}} = 0.001.$$

These values are chosen with respect to the typical energy change ΔH caused by a single-bit flip, since this is the quantity that appears in the Metropolis acceptance probability. Flipping one bit changes the number of selected vertices by one and may also change the number of edge conflicts involving that vertex. Thus, the resulting energy changes are usually of order one for the parameter values used in the simulations.

3.4.7. Why $\alpha > 1$

The parameter α controls how strongly we penalize the selection of adjacent vertices. In the Hamiltonian in Equation 3.5, the first term rewards selecting vertices, while the second term penalizes selecting two neighboring vertices at the same time.

If $\alpha = 0$, then the Hamiltonian becomes

$$H(x) = -\sum_{i \in V} x_i.$$

In that case, the lowest energy is obtained by setting $x_i = 1$ for every vertex i. This would select all vertices, even if many of them are adjacent. Therefore, $\alpha = 0$ does not encode the independent set constraint.

The value of α must be large enough so that violating the independence condition is more expensive than the reward gained by selecting an extra vertex. To see this, suppose two adjacent vertices i and j are both selected, so

$$x_i = x_j = 1 \quad \text{and} \quad \{i, j\} \in E.$$

Then the edge $\{i, j\}$ contributes a penalty of α to the Hamiltonian. If we remove one of these vertices from the set, the reward term becomes worse by 1, because we selected one fewer vertex. However, we also remove at least one penalty term of size α. So the change in energy is at most

$$\Delta H = 1 - \alpha.$$

If $\alpha > 1$, then $\Delta H < 0$. This means that removing one endpoint of a conflicting edge lowers the energy. Therefore, a lowest-energy configuration cannot contain two adjacent selected vertices.

This is why we choose $\alpha > 1$. It guarantees that configurations violating the independence condition are energetically unfavorable. Among configurations that are independent sets, the penalty term is zero, so the Hamiltonian reduces to

$$H(x) = -\sum_{i \in V} x_i.$$

Minimizing $H(x)$ is then the same as maximizing the number of selected vertices. Thus, for $\alpha > 1$, the ground states of the Hamiltonian correspond to maximum independent sets [8, 22]. A more detailed proof is given in Section 3.4.8.

In the simulations in this thesis, we use

$$\alpha = 1.1.$$

This choice satisfies $\alpha > 1$, so selecting adjacent vertices is penalized more strongly than selecting an additional vertex is rewarded.

3.4.8. How $H(x)$ Connects to Independent Sets

We now make precise how the Hamiltonian is connected to independent sets. This argument is included to show why the QUBO formulation used in this thesis represents the maximum-independent-set problem. In this section, we assume $\alpha > 1$ and prove two claims:

$$\text{ground states of } H \quad \Leftrightarrow \quad \text{maximum independent sets,}$$

and

$$\text{local minima of } H \quad \Leftrightarrow \quad \text{maximal independent sets.}$$

Here, a *ground state* means a global minimizer of H over the binary domain $\{0,1\}^n$. A *local minimum* means a configuration whose energy cannot be decreased by flipping a single bit.

Proof. We first show that the ground states of H correspond exactly to maximum independent sets.

First direction: ground state implies maximum independent set. Let $x \in \{0,1\}^n$ be a ground state of H, and let

$$S = \{i \in V : x_i = 1\}$$

be the set of selected vertices.

We first show that S must be an independent set. Suppose, for contradiction, that S is not independent. Then there exist two adjacent vertices $i, j \in S$, so

$$\{i,j\} \in E \quad \text{and} \quad x_i = x_j = 1.$$

Choose one selected vertex i, and define m as the number of selected neighbors of i:

$$m = \left|\{j \in S : \{i,j\} \in E\}\right|.$$

In other words, the selected set contains a conflict, meaning an edge whose two endpoints are both selected.

Now consider the new configuration obtained by removing i, that is, changing x_i from 1 to 0. Removing i increases the first term of H by 1, because one fewer vertex is selected. However, it removes m penalty terms, each of size α. Therefore the change in energy is

$$\Delta H = 1 - \alpha m.$$

Since $\alpha > 1$ and $m \geq 1$, we have

$$\Delta H \leq 1 - \alpha < 0.$$

So removing i lowers the energy. This contradicts the assumption that x is a ground state. Therefore, S must be an independent set.

For an independent set,

$$x_i x_j = 0 \quad \text{for all } \{i,j\} \in E.$$

Therefore

$$H(x) = -\sum_{i \in V} x_i = -|S|.$$

Thus, among independent sets, minimizing $H(x)$ is exactly the same as maximizing $|S|$. Since x is a ground state, S must therefore be a maximum independent set.

Second direction: maximum independent set implies ground state. Now let $S \subseteq V$ be a maximum independent set, and let $x = \chi_S$ be its characteristic vector. Since S is independent, the penalty term is zero, so

$$H(\chi_S) = -|S|.$$

Suppose, for contradiction, that χ_S is not a ground state. Then there exists some configuration $y \in \{0,1\}^n$ such that

$$H(y) < H(\chi_S).$$

Let

$$T = \{i \in V : y_i = 1\}.$$

If T is not independent, then by the same argument as above, we can remove a vertex involved in a selected edge and strictly lower the energy. Repeating this process eventually gives an independent set T' with

$$H(\chi_{T'}) < H(\chi_S).$$

But for independent sets, the Hamiltonian equals minus the size of the set, so

$$H(\chi_{T'}) = -|T'| \quad \text{and} \quad H(\chi_S) = -|S|.$$

Thus

$$-|T'| < -|S|,$$

which implies

$$|T'| > |S|.$$

This contradicts the assumption that S is a maximum independent set. Therefore, no configuration has lower energy than χ_S, and χ_S is a ground state.

Hence,

$$\text{ground states of } H \quad \Leftrightarrow \quad \text{maximum independent sets.}$$

□

We now prove that the local minima of H, with respect to single-bit flips, correspond exactly to maximal independent sets.

First direction: local minimum implies maximal independent set.

Proof. Let $x \in \{0,1\}^n$ be a local minimum of H, and let

$$S = \{i \in V : x_i = 1\}.$$

First, we show that S is independent. Suppose not. Then there exists a vertex $i \in S$ with at least one neighbor in S. Let

$$m = \left|\{j \in S : \{i,j\} \in E\}\right|.$$

Then $m \geq 1$. If we flip x_i from 1 to 0, the energy changes by

$$\Delta H = 1 - \alpha m.$$

Since $\alpha > 1$ and $m \geq 1$, we have $\Delta H < 0$. So a single-bit flip lowers the energy. This contradicts the assumption that x is a local minimum. Therefore, S must be independent.

Next, we show that S is maximal. Suppose, for contradiction, that S is not maximal. Then there exists a vertex $i \notin S$ with no neighbors in S. This means that adding i to S still gives an independent set.

In terms of the binary vector, this corresponds to flipping x_i from 0 to 1. Since i has no selected neighbors, no penalty term is added. The only change in energy is that the first term decreases by 1. Hence

$$\Delta H = -1 < 0.$$

So adding i lowers the energy, contradicting the assumption that x is a local minimum. Therefore, S must be maximal. Thus every local minimum of H corresponds to a maximal independent set.

Second direction: maximal independent set implies local minimum. Now let $S \subseteq V$ be a maximal independent set, and let $x = \chi_S$. We show that x is a local minimum of H with respect to single-bit flips. There are two possible types of single-bit flips.

First, suppose we remove a selected vertex $i \in S$, so x_i changes from 1 to 0. Since S is independent, there are no penalty terms involving selected edges. Removing i simply selects one fewer vertex, so the energy changes by

$$\Delta H = 1 > 0.$$

Thus removing a selected vertex increases the energy.

Second, suppose we add an unselected vertex $v \notin S$, so x_v changes from 0 to 1. Since S is maximal, every vertex outside S has at least one neighbor in S. Let

$$m = \left|\{j \in S : \{v, j\} \in E\}\right|.$$

Then $m \geq 1$. Adding v lowers the first term by 1, but it also creates m penalty terms, each of size α. Therefore

$$\Delta H = -1 + \alpha m.$$

Since $\alpha > 1$ and $m \geq 1$,

$$\Delta H \geq -1 + \alpha > 0.$$

Thus adding any unselected vertex increases the energy.

Since every possible single-bit flip increases the energy, χ_S is a local minimum of H.

Hence,

$$\text{local minima of } H \quad \Leftrightarrow \quad \text{maximal independent sets.}$$

□

The important conclusion is that the energy landscape has two levels of meaning. The global minima, or ground states, correspond to maximum independent sets. The local minima correspond to maximal independent sets, which may be smaller than the maximum. This distinction is important for simulated annealing, because the algorithm can become trapped in local minima if the cooling process is too fast or the energy landscape contains many suboptimal valleys.

3.4.9. Final Simulated Annealing Procedure Used in This Thesis

The simulated annealing procedure used in this thesis is summarized as follows:

1. Initialize a random state $x^{(0)} \in \{0, 1\}^n$.
2. Compute the initial energy $H(x^{(0)})$.
3. For $k = 0, \ldots, N - 1$:
   (a) Select a random vertex i and flip its state:

$$x_i = \begin{cases} 0 \to 1, \\ 1 \to 0. \end{cases}$$

   (b) Compute the new energy and the energy change ΔH.
   (c) Accept the move with probability given by the Metropolis rule in Equation 3.6.
   (d) Update the temperature using Equation 3.7.

Similarly to the Lotka–Volterra method, the output of simulated annealing may not always be an independent set in practice. In theory, low-energy configurations should avoid selecting neighboring vertices when $\alpha > 1$. However, the algorithm only runs for a finite number of steps and ends at a small but nonzero temperature, in our case $T = 0.001$. Therefore, it is still possible that the final configuration contains two adjacent vertices with

$$x_i = x_j = 1.$$

To make sure that the final output is always an independent set, we add a final filtering step, similar to the one used for the Lotka–Volterra algorithm in Section 3.3.10. Since the variables in simulated

annealing are binary, all selected vertices have value $x_i = 1$. Therefore, we cannot rank them by size as in the LV case. Instead, we order the selected vertices by their index.

We then create an empty set S, which will become the final independent set. Going through the selected vertices one by one, we add a vertex to S only if it is not adjacent to any vertex already in S. This removes possible conflicts while keeping as many selected vertices as possible according to the chosen ordering.

This final check does not change the simulated annealing process itself. It only ensures that the returned solution is a valid independent set, even if the final binary configuration still contains neighboring selected vertices. In the experiments performed for this thesis, this correction step was rarely needed: in the tested runs, the final simulated annealing configurations were already valid independent sets.

3.4.10. Running Time

The input is a graph with n vertices and m edges. At each iteration, the algorithm first selects one vertex uniformly at random and flips the corresponding bit. This random selection and bit flip take $O(1)$ time. The algorithm then computes the energy of the candidate solution. If the Hamiltonian in Equation 3.5 is recomputed from scratch, this takes $O(n+m)$ time, because the Hamiltonian contains one sum over all vertices and one sum over all edges.

The remaining operations in one iteration, such as computing the energy difference, evaluating the acceptance probability, and deciding whether to accept the move, take constant time. Therefore, one iteration takes $O(n+m)$ time. If the algorithm runs for N iterations, the total running time is

$$O(N(n+m)).$$

In the worst case of dense graphs, where $m = O(n^2)$, this becomes

$$O(Nn^2).$$

Thus, the computational cost of simulated annealing depends both on the number of iterations and on the size of the graph [32].

3.5. Experimental Methodology

This section summarizes how the numerical experiments are organized.

The experiments are designed around two goals. First, we compare the solution sizes found by the minimum-degree greedy algorithm, simulated annealing, and Lotka–Volterra dynamics on different graph families. Second, we use more targeted experiments to understand why an algorithm performs poorly in certain cases. Thus, the experiments are not only intended to rank the algorithms, but also to identify the structural or parameter-dependent mechanisms behind their failures.

The experimental work started by reproducing simple test graphs and running the algorithms on them. This was used to check that the implementations behaved as expected before moving to larger random and structured graph families. After this initial testing, the experiments were organized by graph family and by the type of information known about the graph. In particular, the files were separated into three main categories: cases where both maximal and maximum independent sets were known, cases where only the maximum independent set was known, and cases where neither was known exactly.

For small graphs, usually with $n = 12, \ldots, 20$, exhaustive enumeration is used as a benchmark. This makes it possible to compare each heuristic directly with the true maximum independent set size. For larger graphs, the exact optimum is usually not computed, because exhaustive search becomes impractical. In those experiments, we compare the algorithms using the largest solution size found among the tested methods and parameter settings. This best-known value is only an empirical reference value, not a proof of optimality.

The graph families are chosen to test different structural effects. Erdős–Rényi graphs are used as a standard random baseline. Random geometric graphs are used to study the effect of clustering and local spatial structure. Random regular graphs are used to remove degree variation as much as possible. Other graph families are constructed to isolate specific failure mechanisms.

The main performance measure is the size of the independent set returned by each algorithm. In classification plots, we also record whether the output is maximum, maximal but not maximum, or non-maximal. For simulated annealing, we additionally study the effect of the cooling length and the

penalty parameter α. For Lotka–Volterra dynamics, we study the effect of the competition parameter τ, the selected final independent set, and the convergence time of the thresholded solution.

Unless stated otherwise, each experiment is run 50 times for each value of n, and the reported value is the average over these runs or graph instances. The same graph instances are used across algorithms whenever possible, so that differences in performance are caused by the algorithms rather than by different random graphs. For randomized methods, repeated runs are used to reduce the effect of a single random initialization.

All experiments were implemented in Python. The code was organized so that the graph constructions, plotting functions, and experiment scripts were kept separate. One Python file contained the graph constructions used throughout the project, and a separate plotting file was used to visualize and inspect the graph structures. The numerical experiments themselves were run using separate scripts for each graph family or experiment type.

The output of each experiment was saved in a structured folder system. Each graph family or experiment had its own main folder. Inside this folder, the results were separated by graph size. Thus, for each value of n, a separate subfolder stored the data from the individual runs for that graph size. These files contained the algorithm outputs, such as the independent set sizes, classifications, parameter values, and, where relevant, convergence or trajectory information.

In addition to the folders for individual graph sizes, each experiment also had a summary folder. This folder contained the aggregated data across all tested values of n, together with the plots used in the thesis. For example, the summary files stored averages over the runs, classification percentages, and data needed to reproduce the final figures. This structure made it possible to inspect individual runs, compare results across graph sizes, and regenerate the plots in a consistent way.

NetworkX was used for graph generation and graph operations, NumPy for numerical computations, and Matplotlib for plotting. Random seeds were used to make the experiments reproducible.

4

Evaluation of the Minimum-Degree Greedy Algorithm

This chapter evaluates the minimum-degree greedy algorithm introduced in Chapter 3. The goal is not only to show examples where Greedy performs poorly, but to understand which structural features make the minimum-degree rule reliable or unreliable.

The minimum-degree rule is locally motivated: selecting a vertex of small degree removes relatively few neighboring vertices from the graph. This can be a good strategy on many graph families, and for this reason Greedy is often a useful baseline. However, the same local rule can also be misleading. A vertex may have small degree, but selecting it can still block access to a much larger independent set.

We therefore study two diagnostic graph families. The first is a known construction from Halldórsson and Radhakrishnan [17], which isolates a structural weakness of the minimum-degree rule. The second is a simpler graph family constructed in this thesis, where the same mechanism can be seen directly from the vertex degrees. This second family is also useful because it shows a transition: Greedy performs poorly for some parameter values, but recovers the maximum independent set for others.

4.1. Structural Stress Tests for Greedy

Greedy is a simple and well-known heuristic for the maximum-independent-set problem. In many graph families, it performs surprisingly well, which makes it difficult to find natural examples where it clearly fails. However, greedy algorithms are sensitive to the choices made during the selection process, especially when several vertices have the same degree. In such cases, the tie-breaking rule or vertex ordering can influence the final independent set [1, 6].

In this thesis, we are mainly interested in structural failures of the minimum-degree rule, rather than failures caused only by unlucky tie-breaking. For this reason, we first test a construction from Halldórsson and Radhakrishnan [17]. In this thesis, we refer to this construction as the *greedy-bad-chain* graph; this name is introduced here for convenience and is not the terminology used in the original article. The graph should be interpreted as a diagnostic stress test: it is useful because it makes the failure mechanism of the minimum-degree rule explicit.

We test greedy-bad-chain graphs with block size $l = 5$ and with the number of chain blocks varying from 1 to 6. The instance with four chain blocks is shown in Figure 4.1.

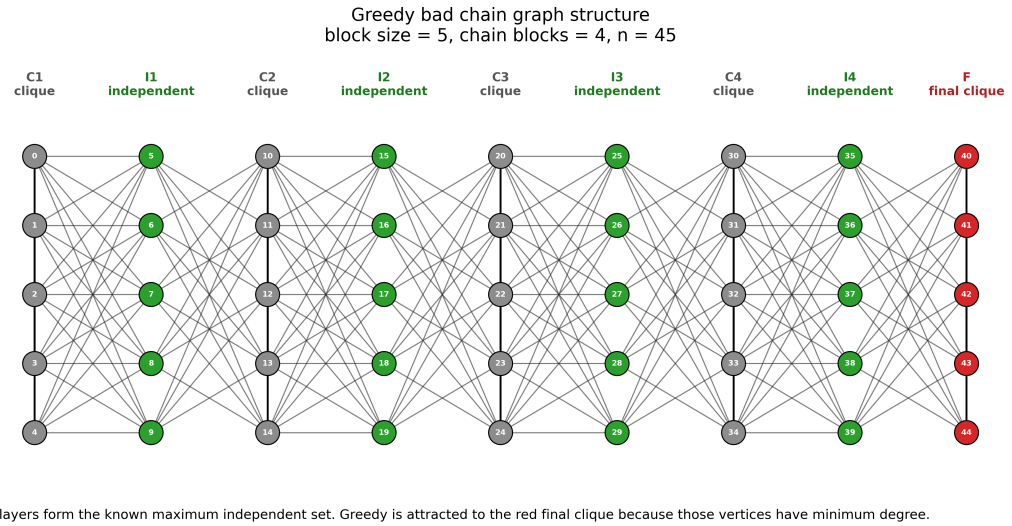


Figure 4.1: Structure of the greedy-bad-chain graph with block size $l = 5$ and four chain blocks. Each gray layer C_i is a clique, each green layer I_i is an independent set, and the red layer F is the final clique. The maximum independent set consists of all green independent layers. Greedy fails because the minimum-degree rule first selects a vertex from the final clique, which blocks almost all vertices in the last independent layer and forces the algorithm into a smaller maximal independent set.

Each chain block consists of a clique C_i of size l and an independent set I_i of size l. Within each block, every vertex in C_i is connected to every vertex in I_i. Between consecutive blocks, I_i is connected to C_{i+1} almost completely: all possible edges are present except for one matching. Here, a matching means a set of edges in which no two edges share a vertex. Thus, one matching pairs the vertices of I_i with the vertices of C_{i+1}, and these paired edges are the only edges omitted. At the end, a final clique F of size l is added and connected to the last independent layer in the same way.

The maximum independent set is$I_1 \cup I_2 \cup \cdots \cup I_b$, so its size is lb, where b is the number of chain blocks.

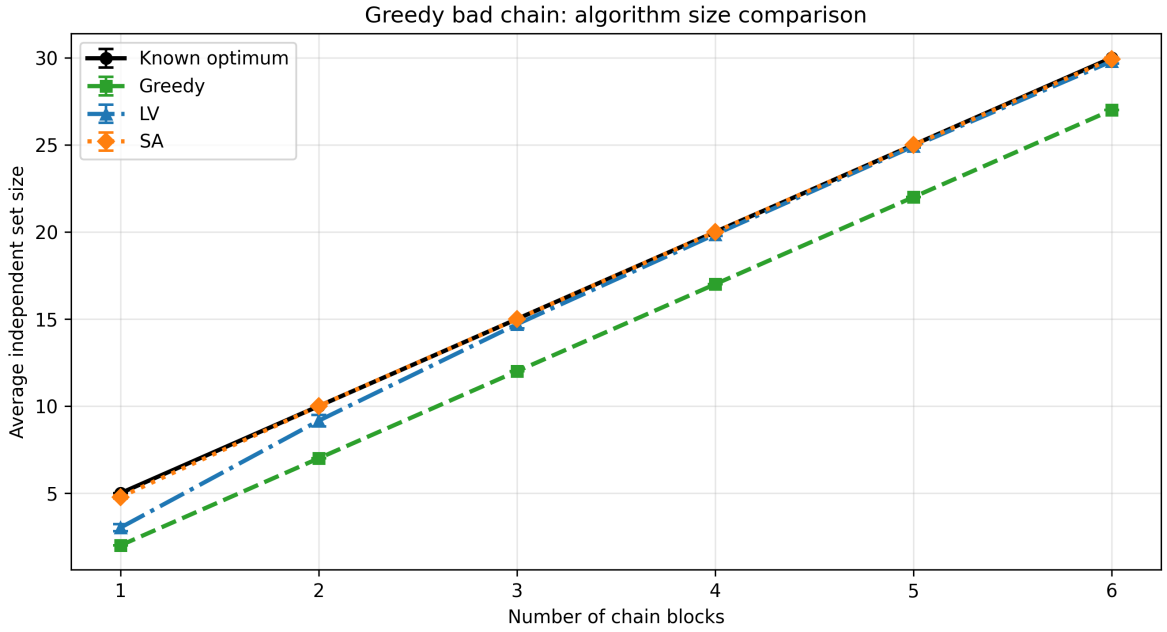


Figure 4.2: Average independent set size found by Greedy, Lotka–Volterra, and simulated annealing on greedy-bad-chain graphs with block size $l = 5$. The optimum grows linearly with the number of chain blocks. Greedy consistently returns a smaller independent set, while Lotka–Volterra and simulated annealing remain close to the optimum. Some curves overlap in the plot, especially for larger numbers of chain blocks. In particular, the Lotka–Volterra and simulated annealing curves are very close to the optimum. The LV and known optimum lines are hidden behind the orange simulated annealing line.

Figure 4.2 shows that Greedy performs much worse than the other algorithms on this graph family. The reason is not simply that the graph was chosen to be difficult, but that the construction exposes a specific weakness of the minimum-degree rule. For $l = 5$, vertices in the final clique have degree $2l - 2 = 8$, while vertices in the first clique and independent layer have degree $2l - 1 = 9$, and internal clique vertices have degree $3l - 2 = 13$. Therefore, the minimum-degree rule selects a vertex from the final clique first.

This is the wrong structural choice. Selecting a final clique vertex blocks almost all vertices in the last independent layer, leaving only the unmatched vertex available. The algorithm is then forced backward through the chain and loses many vertices that would have belonged to the maximum independent set.

Thus, the failure is not mainly caused by random tie-breaking. In this construction, the minimum-degree rule itself points Greedy toward the trap. It chooses the locally best-looking vertex, but this choice prevents the algorithm from reaching the much larger independent set.

To examine the same mechanism in a simpler and more transparent setting, we also construct a graph family in this thesis. We call this family the trap-clique-independent graph. Unlike the greedy-bad-chain construction, this family has only three parts, and the transition between failure and success can be read directly from the vertex degrees.

We test this graph family with three parts:

$$t = \text{one trap vertex}, \qquad I = \text{an independent set of size } |I|, \qquad C = \text{a clique of size } |C|.$$

In the experiment, we fix $|C| = 10$ and vary $|I| = 2, 3, \ldots, 20$.

The graph is constructed as follows. The trap vertex t is connected to every vertex in I. Every vertex in I is connected to every vertex in C. The set C forms a clique, and the trap vertex t is not connected to any vertex in C. We provide an example of the graph structure in Figure 4.3.

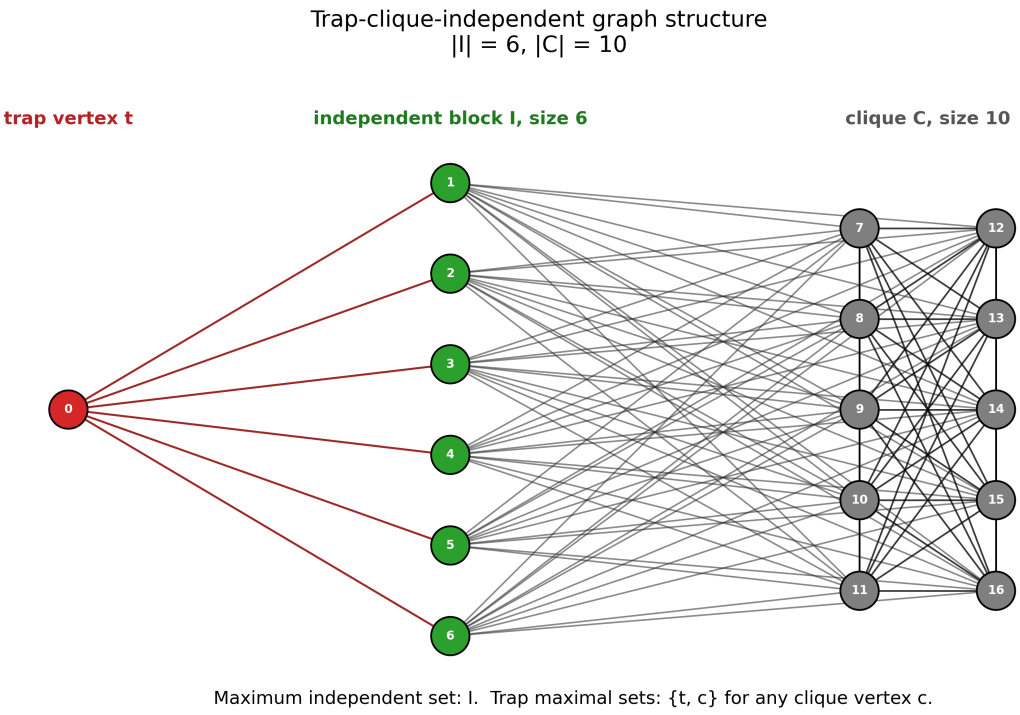


Figure 4.3: Structure of the trap-clique-independent graph with $|I| = 6$ and $|C| = 10$. The red vertex t is the trap vertex, the green vertices form the independent set I, and the gray vertices form the clique C. The trap vertex is connected to all vertices in I, every vertex in I is connected to every vertex in C, and C is fully connected internally. The name *trap-clique-independent graph* is used because the graph contains a trap vertex, a clique, and an independent set.

The maximum independent set is the full independent set I, so its size is $|I|$. The trap maximal independent sets are of the form $\{t, c\}$, where c is any clique vertex. These have size 2.

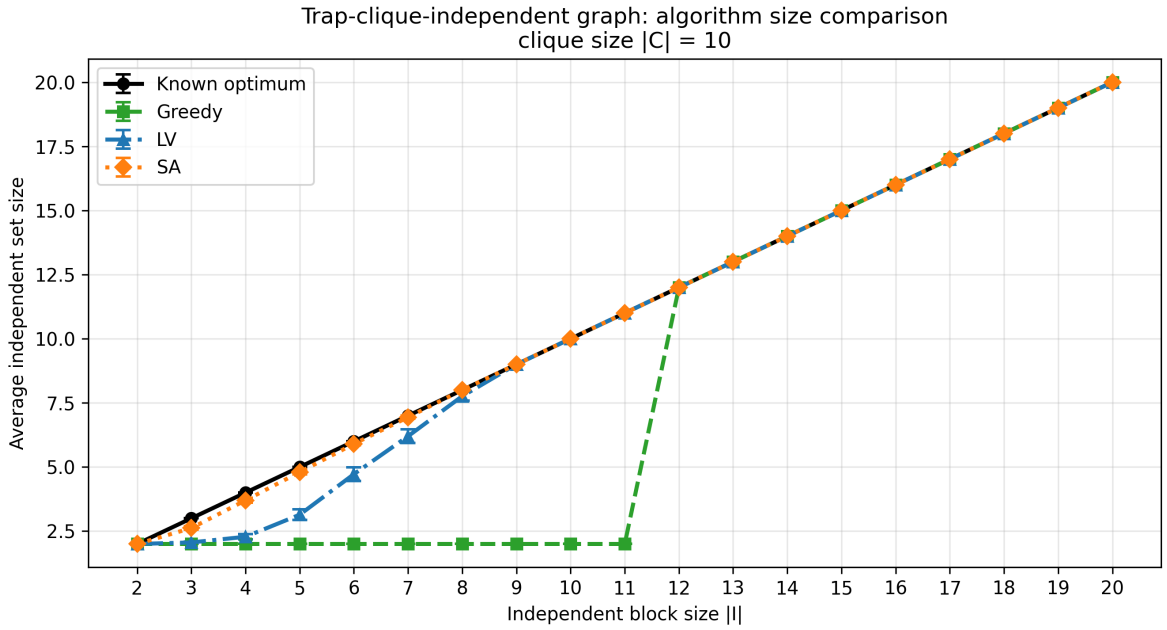


Figure 4.4: Average independent set size found by Greedy, Lotka–Volterra, and simulated annealing on the trap-clique-independent graph with fixed clique size $|C| = 10$. The optimum is the full independent set I. Greedy performs poorly for small values of $|I|$, but improves once $|I|$ becomes sufficiently large. Several curves overlap for larger $|I|$: from $|I| = 9$, Lotka–Volterra and simulated annealing coincide with the optimum, and from $|I| = 12$, Greedy also coincides with the optimum. Thus, some lines are hidden behind the optimum or algorithm curves in this region.

Figure 4.4 shows that Greedy performs poorly when $|I|$ is small, but that this behavior changes as $|I|$ grows. In the small-$|I|$ regime, Greedy often returns a trap maximal independent set of size 2, while the true optimum has size $|I|$. For larger $|I|$, Greedy improves and eventually finds the maximum independent set. Figure D.1 in the appendix gives an additional classification of the outcomes, showing whether each algorithm finds the maximum independent set, the trap maximal set, or another solution.

To understand this transition, we examine the vertex degrees. Greedy selects a vertex of minimum degree at each step. In this graph, the degrees are

$$\deg(t) = |I|, \qquad \deg(i) = |C| + 1, \qquad \deg(c) = |I| + |C| - 1.$$

Since $|C| = 10$, this becomes

$$\deg(t) = |I|, \qquad \deg(i) = 11, \qquad \deg(c) = |I| + 9.$$

Therefore, when $|I| < 11$, the trap vertex t has the smallest degree, so Greedy selects it first. This is the wrong choice, because t is adjacent to every vertex in the true maximum independent set I. Once t is chosen, all vertices in I are removed from consideration, and the algorithm can only complete the solution by adding one clique vertex. It then returns a trap maximal independent set of the form $\{t, c\}$, of size 2.

At $|I| = 11$, the trap vertex and the vertices in I have the same degree, so the outcome depends on the deterministic tie-breaking rule. For $|I| \geq 12$, the vertices in I have smaller degree than the trap vertex, so Greedy selects from I and finds the maximum independent set. This transition is visible in Figure D.2.

Overall, these two graph families show that the performance of the minimum-degree greedy algorithm is controlled by whether low degree is a reliable signal for a good choice. In the greedy-bad-chain graph, low degree systematically points toward a trap. In the trap-clique-independent graph, the same mechanism appears for small $|I|$, but disappears once the independent set vertices become the unique minimum-degree vertices. Thus, this chapter does not only identify cases where Greedy performs poorly; it also shows how a structural change can turn the same greedy rule from a failing strategy into a successful one.

4.2. Running Time of Greedy

We also analyze how the runtime of the minimum-degree greedy algorithm scales with the number of vertices n. For this implementation, the runtime increases because the algorithm repeatedly has to find a vertex of minimum degree and then remove this vertex and its neighbors from the graph. Since this update is performed many times, and each step may require checking a large part of the remaining graph, the runtime is expected to grow approximately quadratically in n.

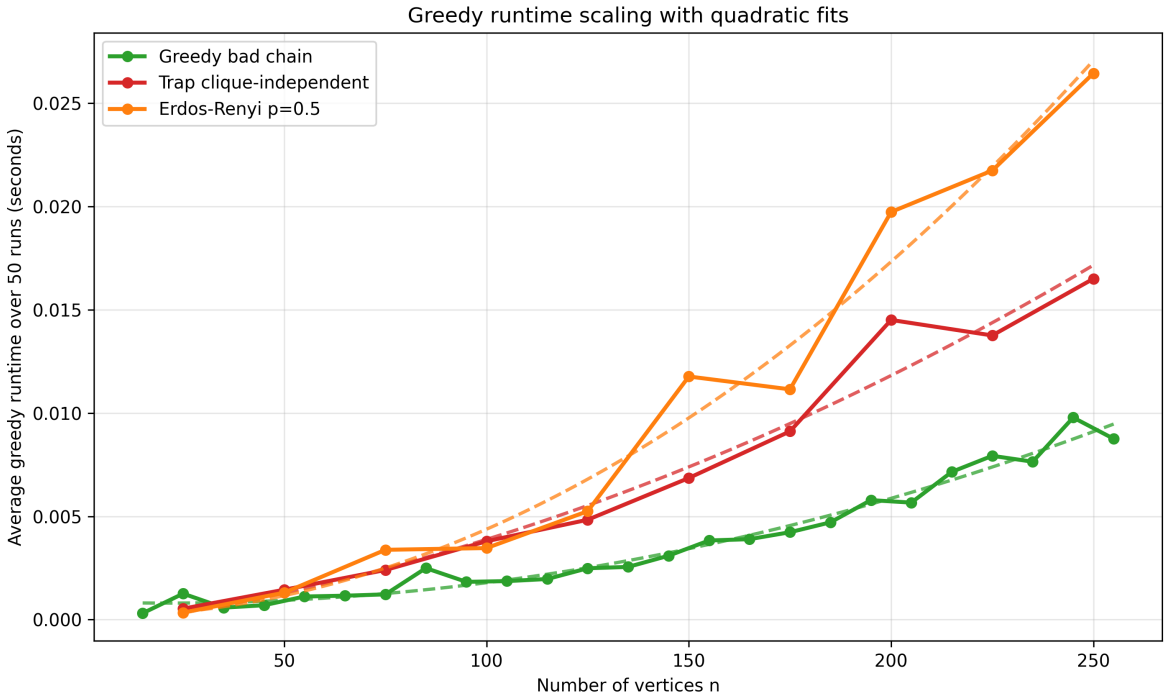


Figure 4.5: Average runtime of the deterministic Greedy algorithm over 50 runs per graph size, shown for the greedy-bad-chain, trap-clique-independent, and Erdős–Rényi $p = 0.5$ graph families. Solid lines show the measured mean runtime, while dashed lines show quadratic fits of the form $T(n) = a + bn + cn^2$.

Figure 4.5 shows the runtime of the minimum-degree greedy algorithm for the greedy-bad-chain, trap-clique-independent, and Erdős–Rényi $p = 0.5$ graph families. To understand the scaling behavior, we compared four possible runtime models: linear, quadratic, exponential, and power-law fits.

Using R^2 as the goodness-of-fit measure, the quadratic model gave the best fit for every tested graph family. This means that, over the tested range, the runtime grows approximately quadratically with n. This is also what we expect from the implementation, and this result is consistent with the theoretical runtime bound of $O(n^2)$.

Additional graph families are shown in Figure D.3 in the appendix.

5

Evaluation of Lotka–Volterra Dynamics

This chapter evaluates the Lotka–Volterra method introduced in Chapter 3. The goal is not only to identify graph families where the method returns smaller independent sets, but also to understand which structural or dynamical mechanisms may explain this behavior.

The Lotka–Volterra method differs from Greedy and simulated annealing because it evolves all vertex variables simultaneously. A vertex is not selected immediately. Instead, neighboring vertices continuously suppress each other through the competition term, and the final independent set is extracted from the limiting state. This means that the outcome can depend on both the graph structure and the initial condition.

We investigate two related explanations. First, we test whether clustering is a major reason for poor Lotka–Volterra performance. Second, when clustering alone does not fully explain the results, we study whether the initial condition and the number of competing maximal independent sets influence the final outcome.

5.1. Clustering Hypothesis

As observed by Mooij [26], the Lotka–Volterra algorithm performs poorly on random geometric graphs. We hypothesize that this poor performance is partly caused by the interaction between the initial conditions and the local structure of these graphs. In particular, unfavorable initial conditions may cause the Lotka–Volterra algorithm to converge to maximal independent sets that are significantly smaller than the maximum independent set found by the exact algorithm.

The reason this effect may be stronger in clustered graphs is the following. Unlike Greedy, the Lotka–Volterra method does not immediately select one vertex and remove its neighbors. Instead, all vertex variables evolve simultaneously, and neighboring vertices suppress each other continuously through the competition term. Suppose that a vertex which should not belong to the final independent set is assigned a relatively high initial value, for example 0.9. During the early dynamics, this vertex can strongly suppress its neighbors before the system has settled.

In a clustered graph, such a vertex is often connected to many vertices in the same local region. Its influence may therefore spread through a dense neighborhood rather than affecting only a few isolated vertices. This can push several competing vertices close to zero early in the simulation. As a result, the dynamics may become biased toward a suboptimal local configuration before the final independent set is extracted.

This mechanism is different from the failure mechanism of Greedy. Greedy fails when it makes a discrete local choice based on the minimum-degree rule that later turns out to block a better solution. In contrast, the Lotka–Volterra method may fail because continuous suppression during the early dynamics amplifies the effect of unfavorable initial values.

We therefore test the following hypothesis:

> **Hypothesis 1.** The Lotka–Volterra algorithm performs worse on graphs with a high average local clustering coefficient.

To test this hypothesis, we compare the performance of the Lotka–Volterra algorithm on graphs with low clustering coefficients and graphs with high clustering coefficients. For both types of graphs, we compare the size of the independent set found by the Lotka–Volterra algorithm with the size found by exhaustive enumeration and with the results of the other heuristic algorithms.

5.2. Clustering Coefficient Theory

To study clustering in graphs, we use the average local clustering coefficient. This coefficient measures how often the neighbors of a vertex are also connected to each other. In other words, it quantifies how close the local neighborhood of a vertex is to forming a clique.

For a vertex v, let d_v denote the degree of v, meaning the number of neighbors of v. Let e_v denote the number of edges that exist between those neighbors. The local clustering coefficient of v is defined as

$$C_v = \begin{cases} \dfrac{2e_v}{d_v(d_v-1)}, & d_v \geq 2, \\ 0, & d_v < 2. \end{cases}$$

Here, d_v denotes the degree of vertex v, while d will later be used for the dimension of the underlying geometric space.

This formula compares the number of actual edges between the neighbors of v with the number of possible edges between them. If v has d_v neighbors, then the maximum possible number of edges between these neighbors is

$$\binom{d_v}{2} = \frac{d_v(d_v-1)}{2}.$$

Therefore, C_v measures the fraction of possible edges between the neighbors of v that are actually present. If C_v is close to 1, then the neighbors of v are almost all connected to each other, so the neighborhood of v is very dense. If C_v is close to 0, then the neighbors of v are mostly not connected to each other.

For vertices with degree smaller than 2, the local clustering coefficient is set to 0. This is a convention: a vertex needs at least two neighbors before there can be an edge between its neighbors. If a vertex has zero or one neighbor, there is no pair of neighbors whose connection can be checked.

The average local clustering coefficient of a graph is obtained by averaging the local clustering coefficients over all vertices:

$$C_{\text{avg}}(G) = \frac{1}{|V|}\sum_{v\in V} C_v.$$

This gives a measure of the typical local clustering experienced by a vertex in the graph.

For random geometric graphs, clustering is an important structural property. In these graphs, vertices represent points in an underlying geometric space, and two vertices are connected if their distance is at most a fixed radius r. Because of this geometric construction, two neighbors of the same vertex are often close to each other as well. Therefore, they are more likely to be connected, which creates many triangles and gives random geometric graphs relatively high local clustering.

In the large graph limit, the expected local clustering coefficient of a random geometric graph is mainly determined by the dimension d of the underlying space [9]. This does not mean that the neighborhood of a vertex is no longer a ball. In a random geometric graph, the neighbors of a vertex are still the points lying within distance r, so geometrically they lie inside a d-dimensional ball. However, the dimension affects how points are distributed inside this ball and how much two radius-r neighborhoods overlap.

To see this, consider a vertex v and two neighbors u and w of v. Both u and w lie within distance r of v. They are also adjacent to each other only if their mutual distance is at most r. Equivalently, after choosing one neighbor u, the second neighbor w must lie in the overlap of two d-dimensional balls: the ball of radius r around v and the ball of radius r around u. The relative size of this overlap depends on the dimension d. Therefore, the dimension influences the probability that two neighbors of the same vertex are also connected.

For finite graph instances, the measured clustering coefficient can also be affected by parameters such as the number of vertices n, the connection radius r, and boundary effects. By boundary effects, we mean that for a vertex close to the boundary of the domain, part of its radius-r ball lies outside

the sampling region. As a result, the part of the neighborhood ball from which sampled vertices can become neighbors is smaller than it would be for a vertex in the interior. This can affect local degrees and, consequently, the measured clustering coefficient in finite samples.

The reason we use the average local clustering coefficient in this thesis is that the Lotka–Volterra dynamics are local. Each vertex competes directly with its neighbors, and the local structure around a vertex can influence how this competition evolves. A high average local clustering coefficient means that many vertices lie in locally dense regions. In such regions, neighboring vertices are often also connected to each other, which may create stronger local interactions and more complicated competition patterns.

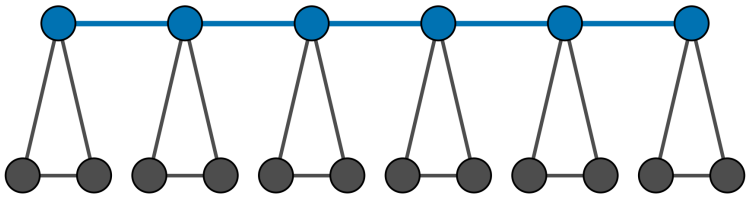
There is also another common measure of clustering, called the global clustering coefficient. It is defined by

$$C_{\text{global}}(G) = \frac{3T}{\sum_{v \in V} \binom{d_v}{2}},$$

where T is the number of triangles in the graph. This coefficient measures the fraction of connected triples that are closed into triangles. A connected triple consists of one vertex connected to two other vertices. If those two other vertices are also connected, then the triple is closed and forms part of a triangle. The factor $3T$ appears because each triangle contains three connected triples, one centered at each of its three vertices.

Although both coefficients describe clustering, they do not measure it in exactly the same way. The average local clustering coefficient first measures clustering around each individual vertex and then averages these values. This means that every vertex contributes equally to the final value. By contrast, the global clustering coefficient counts triples over the whole graph. Vertices with high degree contribute many more triples than vertices with low degree, so the global clustering coefficient gives more weight to high-degree vertices.

Figure 5.1 illustrates this difference using a path-of-cliques graph. This example has locally dense clique regions, but these cliques are connected by bridge vertices. As a result, most vertices have highly connected neighborhoods, while the bridge structure creates many open triples. The average local clustering coefficient is therefore larger than the global clustering coefficient.



$C_{\text{avg}} = 0.741 \quad C_{\text{global}} = 0.429$

Figure 5.1: Example of a path-of-cliques graph, consisting of six triangles connected in a chain by single bridge edges. The blue vertices and edges show the bridge structure between cliques, while the dark gray vertices form highly clustered local triangles. This graph illustrates why the average local clustering coefficient can be much larger than the global clustering coefficient: most vertices have highly connected neighborhoods, but the bridge vertices create many open triples. For this example, $C_{\text{avg}} = 0.741$ and $C_{\text{global}} = 0.429$.

This example shows why the average local clustering coefficient is the more relevant measure for this thesis. Since the Lotka–Volterra dynamics are based on local competition, we are mainly interested in the typical neighborhood structure experienced by individual vertices. The global clustering coefficient is still a useful graph statistic, but it can be more strongly influenced by high-degree vertices and therefore does not always reflect the local clustering experienced by a typical vertex.

5.3. Testing the Clustering Hypothesis

We first test Hypothesis 1 on random geometric graphs with radius $r = 0.35$. This graph family is a natural starting point because random geometric graphs have local spatial structure, and Mooij observed

that Lotka–Volterra can perform poorly on them [26]. In this experiment, we use $\tau = 1.1$ and compare the solution sizes found by Lotka–Volterra, Greedy, simulated annealing, and exhaustive enumeration for $n = 12, \ldots, 20$.

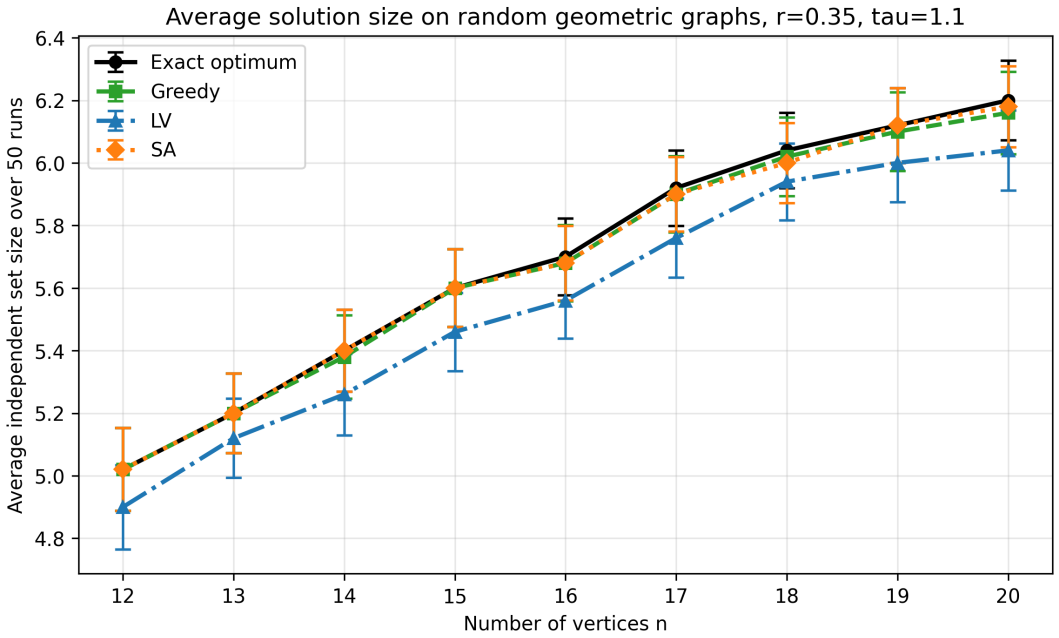


Figure 5.2: Average solution size on random geometric graphs with $r = 0.35$, $\tau = 1.1$, and $n = 12, \ldots, 20$. The Lotka–Volterra algorithm finds smaller independent sets than exhaustive enumeration and the other heuristic algorithms. Several curves overlap because Greedy and simulated annealing often match the exact optimum, or differ from it only by a very small amount. Thus, parts of the Greedy, SA, and optimum curves are hidden behind one another, while the Lotka–Volterra curve lies slightly below them for most n.

Figure 5.2 shows that the Lotka–Volterra algorithm returns slightly smaller independent sets than the exact optimum on these random geometric graphs. The difference is not extremely large, but it is consistent across the tested values of n. Moreover, the Lotka–Volterra algorithm also performs slightly worse than Greedy and simulated annealing on the same graph instances.

This suggests that the poorer performance is not only caused by the graph size, but may also be related to the structure of random geometric graphs. In a clustered region, early suppression by an initially favored vertex may bias the system toward a suboptimal maximal independent set.

At the same time, this figure alone does not prove that graph structure is the only reason for the lower performance. The outcome of the Lotka–Volterra method can also depend on parameter choices such as τ, the initial conditions, the integration time, and the numerical threshold used to extract the independent set. Therefore, we interpret the result as evidence that the structure of random geometric graphs may contribute to the weaker performance of Lotka–Volterra, rather than as a complete explanation by itself. The parameter experiments later in this thesis are used to investigate this further.

Figure D.4 in the appendix shows that this random geometric graph family, for $n = 12, \ldots, 20$, has an average local clustering coefficient of approximately 0.6. This is relatively high, meaning that the neighborhoods of vertices are often densely connected.

We next consider other graph types with high clustering coefficients.

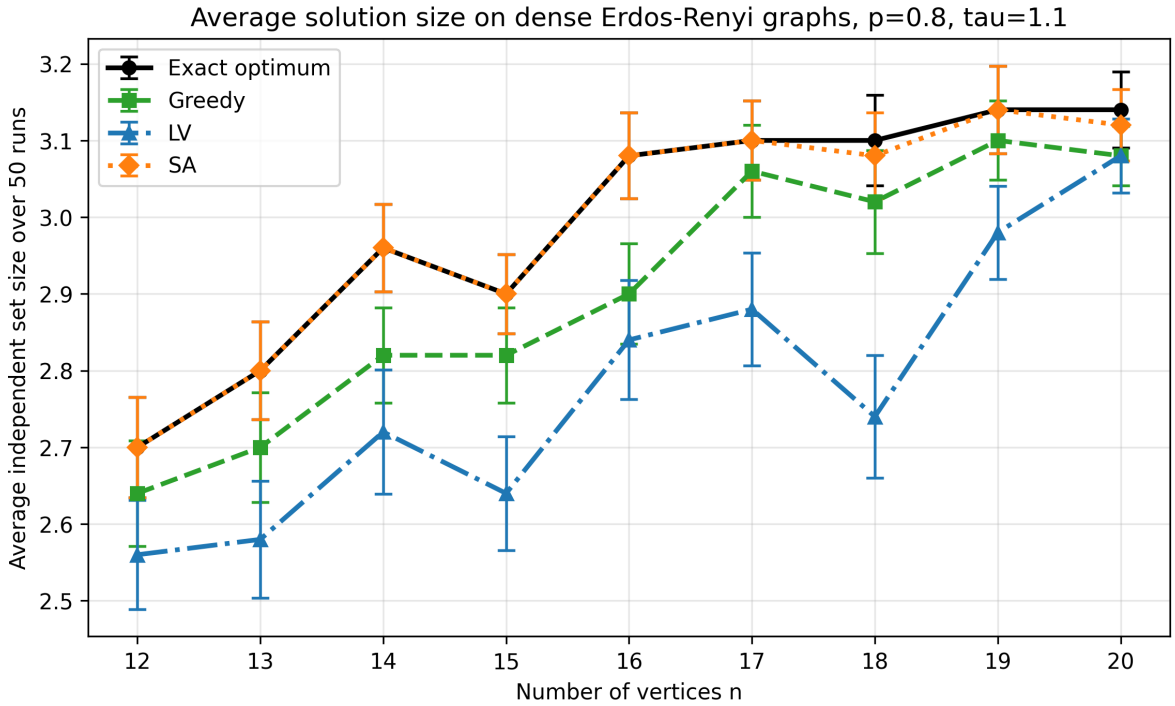


Figure 5.3: Average solution size on dense Erdős–Rényi graphs with $p = 0.8$ and $\tau = 1.1$ for the Lotka–Volterra algorithm. The Lotka–Volterra algorithm again finds smaller independent sets than exhaustive enumeration and the other heuristic algorithms. For this dense Erdős–Rényi plot, the overlap is mainly between the exact optimum and simulated annealing: SA matches the exact optimum for most n, and when it differs, the difference is very small. Greedy also lies close to the optimum, so parts of the optimum, Greedy, and SA curves may be hidden behind one another; the Lotka–Volterra curve is the one that remains visibly lower for most n.

Figure D.5 in the appendix shows that the dense Erdős–Rényi graphs have an average clustering coefficient close to 0.8 in the tested range. Figure 5.3 shows that the Lotka–Volterra algorithm also returns smaller independent sets on these dense Erdős–Rényi graphs. This is consistent with Hypothesis 1: in a highly clustered graph, an incorrectly favored vertex can suppress several neighbors that together may have formed a larger independent set.

A similar effect is seen for the high-degree random regular graph in Figure D.6 in the appendix. This graph has an average clustering coefficient of about 0.75, as shown in Figure D.7, and the Lotka–Volterra algorithm again performs much worse than the exact algorithm. Since the vertices in random regular graphs have equal degrees, this suggests that the poor performance is not only caused by degree differences.

So far, the examples are consistent with the clustering hypothesis. However, to test whether high clustering is really the main explanation, we also need to consider graphs with low clustering coefficients. If Hypothesis 1 is correct, then the Lotka–Volterra solution sizes should lie closer to the exact maximum independent set sizes in these low-clustering cases.

We first test this on random geometric graphs with $r = 0.2$. In these finite graph instances, lowering the radius reduces the number of edges and also lowers the measured average local clustering coefficient.

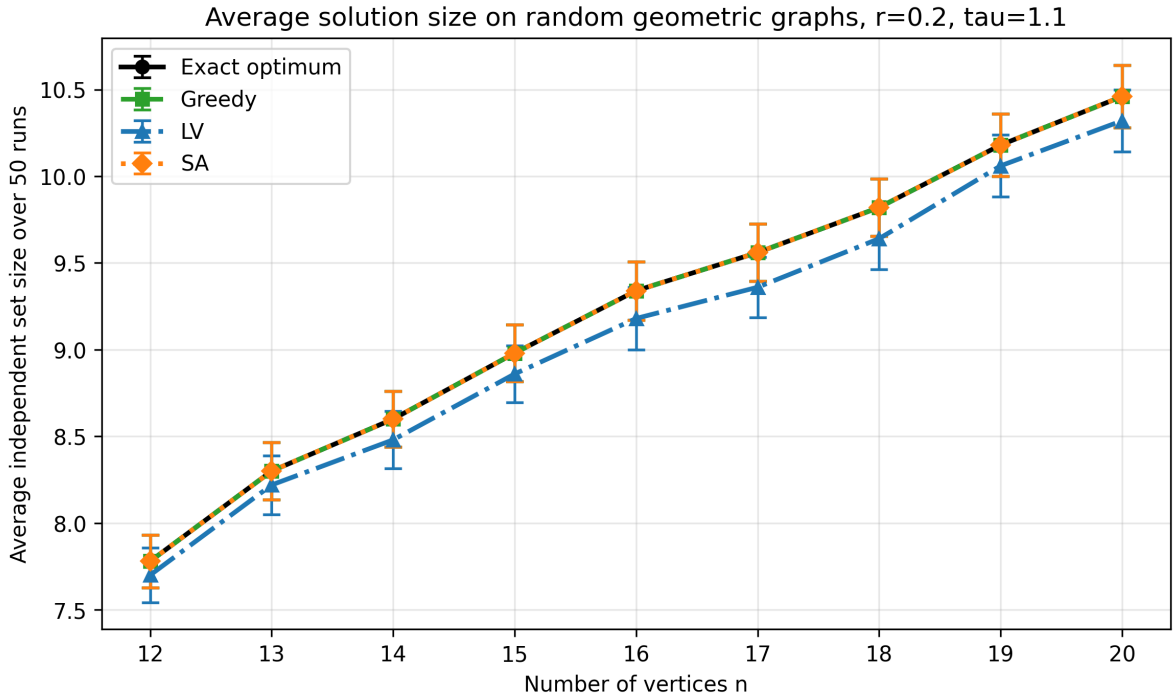


Figure 5.4: Average solution size on random geometric graphs with $r = 0.2$, $\tau = 1.1$, and the standard algorithm settings. The exact, greedy, and simulated annealing algorithms almost overlap, while the Lotka–Volterra algorithm still finds smaller independent sets. Note that the y-axis scale differs from the previous plots.

Figure D.8 in the appendix shows that lowering the radius also lowers the measured average clustering coefficient in these finite experiments. In Figure 5.4, at first sight, the Lotka–Volterra algorithm seems to improve, since its solution size lies closer to the exact optimum. However, the y-axis scale has also changed, so this visual difference should be interpreted carefully. This means that we cannot conclude from this plot alone that the Lotka–Volterra algorithm improved because the clustering coefficient decreased.

We next test sparse Erdős–Rényi graphs with $p = 0.1$, which also have low clustering.

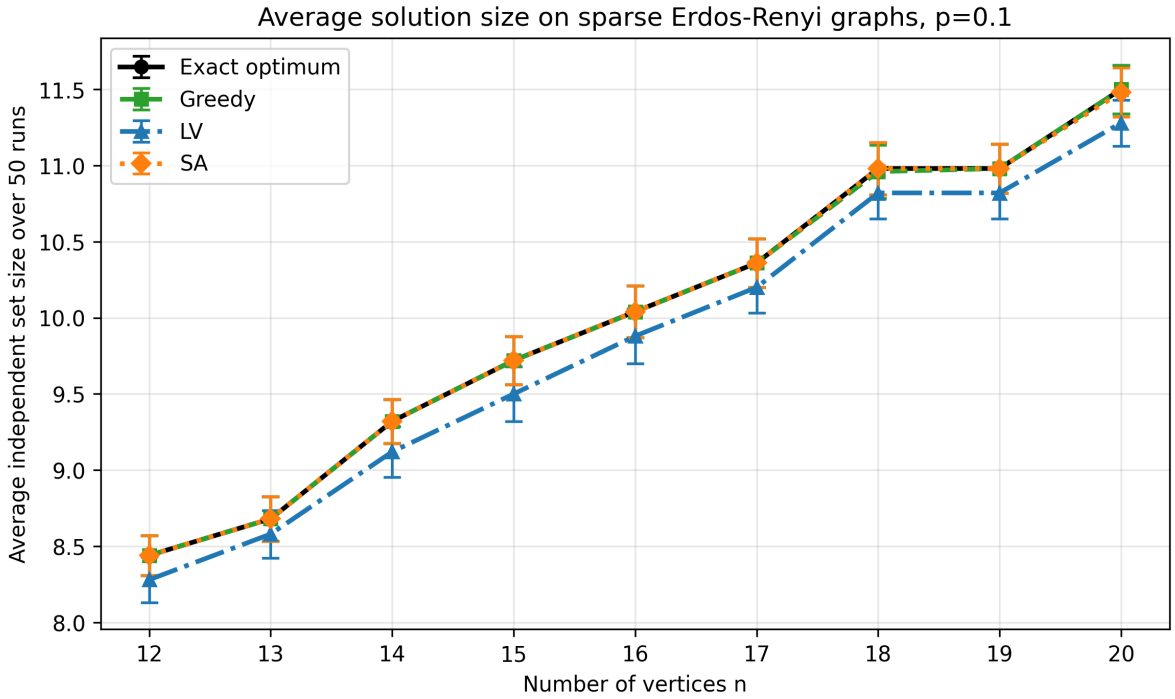


Figure 5.5: Average solution size on sparse Erdős–Rényi graphs with $p = 0.1$, $\tau = 1.1$, and the standard algorithm settings. The greedy and simulated annealing algorithms find values close to the exact maximum independent set, while the Lotka–Volterra algorithm still performs clearly worse.

Figures 5.5 and D.9 in the appendix show that the Lotka–Volterra algorithm can still perform poorly even when the average clustering coefficient is low. This means that the results do not clearly support the hypothesis that high clustering is the main reason for the poor Lotka–Volterra performance.

These results do not rule out clustering as a contributing factor, but they suggest that clustering alone is not enough to explain the behavior. A similar pattern is observed for the low-degree random regular graph in Figures D.10 and D.11 in the appendix. Additional graph families are also tested in the appendix in Figures D.12 and D.13. Thus, other graph properties may be influencing the Lotka–Volterra dynamics.

5.4. Initial Conditions and Competing Maximal Sets

The clustering experiments show that high clustering can be associated with poor Lotka–Volterra performance, but that clustering alone does not fully explain the behavior. The poor performance of the Lotka–Volterra algorithm therefore remains interesting. In particular, we want to understand whether the initial condition plays an important role in the maximal independent set found by the dynamics.

This leads to a second hypothesis:

> **Hypothesis 2.** The Lotka–Volterra dynamics are strongly influenced by the initial condition, especially when there are many competing maximal independent sets.

We analyze this behavior for random geometric graphs. We first ask the following question:

> *How often does the Lotka–Volterra algorithm converge to the maximal independent set that is closest to the initial condition?*

Here, “closest” means closest in Euclidean distance between the initial condition x_0 and the indicator vector of a maximal independent set.

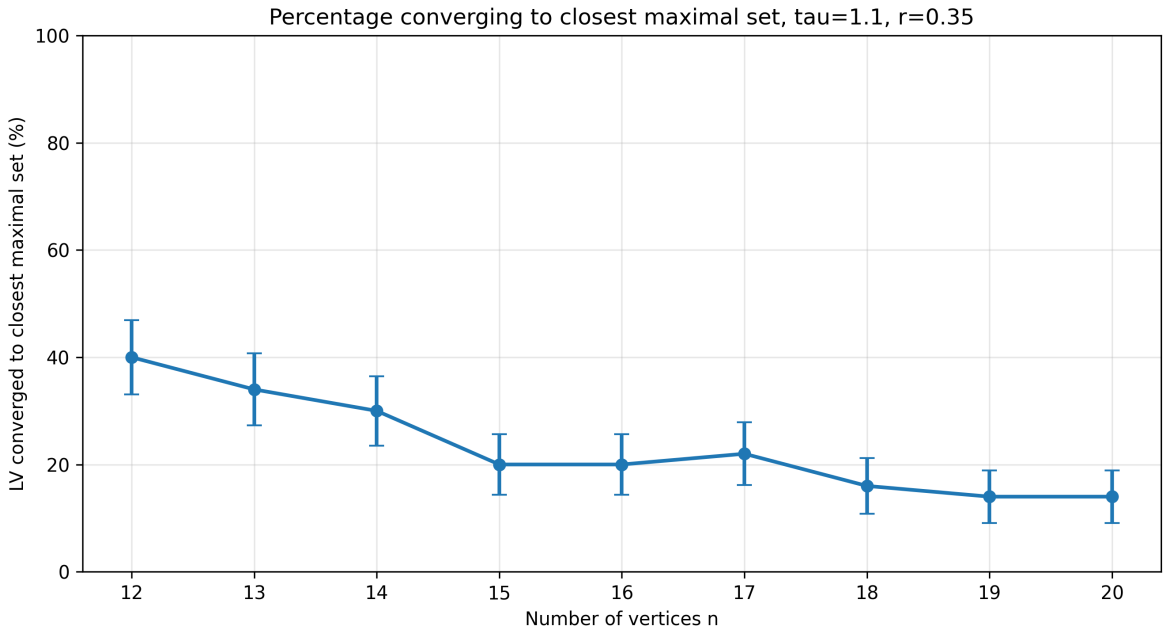


Figure 5.6: Percentage of runs in which the Lotka–Volterra algorithm converges to the closest maximal independent set, for random geometric graphs with $r = 0.35$, $\tau = 1.1$, and $n = 12, \ldots, 20$. The LV dynamics are initialized with independent random positive values $x_i(0) \sim \mathrm{Uniform}(0.000001, 0.999999)$ for each vertex.

Figure 5.6 shows that the Lotka–Volterra algorithm regularly converges to the closest maximal independent set. For $n = 12$, this happens in about 40% of the runs. However, this percentage decreases as n increases. This suggests that the initial condition has an influence, but that the relation between the initial condition and the final maximal set becomes less direct for larger graphs.

The previous figure only shows how often the Lotka–Volterra algorithm selects the closest maximal independent set exactly. However, even when it does not select the closest one, the selected set may still be very close to the initial condition. To check this, we compare the distance from x_0 to the closest maximal independent set with the distance from x_0 to the maximal independent set selected by the Lotka–Volterra algorithm.

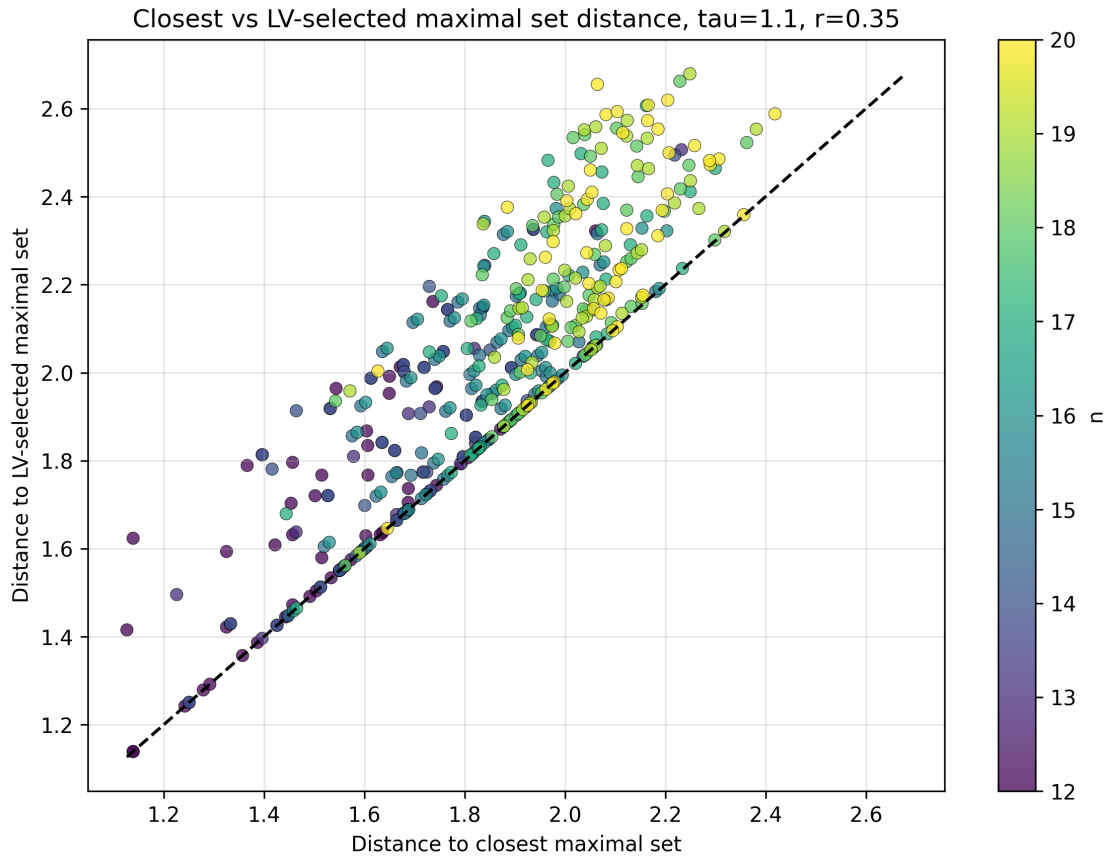


Figure 5.7: Distance from the initial condition x_0 to the closest maximal independent set compared with the distance from x_0 to the maximal independent set selected by the Lotka–Volterra algorithm. The colors indicate different values of n.

Figure 5.7 shows that LV does not always converge to the closest maximal independent set in Euclidean distance from the initial condition. However, many selected maximal sets still lie at comparable distances to the closest one, suggesting that distance to the initial condition substantially influences the outcome. A related rank plot is included in Figure D.14 in the appendix. Together, these plots show that as n increases, the algorithm converges less often to the nearest maximal set.

This leads to three follow-up questions. First, if the algorithm does not converge to the nearest maximal set, does it at least converge to a larger maximal set? Second, if it converges to a maximal set of the same size, is this set still almost as close to the initial condition as the nearest one? Third, why does the value of n influence how often the algorithm converges to the closest maximal independent set?

We first answer the first question.



Figure 5.8: Comparison of the size of the Lotka–Volterra selected set in the cases where it does not converge to the nearest maximal independent set. The plot shows the percentage of cases where the selected set is larger, equal in size, or smaller than the nearest maximal set.

Figure 5.8 shows that when the Lotka–Volterra algorithm does not converge to the nearest maximal set, it often converges to a larger one. Importantly, in these experiments it does not converge to a smaller set than the nearest maximal set.

For the remaining cases, where the selected maximal set has the same size as the nearest one, we check whether the two sets are similar in distance from the initial condition.

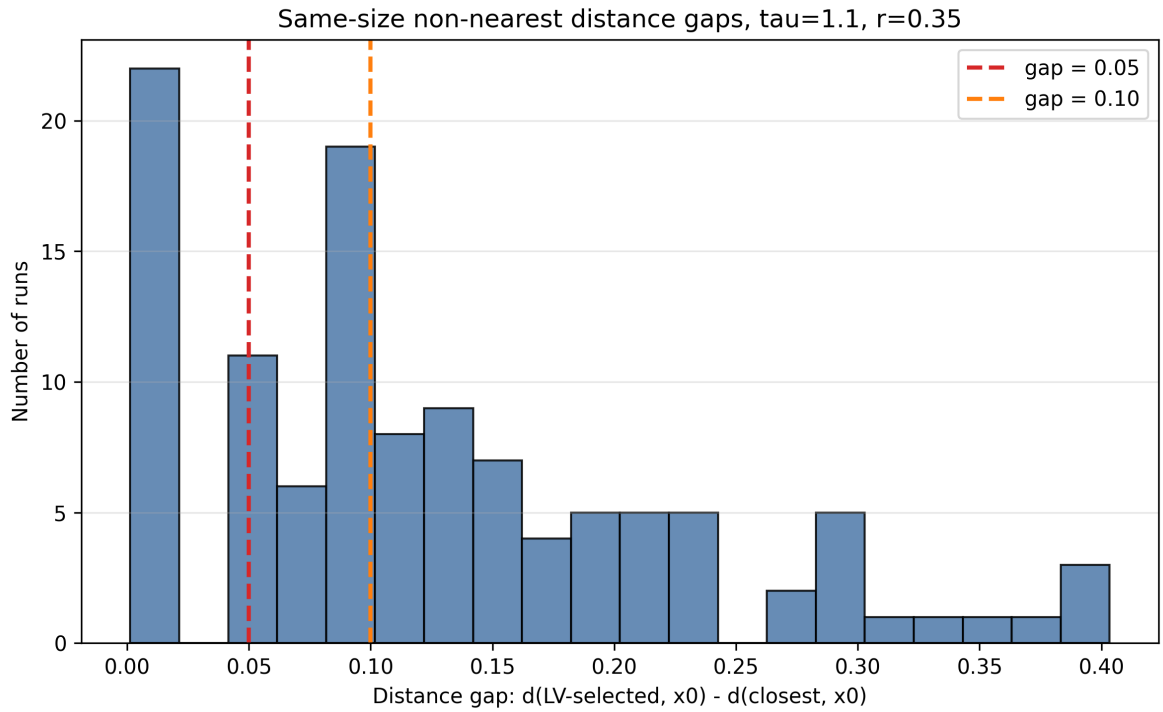


Figure 5.9: Histogram of the distance gap for cases where the Lotka–Volterra algorithm does not converge to the nearest maximal independent set and instead converges to a different maximal independent set of the same size. Cases where LV converges to a larger maximal independent set are excluded. The gap is defined as the distance from the initial condition x_0 to the LV-selected set minus the distance from x_0 to the nearest maximal independent set.

Figure 5.9 shows that in many of these same-size cases, the selected maximal set is still very close to the nearest one. In about half of these cases, the distance gap is at most 0.1. This means that even when the algorithm does not select the closest maximal set, it often selects another set that is almost equally close to the initial condition.

Overall, we find the following breakdown over all 450 runs:

Closest maximal set	:	$105/450 = 23.3\%$,
Not closest, but larger	:	$230/450 = 51.1\%$,
Not closest, same size, gap ≤ 0.1	:	$58/450 = 12.9\%$,
Residual cases (same size, larger gap)	:	$57/450 = 12.7\%$.

Thus, in 87.3% of all runs, the Lotka–Volterra algorithm either converges to the closest maximal set, skips it but finds a larger maximal set, or skips it and finds a same-size maximal set with a very similar distance from the initial condition. This suggests that the initial condition still plays an important role. Even when the algorithm does not converge to the nearest maximal set, it usually converges to a set that is either better in size or almost equally close to the initial condition.

We now return to the third question:

> *Why does the value of n influence how often the algorithm converges to the closest maximal independent set?*

For larger n, several things change. First, the number of maximal independent sets increases. Each additional vertex creates more possible combinations, so the closest maximal set is only one option among many. As a result, the Lotka–Volterra algorithm has more competing attractors. In addition, for larger n, there are often several maximal sets whose distance to the initial condition is very similar.

Second, the closest maximal set is not necessarily the one to which the dynamics most easily converge. Euclidean distance from x_0 to a $\{0,1\}$-vector is only a geometric notion of closeness. The Lotka–Volterra dynamics are also influenced by the graph structure and the interaction between vertices.

To investigate this, we consider three questions. First, how does the number of maximal independent sets grow with n? Second, does the Lotka–Volterra algorithm converge more often to the nearest maximal set when there are fewer maximal sets available? Third, does the distance gap between the closest and second closest maximal set decrease as n increases?

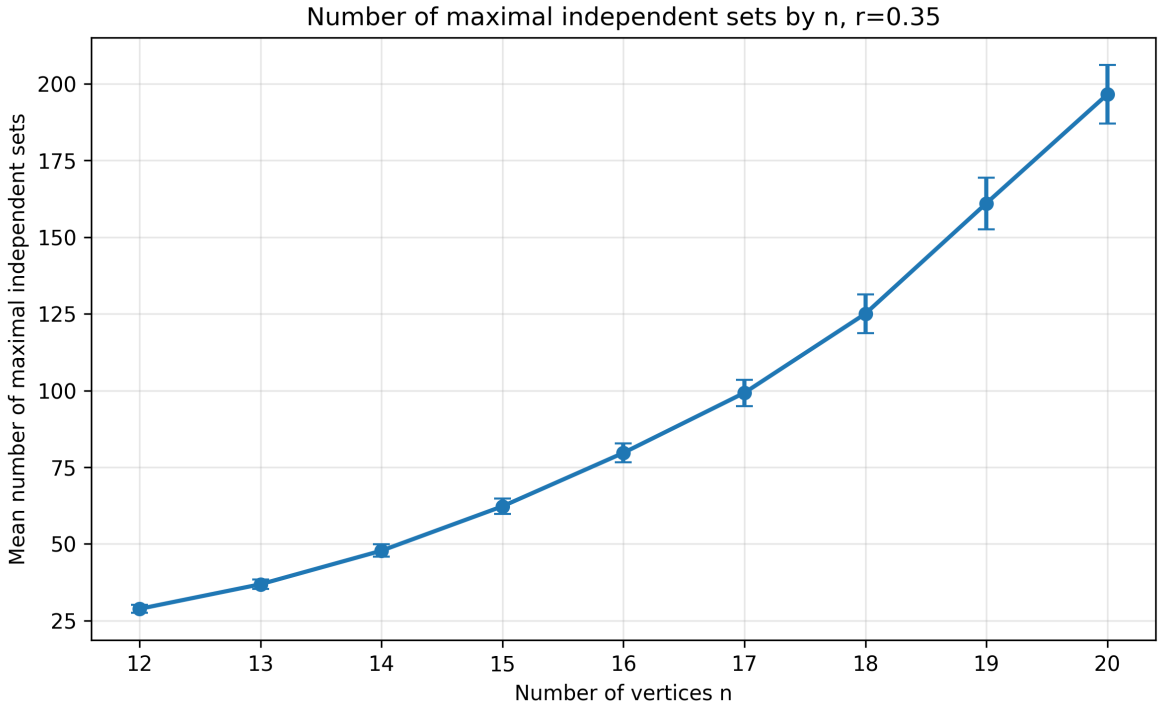


Figure 5.10: Number of maximal independent sets as a function of n, for random geometric graphs with $r = 0.35$ and $\tau = 1.1$.

Figure 5.10 shows that, for the random geometric graphs tested here, the number of maximal independent sets increases rapidly with n. Empirically, this growth appears close to exponential over the tested range. This means that for larger graphs, the Lotka–Volterra dynamics have many more possible maximal sets to converge to.

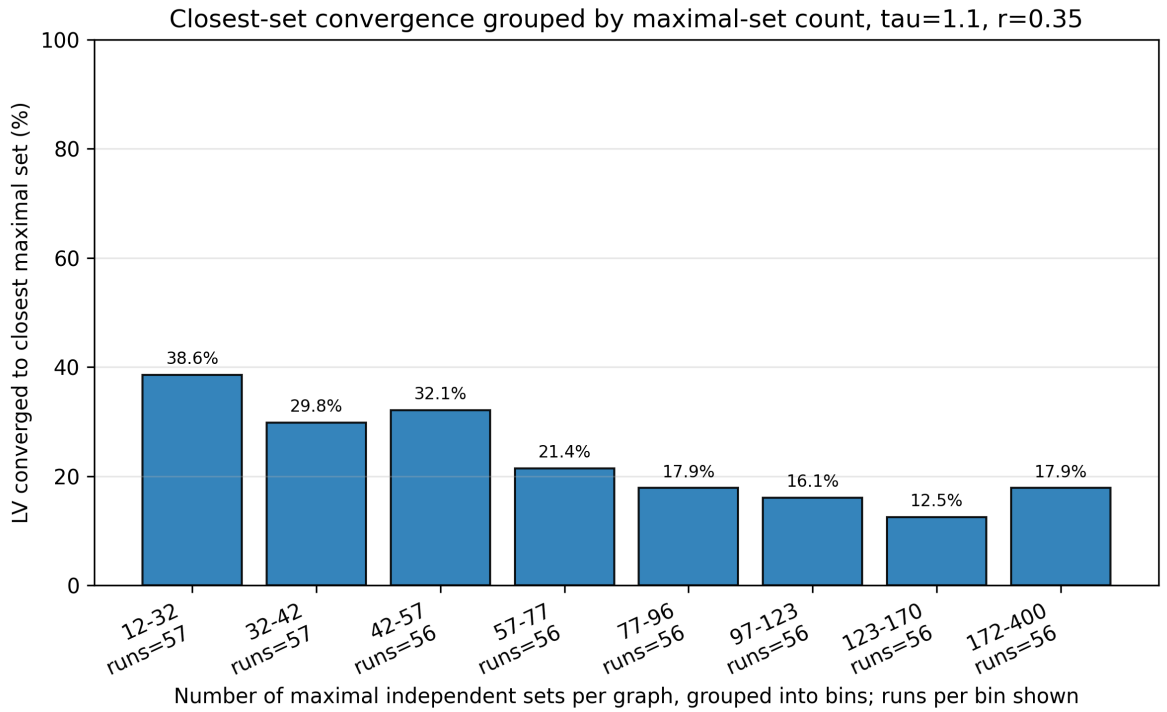


Figure 5.11: Percentage of runs in which the Lotka–Volterra algorithm converges to the maximal independent set closest to the initial condition x_0, grouped by the total number of maximal independent sets in the graph. Each bar contains a bin of graph instances with a similar number of maximal-set counts; the label below each bar gives the range of maximal-set counts and the number of runs in that bin. The decreasing trend suggests that, when a graph has more competing maximal independent sets, LV is less likely to converge to the one nearest to x_0.

Figure 5.11 supports this idea. The Lotka–Volterra algorithm converges more often to the nearest maximal independent set when there are fewer maximal sets available. When the number of maximal sets increases, the nearest one becomes only one of many possible attractors, so convergence to it becomes less likely.

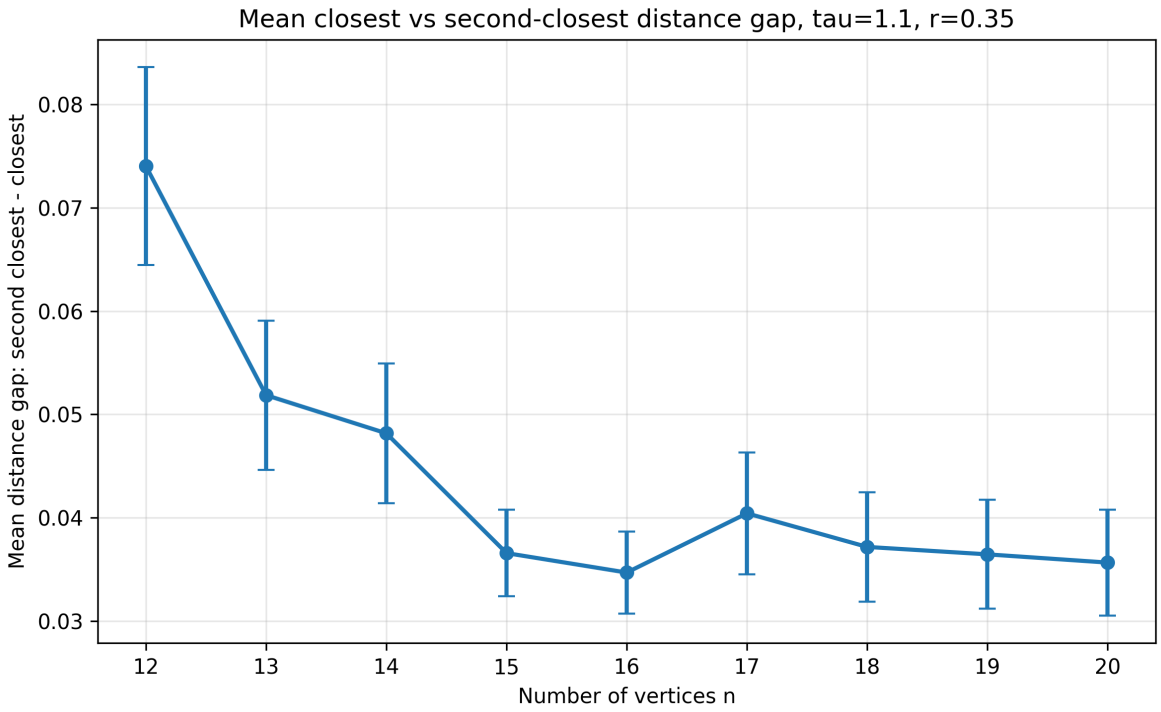


Figure 5.12: Distance gap between the closest and second closest maximal independent set as a function of n.

Figure 5.12 shows that the distance gap between the closest and second closest maximal independent set tends to decrease as n increases. This means that, for larger graphs, several maximal sets may be almost equally close to the initial condition. Therefore, even a small difference in the dynamics can determine which maximal set is selected.

We should also be careful when comparing the distance gap for different values of n. The plotted quantity is the difference between the distance to the closest candidate solution and the distance to the second-closest candidate solution. As n increases, the number of possible maximal independent sets also increases. Therefore, it becomes more likely that there are several candidate solutions close to the final Lotka–Volterra state, which can make the closest and second-closest distances more similar and hence reduce the gap.

At the same time, the distances are computed as Euclidean distances between vectors in $\mathbb{R}^n$. Since the dimension of this space increases with n, the absolute scale of these distances can also change with n. This scaling effect may partly counteract the tendency of the gap to shrink, which could explain why the mean gap becomes relatively flat for $n = 15, \ldots, 20$ in Figure 5.12.

Overall, the results show that the failure of the Lotka–Volterra algorithm cannot be explained by clustering alone. High clustering is often correlated with poor performance, but the algorithm can also fail on graphs with low clustering. This suggests that other structural properties of the graph also influence the dynamics. However, the experiments with the nearest maximal independent set support Hypothesis 2: the initial condition plays an important role, especially when many competing maximal independent sets are available. In many cases, the algorithm converges either to the closest maximal set, to a larger maximal set, or to a set with a very similar distance from the initial condition. The main failure moments therefore seem to occur when the dynamics are pulled toward a locally stable maximal set that is not maximum, especially when many competing maximal sets are available.

6

Evaluation of Simulated Annealing

For the maximum-independent-set problem, simulated annealing is expected to struggle when the energy landscape contains many suboptimal local minima, especially when escaping from these minima requires temporarily moving to worse configurations. In simple terms, simulated annealing can often escape small traps, but it becomes less reliable when the landscape contains many competing valleys separated by barriers.

Planted-clique graph. A planted-clique graph is constructed by first generating an Erdős–Rényi random graph and then selecting a subset of vertices whose missing internal edges are added. This forces the selected vertices to form a clique. An example is shown in Figure 6.1. In this thesis, this construction is useful because, after taking the complement graph, the planted clique becomes a planted independent set. This gives a controlled way to test how the algorithms behave when the graph contains deliberately planted structure.

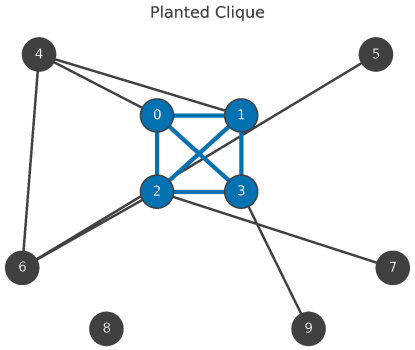


Figure 6.1: Example of a planted-clique graph. The red vertices form a clique that has been planted inside a larger graph by adding all missing edges between them. Note that an Erdős–Rényi random graph does not need to be connected.

In the small exact-test regime, where $n = 12, \ldots, 20$, the exact optimum is still available. This makes it possible to compare the heuristic algorithms directly with the exact algorithm. In this range, simulated annealing with 3000 steps performs reasonably well on most of the tested graph families, including random geometric graphs, Erdős–Rényi graphs, planted-clique graphs, random regular graphs of different degrees, trap-clique-independent graphs, and complete bipartite-type graphs.

We include two representative examples here: planted-clique graphs, shown in Figures D.15 and D.16, and dense Erdős–Rényi graphs, shown in Figures D.17 and D.18 in the appendix.

Figures D.15 and D.16 show that simulated annealing is robust on planted-clique graphs in the tested range. It usually finds the maximum independent set, or at least a maximal independent set whose size is close to the optimum. Overall, these small-graph experiments suggest that simulated annealing performs well for $n = 12, \ldots, 20$ when the cooling schedule is slow enough. With 3000 steps, the algorithm has enough time to explore the energy landscape, and we do not find a graph family where it consistently fails. However, this does not mean that the same schedule remains sufficient for

larger graphs. When n increases while the cooling schedule is kept fixed, the performance decreases, suggesting that larger instances require slower cooling.

We next test simulated annealing on larger planted-clique instances. For these larger graphs, computing the exact maximum clique size becomes impractical. Therefore, we use a theoretical estimate for the clique number of an Erdős–Rényi graph as a proxy. The clique number $\omega(G)$ is the size of the largest clique in a graph G. For $G(n, 1/2)$, it is known that the clique number is approximately $2\log_2 n$ with high probability [3]. Here, "with high probability" means that the probability of this statement being true tends to 1 as $n \to \infty$.

We therefore use $2\log_2 n$ as an estimate for the size of the largest clique that would naturally occur in the Erdős–Rényi background graph. We then plant a clique of roughly twice this size by selecting a set of vertices and adding all missing edges between them. This creates a planted clique that is larger than the cliques expected to occur naturally in the random background graph.

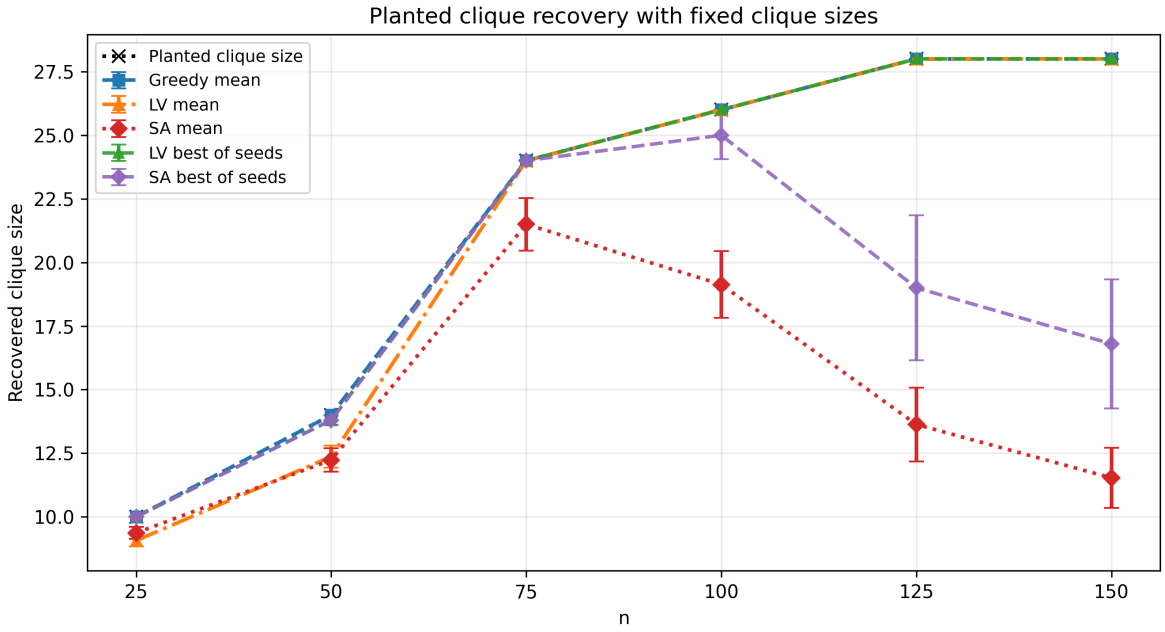


Figure 6.2: Average recovered clique size on planted-clique instances. The plot compares the greedy algorithm, Lotka–Volterra, best-of-three Lotka–Volterra, simulated annealing, best-of-three simulated annealing, and the planted clique size estimate. Simulated annealing uses 3000 steps. Several curves overlap because Greedy recovers the planted clique size exactly for all tested n, while Lotka–Volterra and best-of-three Lotka–Volterra coincide with the planted-size line for most larger instances. Thus, these curves may be hidden behind the planted clique size estimate, whereas the simulated annealing curves separate more clearly for larger n.

Figure 6.2 shows that simulated annealing with 3000 steps performs worse as n increases. This suggests that the fixed cooling schedule becomes too fast for larger graphs.

To check whether this is specific to planted-clique instances, we also test Erdős–Rényi graphs with $p = 0.5$, as shown in Figure D.19 in the appendix.

In these larger graph experiments, simulated annealing again performs worse as n increases. This leads to three possible explanations. First, the energy barriers may become larger. By an energy barrier, we mean the temporary increase in energy that the algorithm must accept in order to move from one local minimum to a better region of the search space. Here, a better region means a part of the search space with access to lower-energy configurations, which usually correspond to larger independent sets or configurations with fewer edge conflicts. If these barriers become larger, then the probability of crossing them decreases unless the temperature is high enough. Second, the number of competing local minima may increase, so the algorithm has more suboptimal states in which it can get stuck. Third, the cooling schedule may simply be too fast for larger graphs, giving the algorithm too little time to explore before the temperature becomes low.

The third explanation is partly expected: when n increases, the number of variables also increases, so a fixed number of steps gives fewer update attempts per variable. This effect is studied separately

in Section 7.2. In this section, we focus on the first two explanations: the role of energy barriers and the number of available local minima.

A complete bipartite graph $K_{a,b}$ is a bipartite graph whose vertices are split into two parts of sizes a and b. All possible edges between the two parts are present, while there are no edges within either part. Thus, each part is an independent set.

To isolate the effect of energy barriers, we use complete bipartite graphs with $a < b$. In such a graph, there are exactly two maximal independent sets: the part of size a and the part of size b. The larger part is the maximum independent set.

Suppose that simulated annealing reaches the binary configuration corresponding to the smaller part, meaning that all vertices in the part of size a are selected and all vertices in the part of size b are unselected. To move from this configuration to the better solution, the algorithm has to transform the selected set into the other part. Since simulated annealing changes the configuration by flipping one bit at a time, this requires either removing selected vertices from the smaller part or temporarily selecting vertices from the larger part while vertices from the smaller part are still selected. Both options can temporarily increase the energy: removing a selected vertex reduces the size of the independent set, while selecting a vertex from the other part creates edge conflicts.

In this section, we use the term *QUBO barrier* for the temporary increase in the QUBO Hamiltonian $H(x)$ that has to be crossed when moving from one binary configuration to another by single-bit flips. More precisely, consider a path of binary configurations

$$x^{(0)}, x^{(1)}, \dots, x^{(m)},$$

where each consecutive pair differs in one bit. The barrier height along this path is

$$\max_{0 \le k \le m} H(x^{(k)}) - H(x^{(0)}).$$

The QUBO barrier between two configurations is the smallest such barrier height over all one-bit-flip paths connecting them. In the complete bipartite experiment below, we are interested in the QUBO barrier between the smaller maximal independent set and the larger maximum independent set.

This gives a simple test case where the number of maximal independent sets stays fixed at two, while the QUBO barrier between the smaller and larger independent set can grow with n.

This allows us to ask:

> *Does simulated annealing struggle more when the QUBO barrier between the smaller maximal independent set and the maximum independent set grows, even if the number of maximal independent sets stays fixed?*

We also scale the number of simulated annealing steps with n. Specifically, we test two linear schedules: $20n$ and $50n$ steps. These values are chosen as empirical short and long schedules, rather than as theoretically optimal choices. The reason for using schedules proportional to n is that each simulated annealing step flips one randomly chosen bit. Thus, with cn steps, each variable is selected approximately c times on average. The schedules $20n$ and $50n$ therefore correspond to about 20 and 50 update attempts per variable, respectively.

This avoids comparing larger graphs with the same fixed number of updates. If we kept the number of steps fixed at 3000, then two effects would change at once: the QUBO barrier would grow, but the number of update attempts per variable would decrease. By using $20n$ and $50n$ steps, we partially control for the increasing number of variables while still allowing us to compare a shorter and a longer cooling schedule.

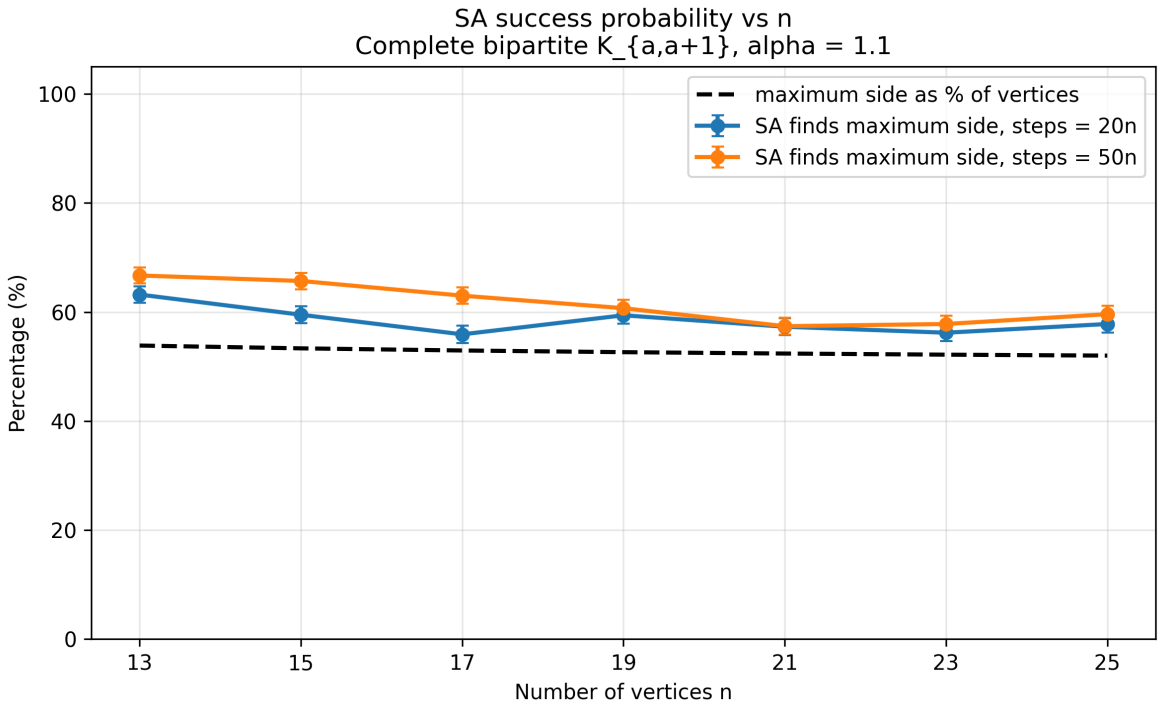


Figure 6.3: Probability that simulated annealing finds the maximum independent set in the odd complete bipartite graph $K_{a,a+1}$, where $n = 2a + 1$. The x-axis shows the total number of vertices n, and the y-axis shows the percentage of runs that return the larger part of the bipartition, which is the unique maximum independent set. The blue line uses a cooling schedule of $20n$ SA steps, while the orange line uses $50n$ SA steps; each point is averaged over 1000 runs, with error bars showing the standard error. The black dashed line shows the fraction of vertices belonging to the maximum part, $(a + 1)/(2a + 1)$, which is slightly above 50% and decreases toward 50% as n grows.

To confirm that this test does increase the QUBO barrier, we also plot the QUBO barrier as a function of n, as shown in Figure D.20 in the appendix.

The complete bipartite test gives a clean comparison: the number of maximal independent sets remains fixed at two, while the QUBO barrier increases with n. In Figure 6.3, we compare two cooling lengths, $20n$ and $50n$, over 1000 runs for each odd n. The dashed black line shows the fraction of vertices that belong to the maximum part of the bipartition, which is only slightly above 50% and decreases toward 50% as n grows.

For the shorter schedule $20n$, the probability of finding the larger part stays close to this baseline. This suggests that the algorithm has little time to explore the landscape and is strongly influenced by the random initial condition and early stochastic choices. In this regime, simulated annealing often settles into either part of the bipartition before the growing QUBO barrier can produce a clear additional effect.

For the longer schedule $50n$, simulated annealing performs better for smaller n, finding the larger part more often than in the $20n$ case. However, as n increases and the QUBO barrier grows, this advantage decreases, and the success probability moves closer to the $20n$ curve. This indicates that the growing QUBO barrier does have an effect once the algorithm is given enough time to explore, but the effect is modest in this simple two-basin landscape.

These results combine all initial conditions. To examine the barrier effect more directly, we also separate runs by whether the initial condition is closer to the smaller maximal set or to the maximum independent set. For a binary configuration x and an independent set S, we measure closeness by the Euclidean distance between x and the indicator vector χ_S. This is shown in Figure D.21 in the appendix. Runs starting closer to the maximum independent set do not need to overcome the same QUBO barrier, while runs starting closer to the smaller maximal set must move away from that initially favored solution to reach the optimum. This split therefore tests whether larger QUBO barriers make this recovery harder. The split shows that larger barriers mainly reduce success when the initial condition starts closer to the smaller maximal set, supporting the idea that recovery from an initially unfavorable

direction becomes harder as the barrier grows.

Overall, these results suggest that a larger escape barrier alone is not sufficient to explain simulated annealing failure. In the complete bipartite graph, there are only two maximal independent sets, so the landscape remains very simple. Simulated annealing appears to struggle more seriously when a growing barrier is combined with many competing local minima. Thus, the difficulty is better explained by the interaction between barrier height, cooling length, and the number of available maximal independent sets, rather than by barrier height alone.

We now test the refined hypothesis more directly:

> *If the cooling schedule and the QUBO barrier are fixed, does increasing the number of local minima make simulated annealing perform worse?*

To study this, we construct a graph family in which the total graph size, the size of the global optimum, the size of the trap solutions, and the QUBO barrier for escaping from a trap are all kept fixed, while only the number of competing trap solutions, meaning second-best maximal independent sets, is increased.

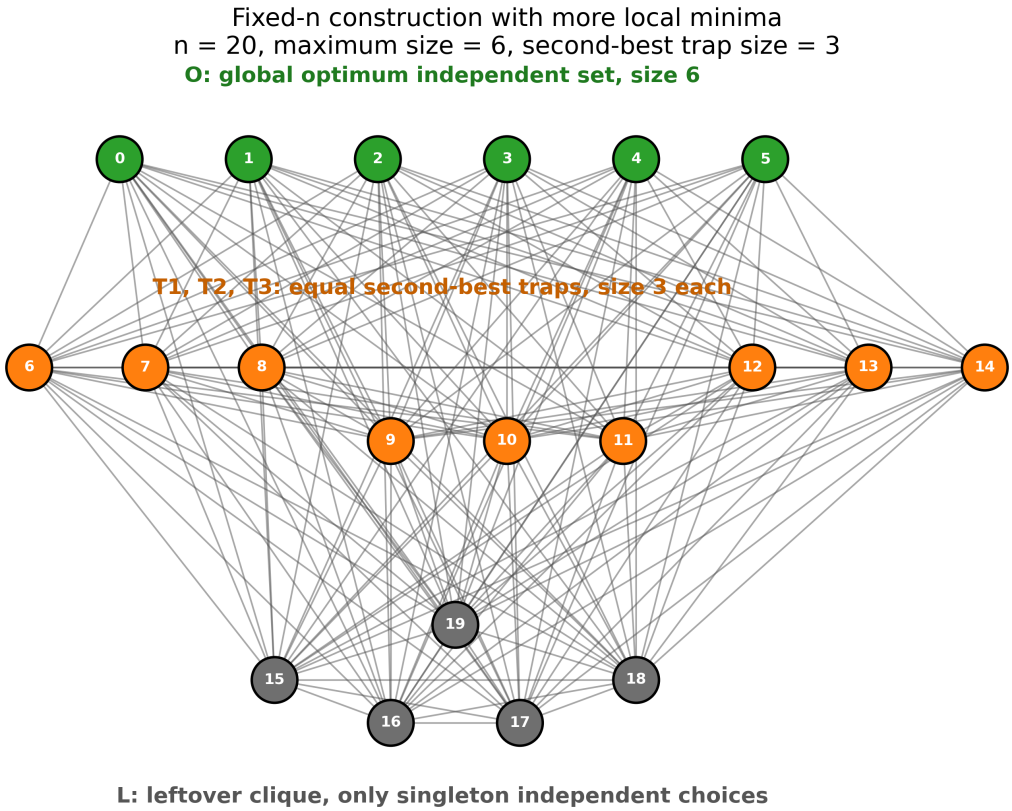


Figure 6.4: Example of the fixed-n trap construction. The block $O = \{0, 1, 2, 3, 4, 5\}$ is the unique maximum independent set. The blocks $T_1 = \{6, 7, 8\}$, $T_2 = \{9, 10, 11\}$, and $T_3 = \{12, 13, 14\}$ are equal-size second-best maximal independent sets, which act as competing local minima. The remaining vertices $L = \{15, 16, 17, 18, 19\}$ form a clique, so they only contribute singleton maximal independent sets and do not create additional large solutions. The vertex labels explicitly show the block membership, making the construction interpretable without relying only on color. Edges are added between every pair of vertices in different blocks, while there are no edges inside O or inside any T_i.

For the experiment, we fix the total graph size at $n = 100$. We choose a global optimum block O of size 30, and trap blocks T_i of size 10. The remaining vertices form a clique L. The graph is constructed so that there are no edges inside O, no edges inside any T_i, the leftover vertices L form a clique, and every vertex in one block is connected to every vertex in every other block.

With this construction, the important maximal independent sets are O of size 30, the trap blocks T_i of size 10, and singleton vertices from L.

Thus, the global optimum has fixed size 30, and each trap has fixed size 10. We consider five cases, where the number of trap blocks increases from 1 to 5, while the total graph size $n = 100$, the optimum

size, the trap size, and the size gap remain fixed. The QUBO barrier for escaping from each trap also remains the same by construction. Here, moving from a trap to the optimum means transforming the binary configuration that selects all vertices of one trap block into the binary configuration that selects all vertices of the optimum block, using the one-bit flips of simulated annealing. Since each trap block is disjoint from the optimum block, the algorithm must first remove the selected trap vertices before the optimum vertices can be added without creating conflicts. Because every trap block has the same size, this required temporary loss in solution size is the same for each trap. In this way, the main change between the five cases is the number of competing second-best maximal independent sets.

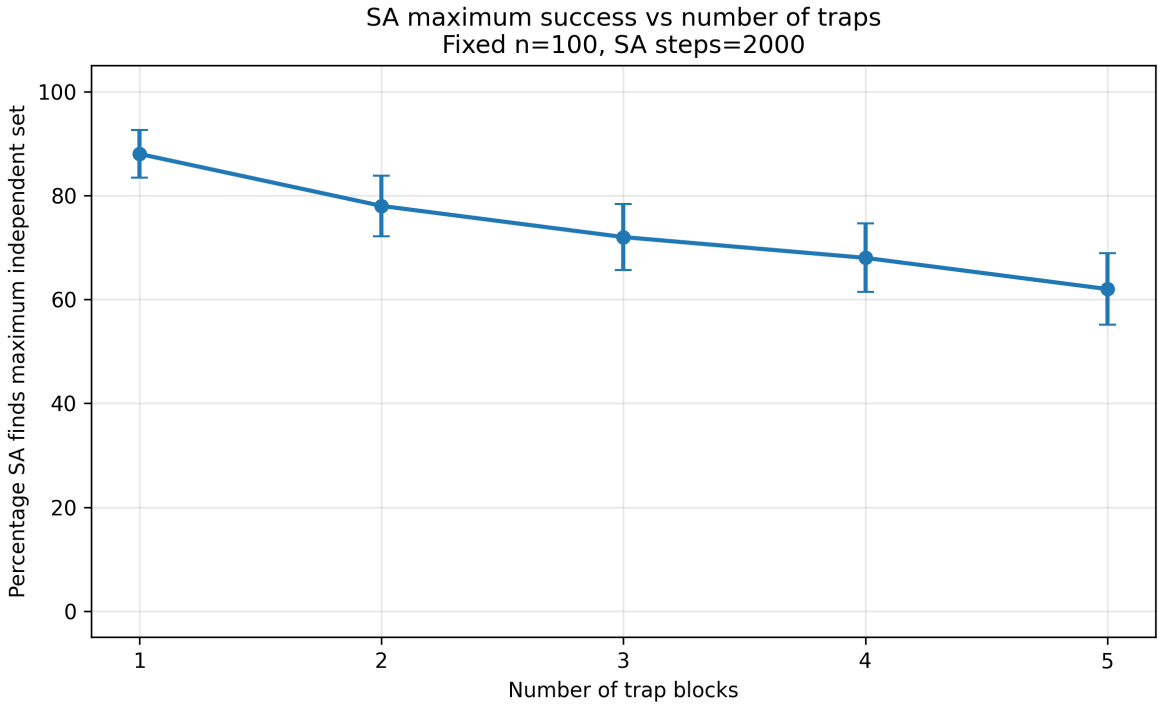


Figure 6.5: Percentage of simulated annealing runs that recover the maximum independent set as the number of trap blocks increases. The graph size, maximum independent set size, trap size, size gap, QUBO barrier, and cooling schedule are kept fixed. The decrease in success rate shows that adding more competing local minima makes simulated annealing less likely to find the global optimum.

Figure 6.5 shows a clear trend: as the number of trap blocks increases, the probability that simulated annealing finds the maximum independent set decreases. Since the main structural quantities are fixed, this indicates that the number of competing local minima itself has a strong influence on performance.

One possible concern is that this effect might simply be caused by an unsuitable cooling schedule. To address this, we check whether simulated annealing still converges to maximal solutions, rather than failing completely.

Figure D.22 in the appendix confirms that the cooling schedule is appropriate for this experiment: simulated annealing does converge to maximal independent sets, but as the number of trap blocks increases, it is more often pulled toward one of the traps instead of the global optimum. In other words, the drop in success is not caused by complete failure to converge, but by convergence to the wrong maximal solution.

To further validate the construction, we also check that the sizes of the maximum and trap solutions remain unchanged across all five cases.

Figure D.23 in the appendix confirms that the graph family behaves as intended: the optimum size and trap sizes stay fixed, so the experiment isolates the effect of increasing the number of competing local minima.

Overall, these results suggest that simulated annealing is more strongly affected by the number of available maximal independent sets than by a single large QUBO barrier alone. When there are more competing trap solutions, the probability of finding the global optimum decreases, even though

the graph size, QUBO barrier, and solution sizes remain fixed. This is consistent with the idea that simulated annealing struggles most in crowded energy landscapes with many competing local minima.

7

Effect of Parameter Choices

7.1. Changing τ

We now test how the value of the competition parameter τ influences the performance of the Lotka–Volterra algorithm. In the previous experiments, we used $\tau = 1.1$. Here, we compare this with a larger value, $\tau = 10$.

We hypothesize that increasing τ causes the Lotka–Volterra algorithm to find smaller independent sets. This is because τ controls how strongly neighboring vertices suppress each other. The dynamics are given by

$$\frac{dx_i(t)}{dt} = x_i(t)\left(1 - x_i(t) - \tau \sum_{j \sim i} x_j(t)\right)$$

as seen in Equation 3.3. When τ is large, the suppression from neighboring vertices becomes much stronger. Therefore, a vertex with a relatively high initial value can suppress its neighbors much more aggressively. If the initial condition favors the wrong vertex, this can push the dynamics toward a smaller maximal independent set.

For example, suppose that two neighboring vertices have initial values $x_i(0) = 0.7$ and $x_j(0) = 0.2$. To illustrate the initial effect of the competition term, we compute the derivative of x_j at time $t = 0$, considering only the influence of vertex i on vertex j. For $\tau = 10$, this gives

$$\left.\frac{dx_j}{dt}\right|_{t=0} = 0.2(1 - 0.2 - 10 \cdot 0.7) = -1.24.$$

For $\tau = 1.1$, we instead get

$$\left.\frac{dx_j}{dt}\right|_{t=0} = 0.2(1 - 0.2 - 1.1 \cdot 0.7) = 0.006.$$

Thus, at the initial time, vertex j is pushed downward for $\tau = 10$, while for $\tau = 1.1$ it can still grow slightly. This example only describes the initial direction of motion, since the values of $x_i(t)$ and $x_j(t)$ change over time. Nevertheless, it illustrates how a larger value of τ can make early suppression much stronger. As a result, bad initial conditions may have a larger effect on the final selected set.

We test this on two graph families: Erdős-Rényi graphs and random geometric graphs.

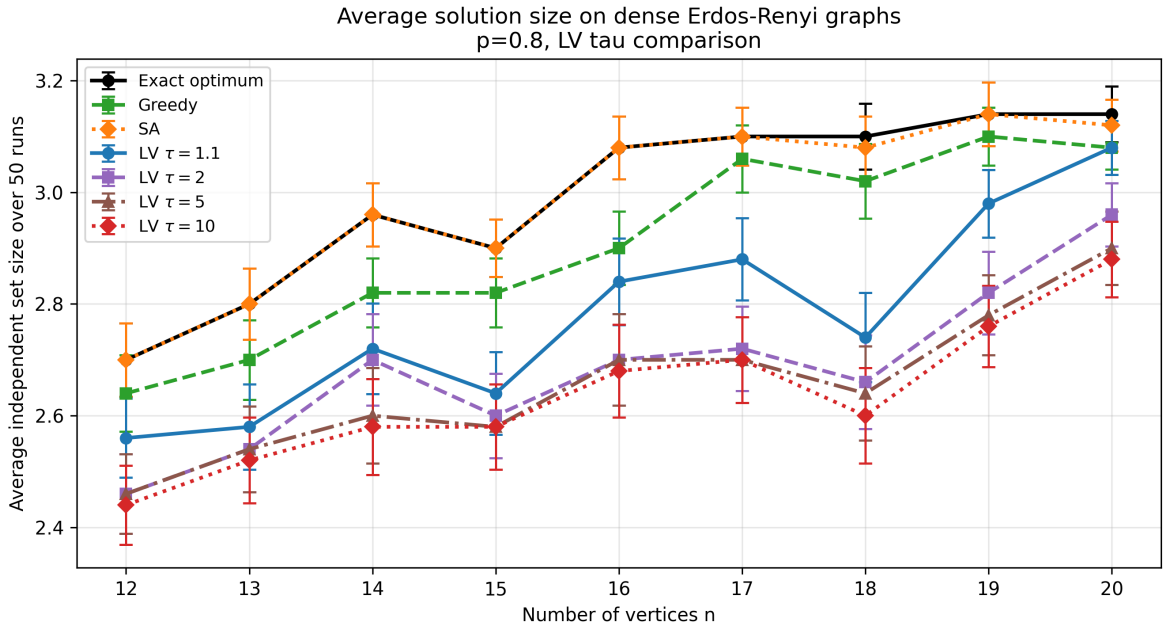


Figure 7.1: Average independent set size over 50 runs for each n on dense Erdős–Rényi graphs with edge probability $p = 0.8$. The plot compares the exact optimum, greedy algorithm, simulated annealing, and Lotka–Volterra with $\tau = 1.1$, $\tau = 2$, $\tau = 5$, and $\tau = 10$. Because these graphs are very dense, the maximum independent sets are small, so several curves lie close together. Simulated annealing almost overlaps with the exact optimum for most values of n, while the greedy curve remains slightly below it. The Lotka–Volterra curves are lower overall, and larger values of τ generally lead to smaller average selected independent sets.

Figure 7.1 shows that larger values of τ generally lead to smaller independent sets for the Lotka–Volterra algorithm. The effect is already visible for intermediate values such as $\tau = 2$ and $\tau = 5$, and becomes strongest for $\tau = 10$. Thus, the figure suggests that increasing the competition strength does not improve the solution quality in this setting.

Figure E.1 in the appendix shows a similar pattern on random geometric graphs. Again, larger values of τ tend to reduce the average independent set size found by the Lotka–Volterra algorithm.

Overall, these experiments suggest that increasing τ does not improve the performance of the Lotka–Volterra algorithm in our setting. Instead, larger values of τ make the dynamics more sensitive to early suppression. This can cause the algorithm to converge to smaller maximal independent sets.

In the remainder of this section, we focus on the two endpoint values $\tau = 1.1$ and $\tau = 10$. The value $\tau = 1.1$ is close to the theoretical threshold $\tau > 1$, while $\tau = 10$ represents much stronger competition between neighboring vertices. Comparing these two values allows us to study the contrast between weak and strong suppression more clearly, without adding too many curves to the diagnostic plots.

In this project, $\tau = 1.1$ is therefore the better choice among the tested values. As discussed in Section 3.3.10, we apply a final correction step to ensure that the output is an independent set. Because of this correction, increasing τ does not provide a clear benefit, while it does increase the risk of finding smaller solutions.

7.1.1. Convergence Time

We also test whether the parameter τ affects the convergence time of the Lotka–Volterra algorithm. From the previous experiments, we saw that larger τ often leads to smaller independent sets. However, it may also make the dynamics converge faster, because selected vertices suppress their neighbors more strongly.

Our hypothesis is therefore:

> *Increasing τ makes the Lotka–Volterra algorithm converge faster because local suppression becomes stronger.*

To test this, we compare the convergence time for $\tau = 1.1$ and $\tau = 10$. We define the convergence

time as the first time after which the thresholded Lotka–Volterra-selected independent set remains equal to the final selected set.

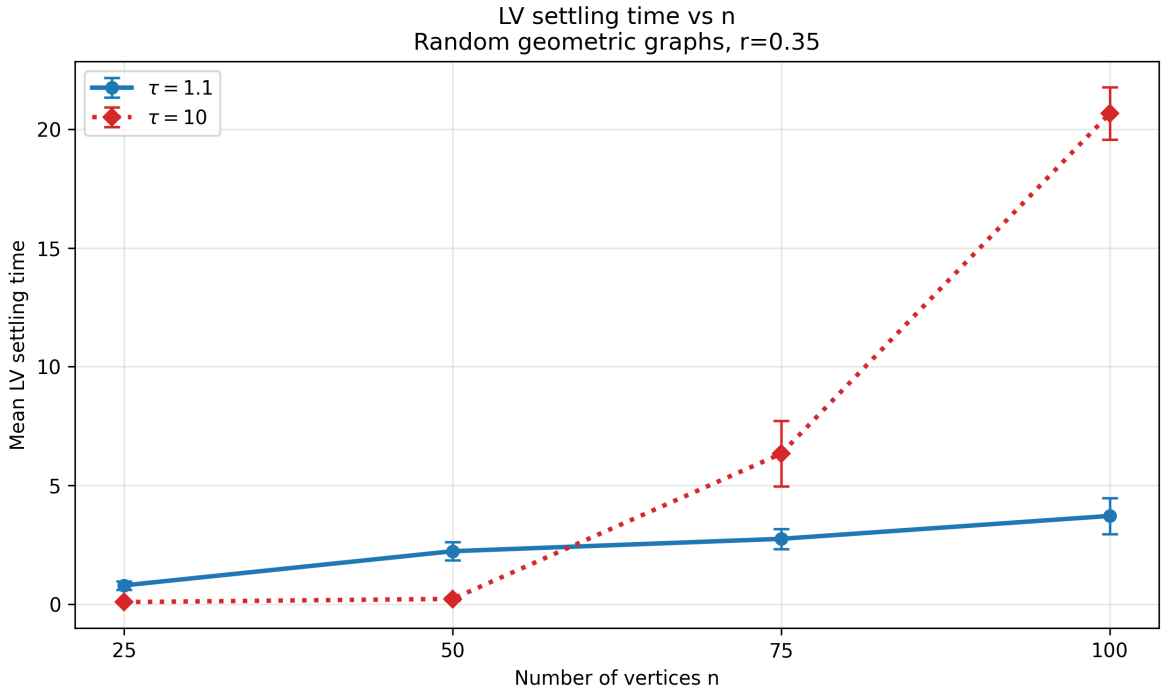


Figure 7.2: Mean Lotka–Volterra final-state convergence time on random geometric graphs with $r = 0.35$, averaged over 50 runs for each n. Time is measured in Euler integration time t: the simulation uses step size $\Delta t = 0.01$, so one unit on the vertical axis corresponds to 100 Euler steps. Convergence time is defined as the first time after which the thresholded Lotka–Volterra-selected independent set remains equal to the final selected set. The two curves compare $\tau = 1.1$ and $\tau = 10$, showing that larger τ does not always lead to faster final-state stabilization for larger graphs.

Figure 7.2 shows that the hypothesis is partly supported. For smaller values of n, $\tau = 10$ often reaches its final selected set faster than $\tau = 1.1$. This fits the intuition that stronger suppression makes the dynamics decide more quickly between neighboring vertices.

However, this effect does not hold uniformly for larger graphs. For larger n, especially on random geometric graphs, $\tau = 10$ does not always stabilize faster. In some cases, it even takes longer before the final thresholded independent set stops changing. Therefore, the effect of τ on convergence time is not simply that larger τ gives faster convergence. Similar results are obtained for random regular graphs of moderate degree, shown in appendix in Figure E.2.

We hypothesize that the convergence time is affected by how strongly the initial conditions influence the dynamics. When the suppression is strong, bad initial conditions can push the system toward a bad set early on. In a smaller graph, this bad direction is more likely to determine the final outcome, because there are fewer other vertices or competing sets that can counteract it. Therefore, the dynamics can converge relatively quickly, even if they converge to a poor solution.

For larger n, there are more vertices and more possible competing sets. As a result, even if the dynamics are initially pushed toward a bad set, other parts of the graph can still push back. The graph therefore becomes more resistant to one or a few bad initial conditions. However, this resistance also means that the dynamics may need more time to escape the initially chosen direction and settle into a final independent set.

For $\tau = 1.1$, the system is not pushed as strongly in the wrong direction at the beginning. Therefore, the effect of bad initial conditions is weaker, and the dynamics do not need the same kind of escape process.

To see this, we create the following Figure 7.3.

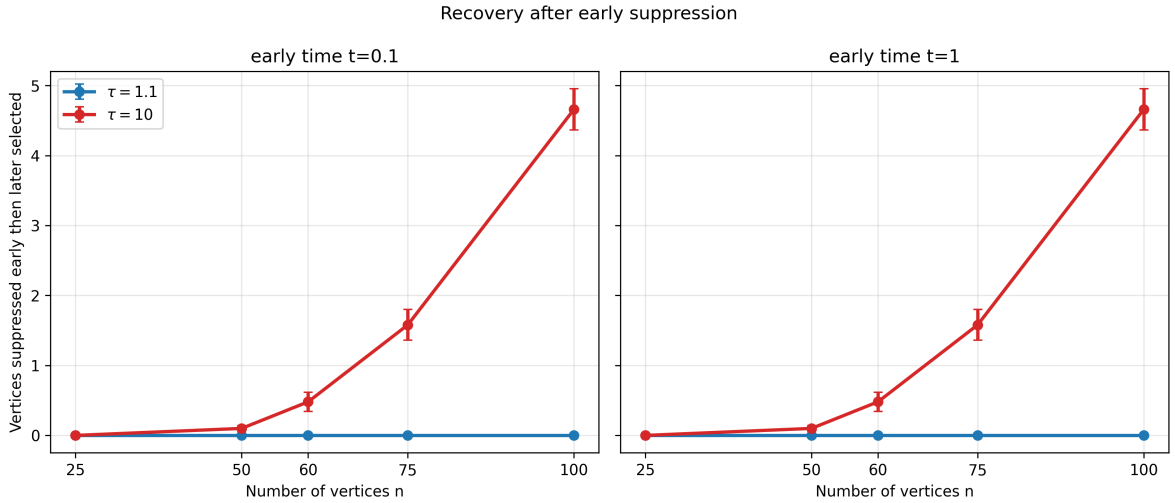


Figure 7.3: Mean number of vertices that were suppressed below the LV threshold at an early time but were selected again later in the trajectory, shown for random geometric graphs with $r = 0.35$. The two panels compare early suppression measured at $t = 0.1$ and $t = 1.0$, while the two lines compare $\tau = 1.1$ and $\tau = 10$. A larger value indicates that LV initially pushed more vertices down but later recovered them, suggesting stronger early commitment followed by later reorganization.

In Figure 7.3, we observe that for larger n, more vertices are suppressed early in the LV dynamics but later become selected. This suggests that in larger graphs, the dynamics can sometimes recover from strong early suppression. However, this recovery process also takes time: vertices that were initially pushed close to zero must regrow before the final independent set stabilizes. Therefore, larger τ may lead to faster early commitment, but for larger graphs it can also create stronger initial traps, causing the system to take longer to reorganize and converge.

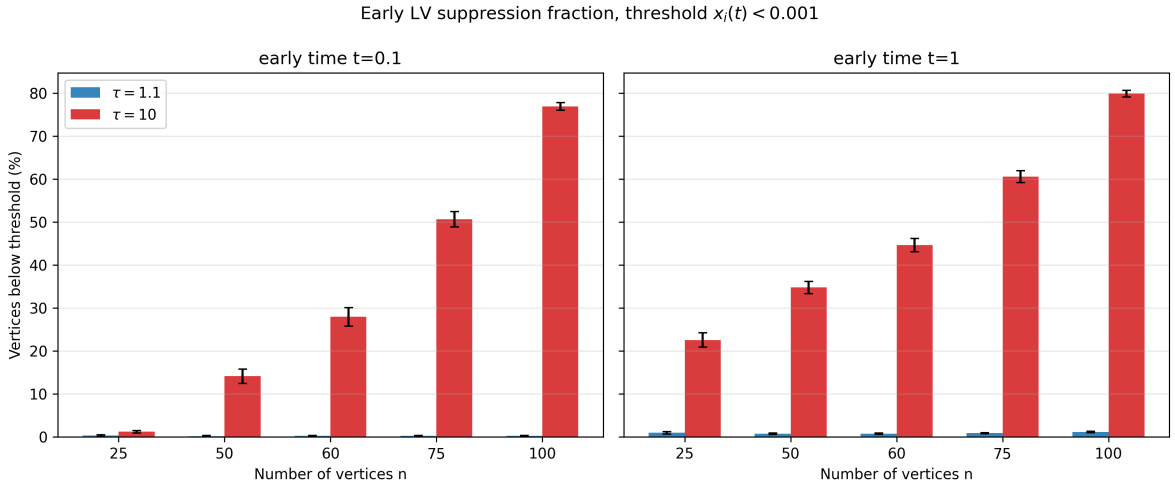


Figure 7.4: Percentage of vertices suppressed below the LV threshold $x_i(t) < 0.001$ at early times $t = 0.1$ and $t = 1.0$ on random geometric graphs with $r = 0.35$, averaged over 50 runs for each n. The initial conditions were sampled from $x_i(0) \in (0.1, 0.9)$, rather than very close to zero, to avoid suppression caused by poor initialization. The grouped bars compare $\tau = 1.1$ and $\tau = 10$, showing that larger τ causes substantially stronger early suppression, especially as n increases.

Figure 7.4 shows that for $\tau = 10$, a much larger fraction of vertices is already below the threshold at early times, and this effect becomes stronger for larger n. This is particularly striking because the initial conditions were chosen from $x_i(0) \in (0.1, 0.9)$, so the suppressed vertices did not start close to the threshold value 0.001. For example, when $n = 100$, almost 80% of vertices are suppressed early on. This suggests that the strong early suppression is caused by the LV dynamics itself, rather than by bad initial conditions. Large τ can therefore create stronger early traps, which the system may later need more time to escape from.

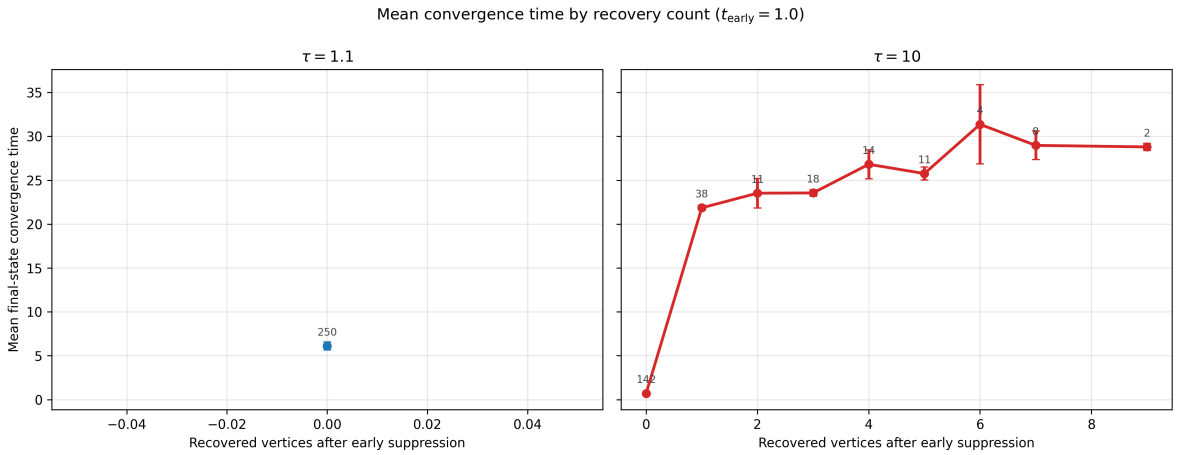


Figure 7.5: Mean LV final-state convergence time as a function of the number of vertices that were suppressed below the threshold $x_i(t) < 0.001$ at $t = 1.0$ but later became selected again. Results are shown for random geometric graphs with $r = 0.35$, averaged over 50 runs for each n, with separate panels for $\tau = 1.1$ and $\tau = 10$. The numbers above the points indicate how many runs contributed to each recovery count.

Figure 7.5 supports the hypothesis by linking recovery after early suppression directly to longer convergence time. For $\tau = 10$, runs with more recovered vertices tend to have larger mean convergence times, meaning that when the dynamics initially suppress vertices that later need to re-enter the selected set, the system takes longer to stabilize. This suggests that large τ can create strong early commitments or traps, and escaping from these traps requires additional reorganization time.

We hypothesize that the longer convergence times observed for larger τ at larger n may be caused by the increasing number of competing maximal independent sets. As n grows, the Lotka–Volterra dynamics may have to resolve competition between more possible stable thresholded outcomes before settling on its final independent set.

To test this explanation more directly, we reuse the fixed-n trap construction as in Figure 6.4. In this construction, n is kept constant while the number of competing trap independent sets is varied. This allows us to isolate the effect of the number of competing maximal independent sets from the effect of graph size. By measuring the LV settling time on these graphs, we can test whether adding more competing maximal sets increases the time needed for the dynamics to stabilize, and whether this helps explain why larger τ can still recover from an initially unfavorable direction.

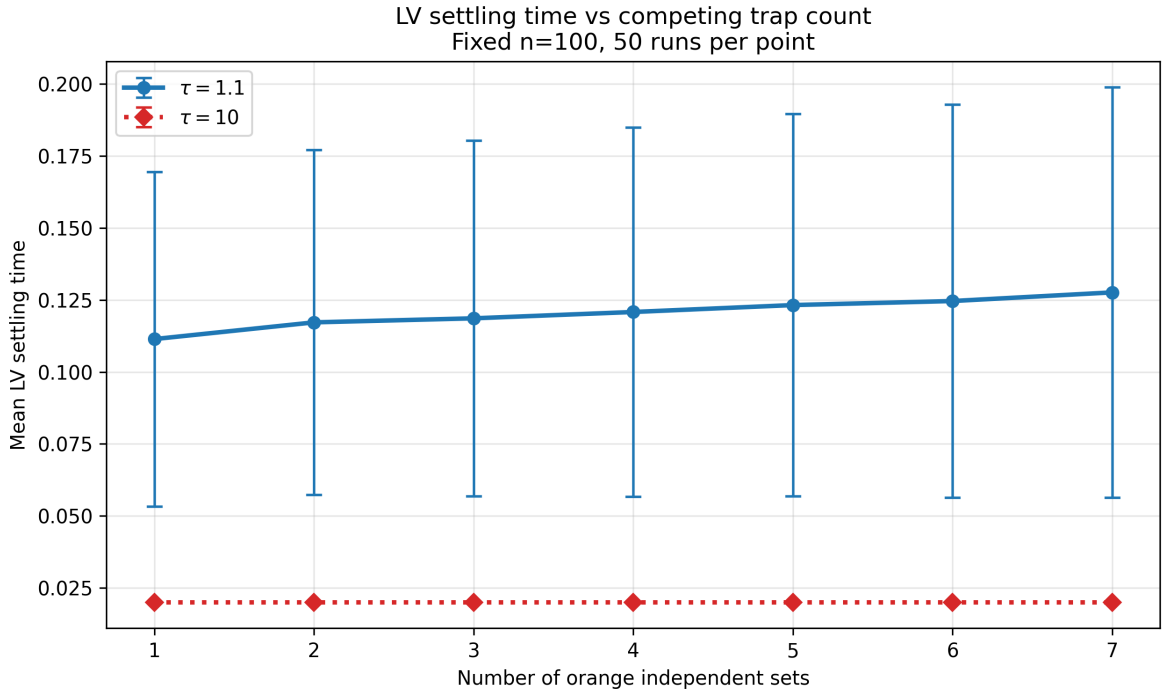


Figure 7.6: Mean Lotka–Volterra settling time on the fixed-n trap construction with $n = 100$, averaged over 50 runs for each number of orange trap independent sets. The green maximum independent set has size 30, while each orange competing trap independent set has size 10. Settling time is measured in LV simulation time units and is defined as the first time after which the thresholded LV-selected independent set remains equal to the final selected set. The plot compares $\tau = 1.1$ and $\tau = 10$, showing that the lower-τ dynamics exhibit a small increase in settling time as more competing trap sets are added, while the higher-τ dynamics stabilize almost immediately for all trap counts.

Figure 7.6 gives only limited support for the hypothesis. For $\tau = 1.1$, the mean settling time increases slightly as the number of competing trap independent sets grows, suggesting that additional competing maximal sets can make the LV dynamics take longer to stabilize. However, this effect is small, and for $\tau = 10$ the settling time remains essentially constant and very low across all trap counts. Therefore, the number of competing maximal independent sets alone does not fully explain the longer convergence times observed for larger τ on larger random geometric graphs.

The results therefore suggest that the number of competing maximal independent sets alone does not fully explain the poor performance at larger τ. Another important factor is the stronger early suppression caused by the LV dynamics itself. Larger τ amplifies the inhibitory effect between neighboring vertices, so on larger graphs many vertices can be pushed close to zero very early in the simulation. These suppressed vertices may then need more time to recover before the final independent set stabilizes. This suggests that the longer convergence times for large τ on larger random geometric graphs may be caused by early over-suppression, rather than by the competing trap structure alone.

7.2. Changing the Cooling Mechanism

In the previous section, we found that simulated annealing does not mainly fail because the measured QUBO barriers become larger. Instead, the failure seems to be related to the increasing number of competing maximal independent sets and possible search directions. We therefore expect that larger graphs need a slower cooling schedule, not necessarily because the barriers are higher, but because the algorithm needs more time to explore a larger and more crowded energy landscape. As n increases, there may be more maximal independent sets, and each binary state vector has more entries. Moving from one relevant configuration to another may therefore require more bit flips, so a longer cooling schedule may be needed.

We now study how the required cooling length changes with the graph size n. Here, the cooling length means the total number of simulated annealing steps, or equivalently the number of proposed one-bit flips during one run of the algorithm. A longer cooling length means that the temperature is

decreased more slowly, giving the algorithm more time to explore the search space.

For each graph family and each value of n, we test several possible cooling lengths. For each cooling length, we compute the mean solution size, meaning the average size of the independent set returned by simulated annealing over the tested graph instances and random runs. We then record the smallest cooling length for which this mean solution size reaches at least 90% of a best-known reference value.

For each graph instance, the best-known solution size is defined as the largest independent set size found among Greedy, Lotka–Volterra with $\tau = 1.1$, and simulated annealing with 500, 1000, 3000, 10000, and 30000 steps. This value is an empirical reference value, not necessarily a proven optimum. Therefore, the 90% threshold should be interpreted as a practical performance criterion rather than as a guarantee of being within 90% of the true maximum independent set.

We expect that the required number of steps grows quickly with n. Since the state space $\{0,1\}^n$ grows exponentially, and since larger graphs tend to contain more competing local minima, our first hypothesis is that the required cooling length may also grow approximately exponentially over the tested range.

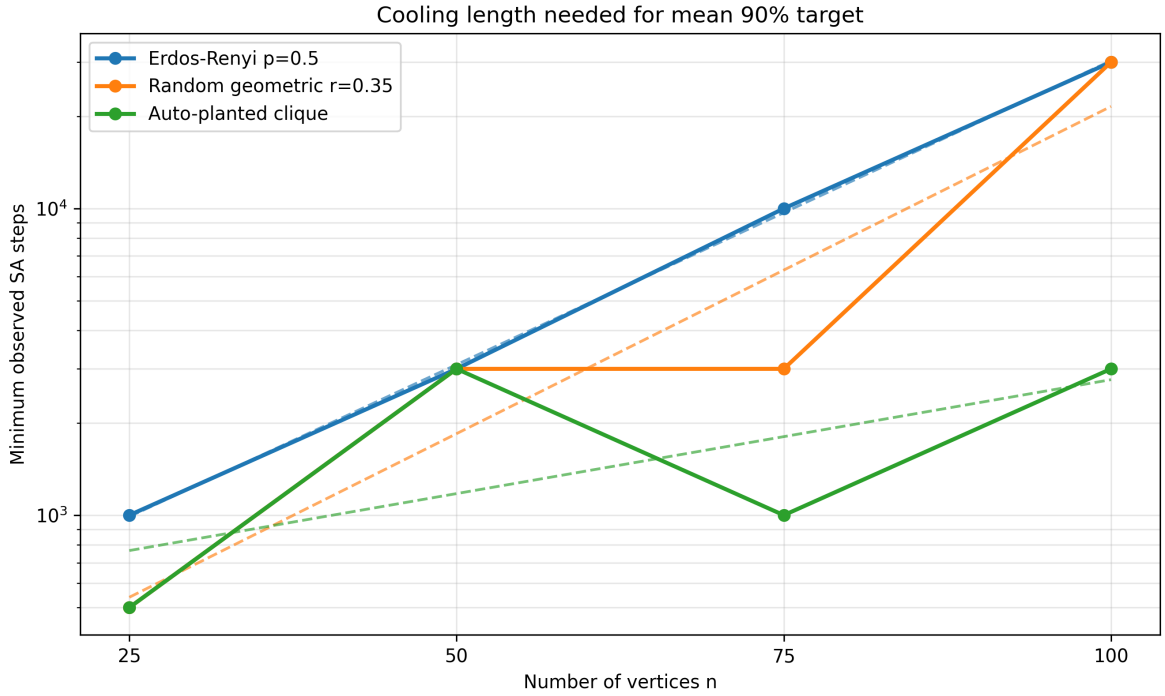


Figure 7.7: Minimum number of simulated annealing steps required for the mean solution size to reach at least 90% of the best-known solution size. Results are shown for Erdős–Rényi, random geometric, and auto-planted clique graphs. Solid lines show the observed required step counts, while dashed lines show exponential fits. The y-axis is shown on a logarithmic scale.

Figure 7.7 shows that the number of required simulated annealing steps increases quickly with n. On the tested range, $n = 25, \ldots, 100$, an exponential trend gives a reasonable fit. However, this should not be interpreted as a proof of exponential scaling. The experiment is limited to moderately large graphs because of runtime constraints, so the result only shows fast growth over the tested range.

The figure also shows that the required cooling length depends on the graph family. Erdős-Rényi and random geometric graphs show a steeper increase than the auto-planted clique graphs. This suggests that the difficulty is not determined by n alone, but also by the structure of the energy landscape.

Additional graph families are tested in the appendix; see Figure E.3.

Overall, these results suggest that a fixed cooling schedule does not scale well with n. Larger graphs require more simulated annealing steps, and the rate at which the required number of steps grows depends strongly on the graph family.

7.3. Changing the Penalty Parameter α

We now investigate what happens when we change the penalty parameter α in the simulated annealing Hamiltonian. Recall from Equation 3.5 that we use $H(x) = -\sum_{i \in V} x_i + \alpha \sum_{\{i,j\} \in E} x_i x_j$, where $x_i \in \{0, 1\}$. Here, α controls how strongly the algorithm is punished for creating edge conflicts.

We compare two values: $\alpha = 1.1$ and $\alpha = 10$. This is similar to the experiment with the parameter τ in the Lotka–Volterra algorithm. We expect that increasing α may decrease the average independent set size found by simulated annealing. The reason is that a larger penalty makes configurations with edge conflicts much more expensive. As a result, simulated annealing is less likely to accept moves that temporarily create or keep conflicts, even if such moves could help the algorithm escape a local trap.

In simulated annealing, uphill moves are accepted with probability

$$P(\text{accept}) = \exp\left(-\frac{\Delta H}{T}\right),$$

as described in Equation 3.6. If α is large, then moves that create additional edge conflicts cause a larger increase in energy ΔH. For example, when a vertex is added to the current set and it is adjacent to already selected vertices, the penalty term in the Hamiltonian increases by α for each new conflict. This makes

$$\exp\left(-\frac{\Delta H}{T}\right)$$

much smaller, so such moves are accepted less often.

This can make it harder for simulated annealing to escape local traps. Sometimes, reaching a better independent set requires temporarily passing through an infeasible state, where two adjacent vertices are selected at the same time. With a small penalty, such moves are still possible during the high-temperature part of the algorithm. With a large penalty, these moves become very unlikely, so the algorithm may become trapped earlier.

The intuition is therefore:

small α	softer penalty, more exploration of infeasible states, easier escape,
large α	harsher penalty, fewer conflict moves accepted, more trapping.

This is especially relevant because simulated annealing already includes a final correction step, described in Section 3.4.9, which ensures that the final output is an independent set. Therefore, a very large penalty is not necessarily useful in our implementation. It may prevent the algorithm from exploring useful intermediate states, while the final correction step would have removed remaining conflicts anyway.

Thus, our hypothesis is:

> *Increasing the penalty parameter α makes simulated annealing less exploratory and may cause it to find smaller independent sets.*

We test this by comparing the performance of simulated annealing for different values of the penalty parameter α, while keeping the other algorithm settings fixed. In this experiment, the number of SA steps is fixed at 1000.

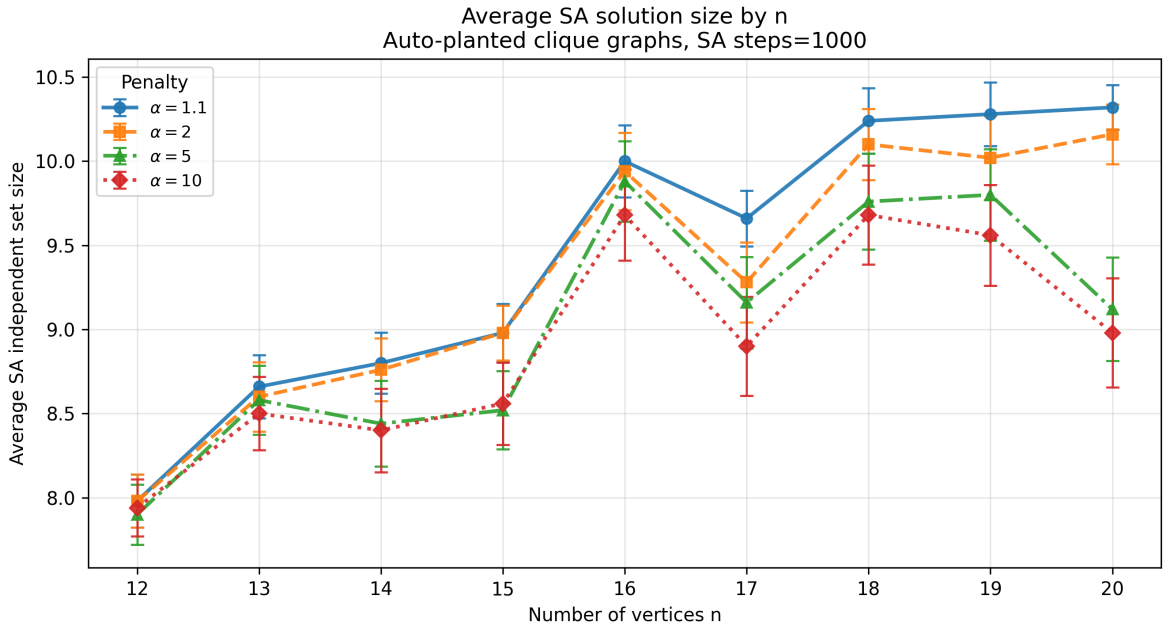


Figure 7.8: Average solution size found by simulated annealing on auto-planted clique graphs for different penalty parameters α, with the number of simulated annealing steps fixed at 1000 for all values of n. Larger values of α penalize edge conflicts more strongly. The dip around $n = 17$ should be interpreted with this fixed-step setting in mind: as n increases, the algorithm has fewer update attempts per variable, so the finite cooling schedule can affect the observed solution size.

Figure 7.8 shows that larger values of α can lead to smaller average solution sizes. This supports the idea that a large penalty parameter makes simulated annealing less willing to explore infeasible intermediate states. When α is large, moves that create additional edge conflicts have a larger positive energy change ΔH, and are therefore less likely to be accepted. As a result, the algorithm may avoid temporary conflict states that could otherwise help it escape a local trap.

The dip around $n = 17$ should be interpreted with some care. In this experiment, the number of simulated annealing steps is fixed at 1000 for all graph sizes. Therefore, as n increases, the number of proposed bit flips per variable decreases. This makes the algorithm more sensitive to finite-time effects and to random variation between graph instances. The dip is therefore likely related to the fixed cooling length and should not be interpreted as a separate structural effect of the graph family.

We also classify the final outputs as maximum, maximal but not maximum, or non-maximal.

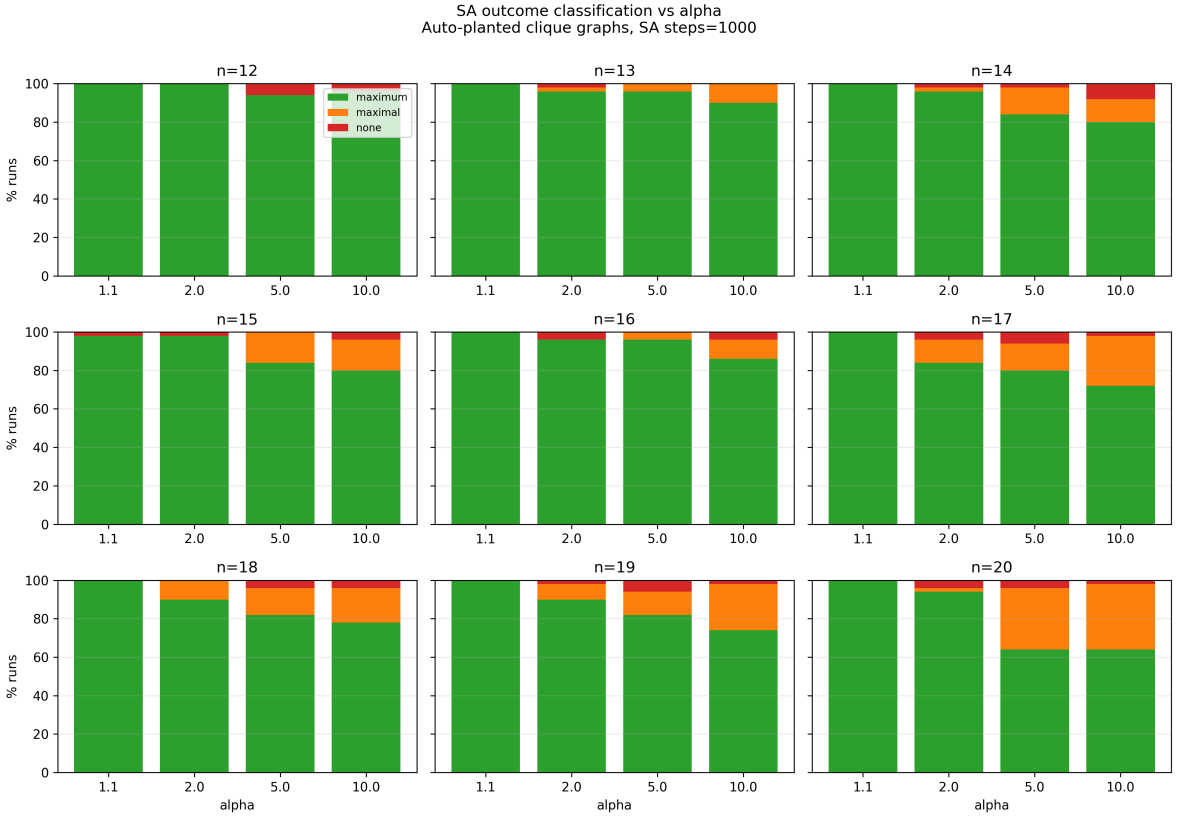


Figure 7.9: Classification of simulated annealing outputs for different penalty parameters α, with 1000 SA steps fixed. The outputs are classified as maximum, maximal but not maximum, or non-maximal.

Figure 7.9 gives more detail about the type of solutions found. In particular, it shows whether a lower average solution size is caused by convergence to smaller maximal independent sets or by outputs that are independent but not yet maximal after the finite number of simulated annealing steps.

An important observation is that for larger α, simulated annealing can sometimes end with a solution that is independent but not maximal. At first this may seem surprising. If the current set is independent and a vertex can be added without creating an edge conflict, then the energy change is

$$\Delta H = -1,$$

so the move lowers the Hamiltonian and would be accepted if it were proposed.

However, simulated annealing only proposes one random bit flip at each step. Therefore, it is not enough that such an improving move exists; the algorithm must also propose this specific move before the cooling process ends. More precisely, the relevant question is not whether every bit has been flipped at some point during the whole run, but whether the missing non-conflicting vertices are proposed after the algorithm has reached, or been repaired to, an independent state. With a finite number of steps, this does not always happen.

To check whether this explanation is plausible, we repeated the experiment with 2000 simulated annealing steps. The corresponding classification plot is shown in Appendix Figure E.4. In this longer run, the non-maximal outcomes disappear for the smaller graph sizes, for example for $n = 12$ and $n = 13$. This supports the interpretation that the non-maximal outputs in Figure 7.9 are at least partly caused by the finite cooling length, rather than by the Hamiltonian favoring non-maximal independent sets. However, this should still be interpreted as empirical evidence rather than a proof for every individual run.

This also explains the difference between small and large penalty values. For large α, moves that create additional edge conflicts have a large positive ΔH, so they are rarely accepted. The algorithm quickly removes conflicts and reaches an independent set, but this independent set may contain relatively few selected vertices. It can then only grow further if non-conflicting vertices are proposed before

the algorithm stops. For smaller α, conflict-creating moves are less costly, so the algorithm can spend more time exploring states with more selected vertices, even if some of these states are temporarily infeasible. This extra exploration can make it easier for the final repaired solution to become a larger independent set.

Overall, these results suggest that a very large penalty parameter is not always beneficial. Because the final repair step already ensures that the output is independent, using a softer penalty can help simulated annealing explore the landscape more effectively before the final solution is selected. Similar results are obtained for random geometric graphs as seen in Figures E.5 and E.6 in the appendix.

To test whether larger values of α make simulated annealing less willing to pass through infeasible states, we track the SA trajectory over time. In particular, we look at the number of edge conflicts, the number of selected vertices, and the first time step at which the current state becomes independent.

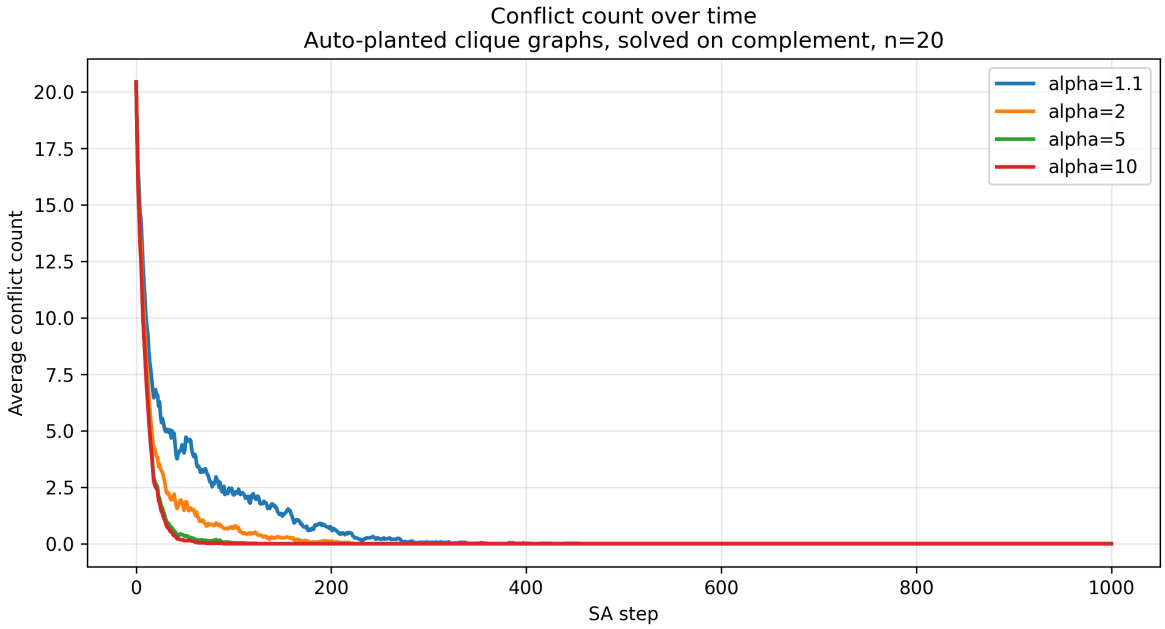


Figure 7.10: Average number of edge conflicts during simulated annealing on planted clique instances, shown for different penalty values α. Larger values of α make conflicts disappear earlier, indicating that the penalty term pushes the algorithm more quickly toward feasible independent sets.

Figure 7.10 supports the first part of the hypothesis. For larger α, the number of conflicts drops much faster. This means that simulated annealing removes infeasible configurations more aggressively and spends less time exploring states with adjacent selected vertices.

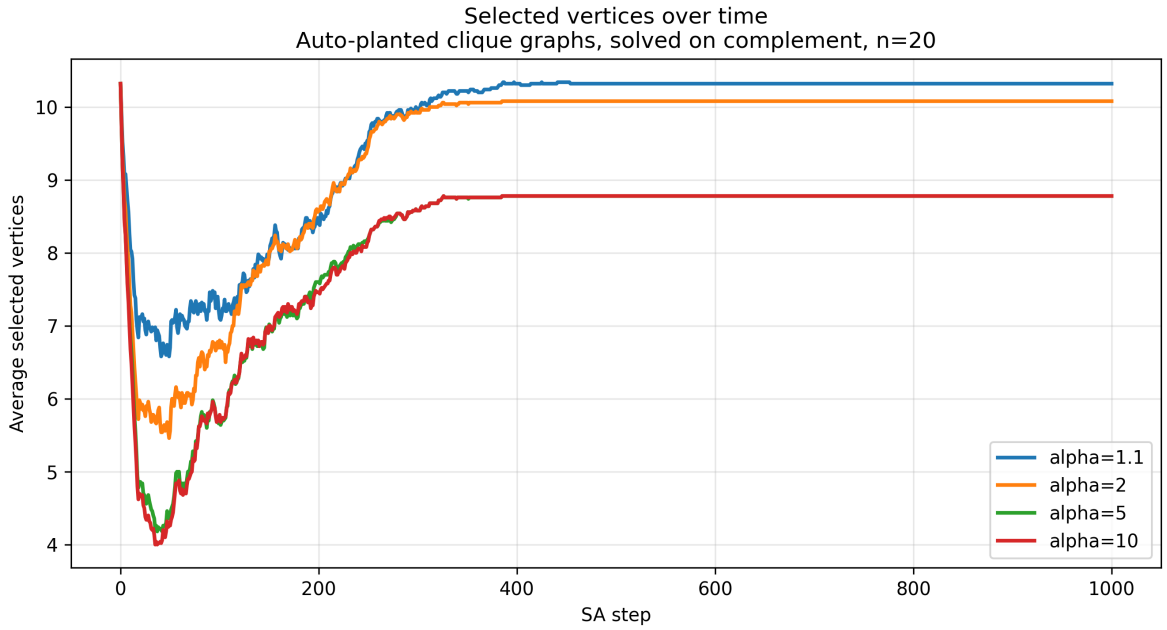


Figure 7.11: Average number of selected vertices during simulated annealing on planted clique instances, shown for different penalty values α. Larger values of α lead to smaller intermediate candidate sets, suggesting more conservative search behavior.

Figure 7.11 shows a similar pattern. Larger values of α keep fewer vertices selected during the run. This suggests that the algorithm removes vertices more readily when conflicts are expensive, which can make the search less exploratory.

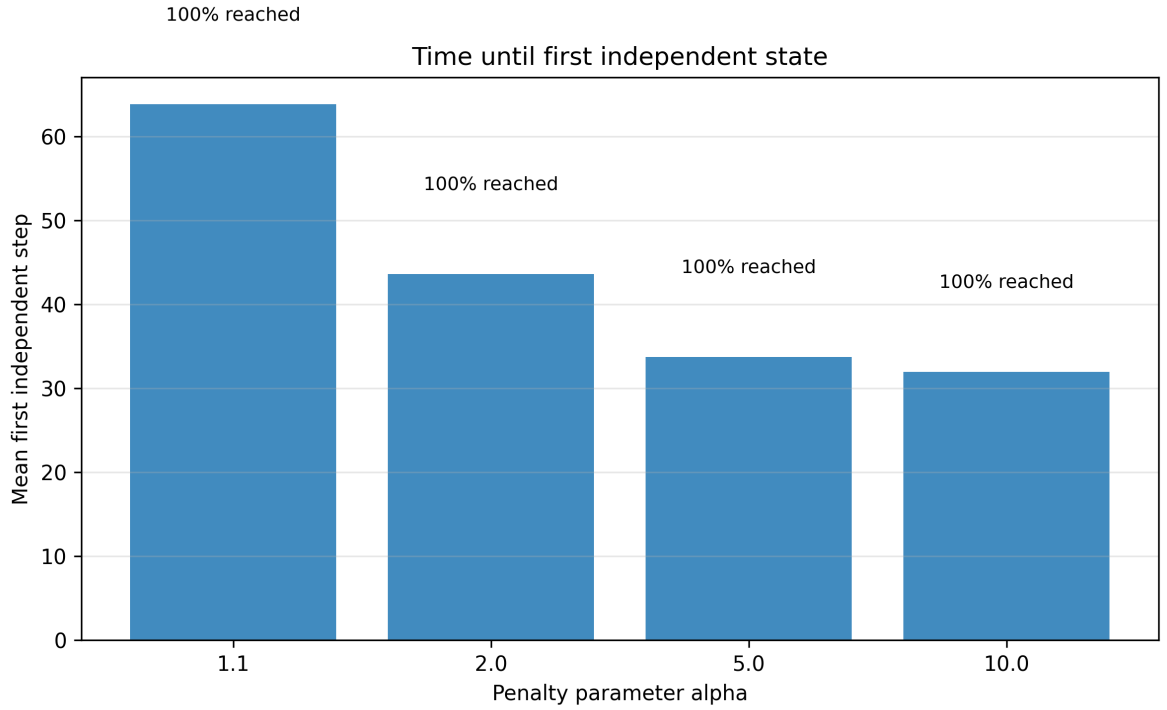


Figure 7.12: First time step at which simulated annealing reaches an independent set on planted clique instances, shown for different penalty values α. Larger values of α drive the system to feasibility earlier.

Figure 7.12 confirms this interpretation. For larger α, simulated annealing reaches an independent

set earlier in the run. Here, a state is called feasible if its selected vertices form an independent set, meaning that no two selected vertices are adjacent. Reaching a feasible state quickly can be useful, but it also means that the algorithm spends less time moving through temporary conflict states that may help it escape local traps.

Similar results were obtained for the random geometric graph experiments, as shown in Appendix Figures E.7, E.8, and E.9.

Overall, these trajectory plots support the hypothesis that larger α values make simulated annealing less willing to explore temporary conflict states. The current state becomes feasible earlier in the run, but this comes at the cost of reduced exploration through states that are not independent.

8

Conclusion

8.1. Main Findings

In this thesis, we studied how graph structure and algorithmic parameter choices affect the performance of the minimum-degree Greedy algorithm, simulated annealing, and Lotka–Volterra dynamics for the maximum-independent-set problem. The main conclusion is that no single method performs best on every graph family. Each method can work well, but each method also has its own failure mechanism.

For the minimum-degree Greedy algorithm, the main difficulty comes from its local selection rule. At every step, Greedy chooses a vertex with minimum degree. This is often a reasonable rule, because low-degree vertices usually block fewer other vertices. However, the experiments show that this rule can also be misleading. In the greedy-bad-chain graph and the trap-clique-independent graph, Greedy selects a vertex that looks good locally, but this choice blocks access to a much larger independent set. Therefore, Greedy performs poorly when a low-degree vertex acts as a trap.

For the Lotka–Volterra algorithm, the failure mechanism is different. The algorithm does not choose vertices one by one. Instead, the vertex values evolve over time, while neighboring vertices suppress each other. This means that the final solution depends on the dynamics and on the initial condition. The experiments show that Lotka–Volterra can settle in a maximal independent set that is not maximum. High clustering can make this worse, because one strong vertex can suppress many nearby vertices. However, clustering alone does not explain all failures. Lotka–Volterra can also perform poorly on graphs with low clustering. The results suggest that the algorithm struggles when there are many competing maximal independent sets, because the dynamics then have many possible final states to settle into.

For simulated annealing, the main difficulty is the energy landscape. The algorithm can escape bad choices by accepting worse moves at high temperature, but this only works if it has enough time before the system cools down. The experiments show that simulated annealing does not fail mainly because the measured energy barriers become much higher. Instead, it struggles when there are many competing local minima or many near-optimal maximal independent sets. In the trap-block construction, adding more maximal independent sets makes simulated annealing more likely to end in a trap, even though the optimum size, trap size, and QUBO barrier stay fixed. This shows that the number of competing traps is itself important.

The simulated annealing parameter experiments support this interpretation. Increasing the cooling length improves performance, which suggests that the algorithm often needs more time to explore a crowded energy landscape. The experiments with the penalty parameter α show a related effect. Increasing α makes edge conflicts more expensive, so the algorithm reaches feasible independent sets earlier. However, this also makes the search less exploratory. With a large α, simulated annealing is less willing to pass through temporary conflict states, even if such states could help it escape a trap. Since the final correction step already ensures that the output is independent, a very large penalty is not always helpful.

The cooling schedule is also important. When the number of simulated annealing steps is fixed, performance becomes worse as the graph size increases. Slower cooling improves the results because it gives the algorithm more time to explore before the temperature becomes too low. The experiments

suggest that larger graphs need more cooling steps, not only because they have more vertices, but also because they contain more competing maximal independent sets and possible search directions.

For Lotka–Volterra, the parameter experiments show a different effect. Increasing τ makes neighboring vertices suppress each other more strongly. This can make the dynamics settle more quickly in some small graphs, but it often leads to smaller independent sets. The experiments therefore suggest that increasing τ does not simply improve the method. Stronger suppression can make the dynamics commit earlier to certain vertices, but this can also make the method more sensitive to bad initial conditions and early over-suppression.

We now return to the research question and subquestions from the introduction. The main research question was:

> **RQ.** How do graph structure and algorithmic parameter choices affect the performance of greedy, simulated annealing, and Lotka–Volterra algorithms for the maximum-independent-set problem?

The answer is that graph structure affects the algorithms by creating different kinds of traps. Greedy is misled by locally attractive low-degree vertices. Lotka–Volterra is pulled toward one of many possible maximal independent sets, depending on the dynamics and the initial condition. Simulated annealing struggles when the energy landscape contains many competing local minima and the cooling schedule is too short. Parameter choices can make these effects better or worse. Stronger suppression in Lotka–Volterra and stronger penalties in simulated annealing can make the algorithms commit earlier, but they can also reduce exploration and lower the quality of the final solution.

For the first subquestion, we asked which graph families cause each algorithm to perform significantly worse than the others, and how this behavior can be explained from the structure of the graph. The results show that the minimum-degree Greedy algorithm performs poorly on the greedy-bad-chain and trap-clique-independent graphs, because these graphs contain low-degree vertices that are locally attractive but globally harmful. Lotka–Volterra performs poorly on several graph families, including random geometric graphs, dense Erdős–Rényi graphs, and some low-clustering examples, suggesting that clustering can contribute to poor performance but is not the only explanation. Simulated annealing performs poorly when the graph creates many competing local minima or trap states, especially when the cooling schedule is not long enough for the algorithm to explore the landscape.

For the second subquestion, we asked how parameter choices affect the algorithms, especially on graph families where they perform poorly. For simulated annealing, longer cooling schedules improve performance because they give the algorithm more time to escape local minima. The penalty parameter α controls the trade-off between enforcing independence and allowing exploration through temporary conflict states. For Lotka–Volterra, the parameter τ controls the strength of suppression between neighboring vertices. Larger values of τ enforce stronger competition, but they can also make the dynamics commit too early and return smaller independent sets.

Thus, the performance of these algorithms cannot be understood from graph size alone. It depends on the structure of the graph, the number of competing maximal independent sets, the initial condition, and the parameter choices. This is why there is no universally best heuristic among the three methods studied here. The best choice depends on the graph family and on how much computation time is available.

8.2. Future Work

The results of this thesis suggest several possible directions for future work. Since each algorithm fails in a different way, future improvements should also be adapted to the specific failure mechanism of each method.

For Greedy, the main problem is that one early local choice can block a much better solution. In the graph families studied in this thesis, this failure is not mainly caused by unlucky tie-breaking, but by the minimum-degree rule itself. Therefore, simply changing the tie-breaking rule would not be enough to solve the structural problem. A more relevant direction for future work would be to add a small look-ahead step, where the algorithm checks not only the degree of a vertex, but also how many useful vertices would be removed after selecting it. Another possibility would be to combine Greedy with a local improvement step afterwards, so that the algorithm can try to repair poor early choices.

For Lotka–Volterra, the results suggest that the initial condition matters. One possible direction is therefore to run the dynamics several times with different initial conditions and keep the largest independent set found. This does not prevent a single run from settling into a poor maximal independent set, but it can compensate for this weakness by giving the algorithm several chances to find a better solution. Another direction is to tune τ more carefully, since larger values of τ can make the dynamics too aggressive and lead to smaller solutions. In addition, the numerical integration method could be improved. In this thesis, the dynamics are simulated with the forward Euler method, which is simple but not always the most accurate or stable choice. Using a more advanced ODE solver could give a more reliable approximation of the continuous Lotka–Volterra dynamics, especially when the dynamics change quickly or when larger values of τ are used.

For simulated annealing, the most important direction for future work is the cooling schedule. The experiments show that larger graphs need more cooling steps. Therefore, the number of steps should not be fixed, but should grow with the graph size and possibly also depend on the graph family. Another direction would be to use multiple simulated annealing runs or a reheating strategy, so that the algorithm gets more chances to escape from trap states.

Finally, the algorithms could also be combined with simple post-processing steps. For example, simulated annealing or Lotka–Volterra could first be used to find an initial independent set. Afterwards, a local improvement step could try to improve this set by adding vertices that do not create conflicts, or by replacing one selected vertex with several non-selected vertices when this increases the solution size. This local improvement step is not the same as the Greedy algorithm itself; it is a post-processing step applied after another algorithm has already produced a candidate solution.

Overall, the most promising future work would use the observed failure mechanisms directly. Greedy needs protection against irreversible early choices made by the minimum-degree rule. Lotka–Volterra needs protection against convergence to suboptimal stable states caused by the dynamics and the initial condition. Simulated annealing needs enough time and flexibility to explore a crowded energy landscape before the cooling schedule becomes too restrictive.

Bibliography

[1] Guy E. Blelloch, Jeremy T. Fineman, and Julian Shun. Greedy sequential maximal independent set and matching are parallel on average. *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 308–317, 2012. doi: 10.1145/2312005.2312058.

[2] Béla Bollobás. *Random Graphs*. Cambridge University Press, 2 edition, 2001.

[3] Béla Bollobás and Paul Erdős. Cliques in random graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 80(3):419–427, 1976. doi: 10.1017/S0305004100053056.

[4] Immanuel M. Bomze, Marco Budinich, Panos M. Pardalos, and Marcello Pelillo. The maximum clique problem. In Ding-Zhu Du and Panos M. Pardalos, editors, *Handbook of Combinatorial Optimization*, pages 1–74. Kluwer Academic Publishers, Boston, MA, 1999.

[5] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. Macmillan, 1976.

[6] Steven Chaplick, Martin Frohn, Steven Kelk, Johann Lottermoser, and Matúš Mihalák. Approximation ratio of the min-degree greedy algorithm for maximum independent set on interval and chordal graphs. *arXiv preprint arXiv:2403.10868*, 2024.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 3 edition, 2009.

[8] Gopal Ramesh Dahale. Maximum independent set with quantum adiabatic algorithm. `https://gopal-dahale.github.io/ILearnQuantum/posts/mis/qaa/`, 2023. Accessed: 2026-05-29.

[9] Jesper Dall and Michael Christensen. Random geometric graphs. *Physical Review E*, 66(1): 016121, 2002. doi: 10.1103/PhysRevE.66.016121.

[10] Reinhard Diestel. *Graph Theory*. Springer, 5 edition, 2017. doi: 10.1007/978-3-662-53622-3.

[11] David Eppstein. Gabriel graph.svg. Wikimedia Commons, 2010. URL `https://commons.wikimedia.org/wiki/File:Gabriel_graph.svg`. Public domain. Accessed: 13 July 2026.

[12] Paul Erdős and Alfréd Rényi. On random graphs. i. *Publicationes Mathematicae*, 6:290–297, 1959.

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

[14] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6): 721–741, 1984. doi: 10.1109/TPAMI.1984.4767596.

[15] Ernst Hairer, Syvert P. Nørsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer, 1993.

[16] Bruce Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13 (2):311–329, 1988. doi: 10.1287/moor.13.2.311.

[17] Magnús M. Halldórsson and Jaikumar Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica*, 18(1):145–163, 1997.

[18] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[19] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi: 10.1126/science.220.4598.671.

[20] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Pearson, 2006.

[21] Alfred J. Lotka. Analytical note on certain rhythmic relations in organic systems. *Proceedings of the National Academy of Sciences*, 6(7):410–415, 1920.

[22] Andrew Lucas. Ising formulations of many np problems. *Frontiers in Physics*, 2:5, 2014. doi: 10.3389/fphy.2014.00005.

[23] Walid Mahdi, Sidi Mohammed Medjahed, and Mohammed Ouali. Performance analysis of simulated annealing cooling schedules in the context of dense image matching. *Computación y Sistemas*, 21(3):493–501, 2017.

[24] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 1953.

[25] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953. doi: 10.1063/1.1699114.

[26] M. N. Mooij. Generating maximal independent sets using lotka-volterra dynamics. Master's thesis, Utrecht University, 2022.

[27] Niek M. Mooij and Ivan Kryven. Finding large independent sets in networks using competitive dynamics. *Communications Physics*, 8(1):305, 2025. doi: 10.1038/s42005-025-02153-7.

[28] Nini. Introduction, implementation and comparison of four randomized optimization algorithms, March 2019. URL https://medium.com/@duoduoyunnini/introduction-implementation-and-comparison-of-four-randomized-optimization-algorithms- Figure 2, accessed 14 July 2026.

[29] Mathew Penrose. *Random Geometric Graphs*. Oxford University Press, 2003.

[30] Pixabay. Namibia wildlife at a waterhole. https://cdn.pixabay.com/photo/2020/03/13/19/52/namibia-4928984_1280.jpg, 2020. Accessed: 2026-05-27.

[31] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 3 edition, 2012.

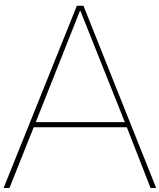
[32] P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, 1987.

[33] Peter J. M. van Laarhoven and Emile H. L. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, 1987.

[34] Vito Volterra. Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. *Memorie della Reale Accademia Nazionale dei Lincei*, 2:31–113, 1926.

[35] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2 edition, 2001.

[36] Nicholas C. Wormald. Models of random regular graphs. In J. D. Lamb and D. A. Preece, editors, *Surveys in Combinatorics, 1999*, volume 267 of *London Mathematical Society Lecture Note Series*, pages 239–298. Cambridge University Press, 1999.



Declaration of AI Usage

During this project, AI tools were used as supportive tools, but not as a replacement for my own scientific work. I used ChatGPT mainly for writing and editing support, for example to improve phrasing, check grammar, and make the text flow more clearly. AI was also used for brainstorming how to present certain ideas in parts of the proofs in the following section, such as the correspondence between maximal independent sets and local minima, and the correspondence between maximum independent sets and ground states. However, the mathematical content, implementation, experiments, and interpretation of the results are my own.

For coding, Codex was used as a source of support and inspiration. This included help with debugging, code structure, and improving or adapting code for plots and visualizations. The main implementation choices and experimental setup were made by me.

For writing, AI was used more extensively as a language and presentation aid. This included grammar correction, rephrasing, improving clarity, and assistance with LaTeX formatting. While AI contributed to the formulation of sentences, the underlying arguments, scientific reasoning, and conclusions remain my own.

All AI-generated suggestions were critically checked before being used. The AI tools used during this project were GPT-5.5 for writing support and Codex 5.5 for coding-related assistance.

B

Proof of the correspondence between maximal independent sets and local minima

This proof rewrites the stability interpretation from [27] in terms of the Lyapunov function used in this thesis.

A maximal independent set induces a local minimum of the Lyapunov function.

Proof. Assume that $S \subseteq V$ is a maximal independent set. We want to show that χ_S is a local minimum of the Lyapunov function

$$V(x) = \frac{1}{2}x^T Mx - \mathbf{1}^T x$$

on the domain

$$D = \mathbb{R}^n_{\geq 0}.$$

To prove that χ_S is a local minimum on D, it is enough to show that for every sufficiently small nonzero feasible perturbation h, we have

$$V(\chi_S + h) > V(\chi_S).$$

Here, feasible means that

$$\chi_S + h \in D.$$

Here,

$$M = I + \tau A,$$

where A is the adjacency matrix of the graph. We assume that $\tau > 1$.

The gradient of V is

$$\nabla V(x) = Mx - \mathbf{1}.$$

At $x = \chi_S$, the i-th component is

$$(\nabla V(\chi_S))_i = (\chi_S)_i + \tau \sum_{j \sim i} (\chi_S)_j - 1.$$

First, consider a vertex $i \in S$. Then

$$(\chi_S)_i = 1.$$

Since S is independent, no neighbor of i is also contained in S. Therefore,

$$\sum_{j \sim i} (\chi_S)_j = 0.$$

Hence,

$$(\nabla V(\chi_S))_i = 1 + \tau \cdot 0 - 1 = 0.$$

So the gradient component $(\nabla V(\chi_S))_i$ is zero for every vertex $i \in S$.

Now consider a vertex $i \notin S$. Then

$$(\chi_S)_i = 0.$$

Since S is maximal, every vertex outside S has at least one neighbor inside S. Otherwise, that vertex could be added to S, contradicting maximality. Therefore,

$$\sum_{j \sim i} (\chi_S)_j \geq 1.$$

It follows that

$$(\nabla V(\chi_S))_i = \tau \sum_{j \sim i} (\chi_S)_j - 1 \geq \tau - 1 > 0.$$

Thus, the gradient is strictly positive in every coordinate outside S.

Now let $h \in \mathbb{R}^n$ be a sufficiently small feasible perturbation of χ_S, meaning that

$$\chi_S + h \in \mathbb{R}^n_{\geq 0}.$$

For every $i \notin S$, we have $(\chi_S)_i = 0$. Therefore feasibility implies

$$h_i \geq 0 \qquad \text{for all } i \notin S.$$

We now compare the value of V at $\chi_S + h$ and at χ_S. By the definition of V,

$$V(\chi_S + h) = \frac{1}{2}(\chi_S + h)^T M(\chi_S + h) - \mathbf{1}^T(\chi_S + h).$$

Expanding this gives

$$V(\chi_S + h) = \frac{1}{2}\chi_S^T M \chi_S + h^T M \chi_S + \frac{1}{2}h^T M h - \mathbf{1}^T \chi_S - \mathbf{1}^T h,$$

where we used that M is symmetric. Since

$$V(\chi_S) = \frac{1}{2}\chi_S^T M \chi_S - \mathbf{1}^T \chi_S,$$

subtracting gives

$$V(\chi_S + h) - V(\chi_S) = h^T M \chi_S - \mathbf{1}^T h + \frac{1}{2}h^T M h.$$

Using

$$\nabla V(\chi_S) = M\chi_S - \mathbf{1},$$

we obtain

$$V(\chi_S + h) - V(\chi_S) = h^T \nabla V(\chi_S) + \frac{1}{2}h^T M h.$$

We first analyze the linear term. Since the gradient is zero on S, only the coordinates outside S contribute to

$$h^T \nabla V(\chi_S).$$

Outside S, the gradient is strictly positive, and every feasible perturbation satisfies $h_i \geq 0$. Hence,

$$h^T \nabla V(\chi_S) \geq 0.$$

More precisely, define

$$c = \min_{i \notin S} (\nabla V(\chi_S))_i.$$

By the previous calculation,

$$c \geq \tau - 1 > 0.$$

We now split the perturbation h into two parts:

$$h = u + w,$$

where u is supported on S, and w is supported on $V \setminus S$. In other words,

$$u_i = \begin{cases} h_i, & i \in S, \\ 0, & i \notin S, \end{cases} \qquad w_i = \begin{cases} 0, & i \in S, \\ h_i, & i \notin S. \end{cases}$$

Since w contains exactly the outside coordinates of the perturbation, we have

$$w_i \geq 0 \qquad \text{for all } i \notin S.$$

Because the gradient is zero on S, we get

$$h^T \nabla V(\chi_S) = w^T \nabla V(\chi_S).$$

Since each outside component of the gradient is at least $c > 0$, and since $w_i \geq 0$ for all $i \notin S$, we obtain

$$h^T \nabla V(\chi_S) = w^T \nabla V(\chi_S) = \sum_{i \notin S} w_i (\nabla V(\chi_S))_i \geq c \sum_{i \notin S} w_i.$$

Because all terms w_i in this sum are nonnegative, we have

$$\sum_{i \notin S} w_i \geq \left(\sum_{i \notin S} w_i^2 \right)^{1/2} = \|w\|.$$

Therefore,

$$h^T \nabla V(\chi_S) \geq c\|w\|,$$

where $\| \cdot \|$ denotes the Euclidean norm.

It remains to control the quadratic term. We cannot simply assume that this term is nonnegative. A symmetric matrix is positive semidefinite if

$$z^T M z \geq 0$$

for every vector z. However, the matrix

$$M = I + \tau A$$

does not necessarily have this property when $\tau > 1$. For example, if i and j are adjacent vertices and we take the direction $d = e_i - e_j$, then

$$d^T M d = 2 - 2\tau = 2(1 - \tau) < 0.$$

Thus, for graphs with at least one edge, M is not positive semidefinite when $\tau > 1$. Therefore, it is not true in general that

$$h^T M h \geq 0.$$

We now use the decomposition $h = u + w$ to control the quadratic term. The reason for this split is that perturbations on S and perturbations outside S behave differently. On the coordinates in S, the gradient is zero, but the quadratic term is well behaved because S is independent. On the coordinates outside S, the quadratic term is not necessarily positive by itself, but the linear term gives a positive contribution because the gradient is strictly positive there and feasibility implies $w_i \geq 0$. Thus, after splitting h, the only term that still needs to be controlled is the interaction between the two parts. We have

$$h^T M h = u^T M u + 2u^T M w + w^T M w.$$

We consider these terms separately. First, since S is independent, there are no edges between vertices in S. Therefore, on the coordinates in S, the adjacency part A gives no contribution. Hence,

$$u^T M u = u^T (I + \tau A) u = \|u\|^2.$$

Second, since all entries of M are nonnegative and $w_i \geq 0$ for all $i \notin S$, we have

$$w^T M w \geq 0.$$

The only term that may be negative is the mixed term

$$2u^T M w.$$

Therefore, the rest of the proof is devoted to showing that this mixed term cannot dominate the positive contributions when the perturbation h is sufficiently small.

Since the graph has n vertices, the matrix M has only finitely many entries. Let

$$m_{\max} = \max_{1 \leq i,j \leq n} |M_{ij}|.$$

Then

$$|u^T M w| = \left| \sum_{i=1}^{n} \sum_{j=1}^{n} u_i M_{ij} w_j \right| \leq m_{\max} \sum_{i=1}^{n} \sum_{j=1}^{n} |u_i||w_j|.$$

This can be written as

$$|u^T M w| \leq m_{\max} \left(\sum_{i=1}^{n} |u_i| \right) \left(\sum_{j=1}^{n} |w_j| \right).$$

Since

$$\sum_{i=1}^{n} |u_i| \leq \sqrt{n}\|u\| \qquad \text{and} \qquad \sum_{j=1}^{n} |w_j| \leq \sqrt{n}\|w\|,$$

we obtain

$$|u^T M w| \leq n m_{\max} \|u\| \|w\|.$$

Thus, by setting

$$K = n m_{\max},$$

we get

$$|u^T M w| \leq K \|u\| \|w\|.$$

Therefore,

$$\frac{1}{2} h^T M h \geq \frac{1}{2} \|u\|^2 - K \|u\| \|w\|.$$

Since we want a lower bound and

$$w^T M w \geq 0,$$

we may leave this positive term out.

Combining this with the lower bound for the linear term gives

$$V(\chi_S + h) - V(\chi_S) \geq c\|w\| + \frac{1}{2} \|u\|^2 - K \|u\| \|w\|.$$

We now show that the negative mixed term cannot dominate for sufficiently small perturbations. Using the inequality

$$\left(\frac{1}{2} \|u\| - K \|w\| \right)^2 \geq 0,$$

we get

$$K \|u\| \|w\| \leq \frac{1}{4} \|u\|^2 + K^2 \|w\|^2.$$

Therefore,

$$V(\chi_S + h) - V(\chi_S) \geq c\|w\| + \frac{1}{4} \|u\|^2 - K^2 \|w\|^2.$$

Now choose the perturbation h sufficiently small so that

$$\|h\| \leq \frac{c}{2K^2}.$$

Since w is one part of the decomposition $h = u + w$, we have

$$\|w\| \leq \|h\|.$$

Therefore,

$$\|w\| \leq \frac{c}{2K^2}.$$

It follows that

$$K^2\|w\|^2 \leq \frac{c}{2}\|w\|.$$

Hence,

$$c\|w\| - K^2\|w\|^2 \geq c\|w\| - \frac{c}{2}\|w\| = \frac{c}{2}\|w\|.$$

Therefore,

$$V(\chi_S + h) - V(\chi_S) \geq \frac{c}{2}\|w\| + \frac{1}{4}\|u\|^2 \geq 0.$$

Moreover, if $h \neq 0$, then either $u \neq 0$ or $w \neq 0$. Hence,

$$V(\chi_S + h) > V(\chi_S)$$

for every sufficiently small nonzero feasible perturbation h.

Therefore, χ_S is a local minimum of V on $\mathbb{R}^n_{\geq 0}$.

□

A local minimum of the Lyapunov function induces a maximal independent set.

Proof. Now assume that $x^* \in \mathbb{R}^n_{\geq 0}$ is a local minimum of V. We show that $x^* = \chi_S$ for some maximal independent set S.

Define the support of x^* by

$$S = \{i \in V : x_i^* > 0\}.$$

We first show that S is independent. Suppose, for contradiction, that S is not independent. Then there exist adjacent vertices $i, j \in S$. Since $x_i^* > 0$ and $x_j^* > 0$, both coordinates are interior coordinates of the domain. Therefore, small perturbations in these coordinates remain feasible. At a local minimum, the second-order necessary condition says that the Hessian must be positive semidefinite along all such feasible directions. In other words, for every sufficiently small direction y supported only on these interior coordinates, we must have

$$y^T\nabla^2 V(x^*)y \geq 0.$$

The Hessian of V is constant and equal to

$$\nabla^2 V(x) = M = I + \tau A.$$

Let e_i denote the i-th standard basis vector in $\mathbb{R}^n$, meaning that its i-th component is equal to 1 and all other components are equal to 0. We consider the direction

$$d = e_i - e_j.$$

Then we have

$$d^T M d = (e_i - e_j)^T(I + \tau A)(e_i - e_j).$$

The identity part gives

$$(e_i - e_j)^T I(e_i - e_j) = 2.$$

The adjacency part gives

$$(e_i - e_j)^T A(e_i - e_j) = -2,$$

because i and j are adjacent. Hence

$$d^T M d = 2 - 2\tau = 2(1-\tau) < 0,$$

since $\tau > 1$. This contradicts the positive semidefiniteness required by the second-order condition for a local minimum. Therefore, S must be independent.

Next, for every $i \in S$, we have $x_i^* > 0$. Hence i is an interior coordinate of the domain, so the first-order condition gives

$$(\nabla V(x^*))_i = 0.$$

Thus

$$0 = x_i^* + \tau \sum_{j \sim i} x_j^* - 1 \qquad \text{for all } i \in S.$$

Since S is independent, every neighbor $j \sim i$ lies outside S, so $x_j^* = 0$. Therefore

$$0 = x_i^* - 1,$$

and hence

$$x_i^* = 1$$

for every $i \in S$. Since $x_i^* = 0$ for $i \notin S$, we obtain

$$x^* = \chi_S.$$

It remains to prove that S is maximal. Suppose, for contradiction, that S is not maximal. Then there exists a vertex $k \notin S$ with no neighbors in S. Since $k \notin S$, we have $(\chi_S)_k = 0$. Consider the feasible perturbation

$$\chi_S + \varepsilon e_k$$

for small $\varepsilon > 0$. The directional derivative in the e_k-direction is

$$(\nabla V(\chi_S))_k = 0 + \tau \sum_{j \sim k} (\chi_S)_j - 1.$$

Because k has no neighbors in S,

$$\sum_{j \sim k} (\chi_S)_j = 0.$$

Therefore

$$(\nabla V(\chi_S))_k = -1 < 0.$$

For this perturbation, from the calculation above, with $h = \varepsilon e_k$, we have

$$V(\chi_S + \varepsilon e_k) - V(\chi_S) = \varepsilon (\nabla V(\chi_S))_k + \frac{1}{2}\varepsilon^2 e_k^T M e_k.$$

We already showed that

$$(\nabla V(\chi_S))_k = -1.$$

Moreover, since $M = I + \tau A$ and the adjacency matrix has zero diagonal, we have

$$e_k^T M e_k = M_{kk} = 1.$$

Therefore,

$$V(\chi_S + \varepsilon e_k) - V(\chi_S) = -\varepsilon + \frac{1}{2}\varepsilon^2.$$

For every $0 < \varepsilon < 2$, this expression is negative. Hence, for sufficiently small $\varepsilon > 0$,

$$V(\chi_S + \varepsilon e_k) < V(\chi_S),$$

which contradicts the fact that χ_S is a local minimum. □

We conclude that

$$S \text{ is a maximal independent set} \quad \Leftrightarrow \quad \chi_S \text{ is a local minimum of } V \text{ on } \mathbb{R}^n_{\geq 0}.$$

C

Proof of the correspondence between maximum independent sets and ground states

This proof is based on the energy-landscape interpretation of the Lotka–Volterra formulation. It complements the correspondence between maximal independent sets and local minima discussed in Section 3.3.6.

We prove that

$$S \text{ is a maximum independent set} \quad \Leftrightarrow \quad \chi_S \text{ is a ground state of } V \text{ on } \mathbb{R}^n_{\geq 0}.$$

Here, a ground state means a global minimizer of V on the domain

$$D = \mathbb{R}^n_{\geq 0}.$$

First, observe what V equals on the characteristic vector of an independent set. Let S be an independent set. Then

$$V(\chi_S) = \frac{1}{2}\chi_S^T M \chi_S - \mathbf{1}^T \chi_S.$$

Since

$$M = I + \tau A,$$

we get

$$\chi_S^T M \chi_S = \chi_S^T I \chi_S + \tau \chi_S^T A \chi_S.$$

Now

$$\chi_S^T I \chi_S = |S|.$$

Moreover, since S is independent, there are no edges between vertices of S, so

$$\chi_S^T A \chi_S = 0.$$

Therefore

$$\chi_S^T M \chi_S = |S|.$$

Also,

$$\mathbf{1}^T \chi_S = |S|.$$

Hence

$$V(\chi_S) = \frac{1}{2}|S| - |S| = -\frac{1}{2}|S|.$$

Thus, for every independent set S,

$$V(\chi_S) = -\frac{1}{2}|S|.$$

So larger independent sets have lower energy.

A maximum independent set induces a ground state of the Lyapunov function.

Proof. Assume that S is a maximum independent set. Let

$$|S|$$

denote the independence number of G. Therefore

$$V(\chi_S) = -\frac{1}{2}|S|.$$

We show that no point in $\mathbb{R}^n_{\geq 0}$ has smaller energy. Let x^* be a global minimizer of V on $\mathbb{R}^n_{\geq 0}$. Such a minimizer exists because V is continuous and the domain $\mathbb{R}^n_{\geq 0}$ is closed. Moreover, $V(x) \to +\infty$ as $\|x\| \to \infty$ in this domain: the quadratic term in V grows like $\|x\|^2$, while the linear term grows only like $\|x\|$. Hence the minimum cannot occur at infinity, and V attains its minimum on $\mathbb{R}^n_{\geq 0}$.

Every global minimizer is also a local minimizer. By the result proved above, $x^* = \chi_T$ for some maximal independent set T. Hence

$$V(x^*) = V(\chi_T) = -\frac{1}{2}|T|.$$

Since T is an independent set,

$$|T| \leq |S|.$$

Therefore

$$V(x^*) = -\frac{1}{2}|T| \geq -\frac{1}{2}|S| = V(\chi_S).$$

Thus no point has energy below $V(\chi_S)$, so χ_S is a global minimizer of V. Hence χ_S is a ground state. □

A ground state of the Lyapunov function induces a maximum independent set.

Proof. Assume that χ_S is a ground state of V on $\mathbb{R}^n_{\geq 0}$. Since every ground state is a local minimum, the first correspondence implies that S is a maximal independent set. In particular, S is independent.

Suppose, for contradiction, that S is not maximum. Then there exists an independent set $T \subseteq V$ such that

$$|T| > |S|.$$

Using the energy formula for independent sets, we have

$$V(\chi_S) = -\frac{1}{2}|S|$$

and

$$V(\chi_T) = -\frac{1}{2}|T|.$$

Since $|T| > |S|$, it follows that

$$-\frac{1}{2}|T| < -\frac{1}{2}|S|.$$

Therefore

$$V(\chi_T) < V(\chi_S),$$

which contradicts the assumption that χ_S is a ground state. Hence no larger independent set exists, and S is a maximum independent set. □

We conclude that

$$S \text{ is a maximum independent set} \quad \Leftrightarrow \quad \chi_S \text{ is a ground state of } V \text{ on } \mathbb{R}^n_{\geq 0}.$$

D

Additional Algorithm Results

D.1. Greedy Additional Results

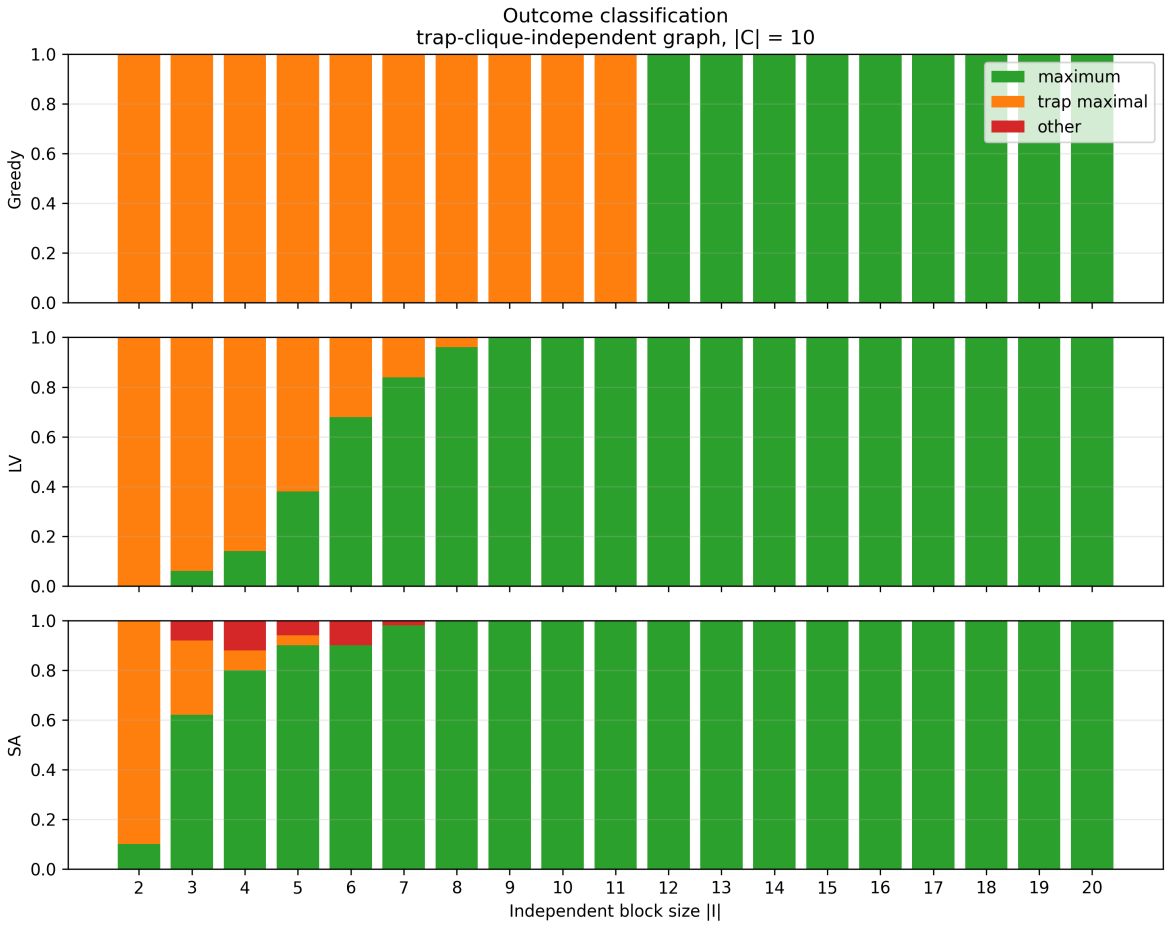


Figure D.1: Outcome classification for Greedy, Lotka–Volterra, and simulated annealing on the trap-clique-independent graph. Outcomes are classified as maximum, trap maximal, or other.

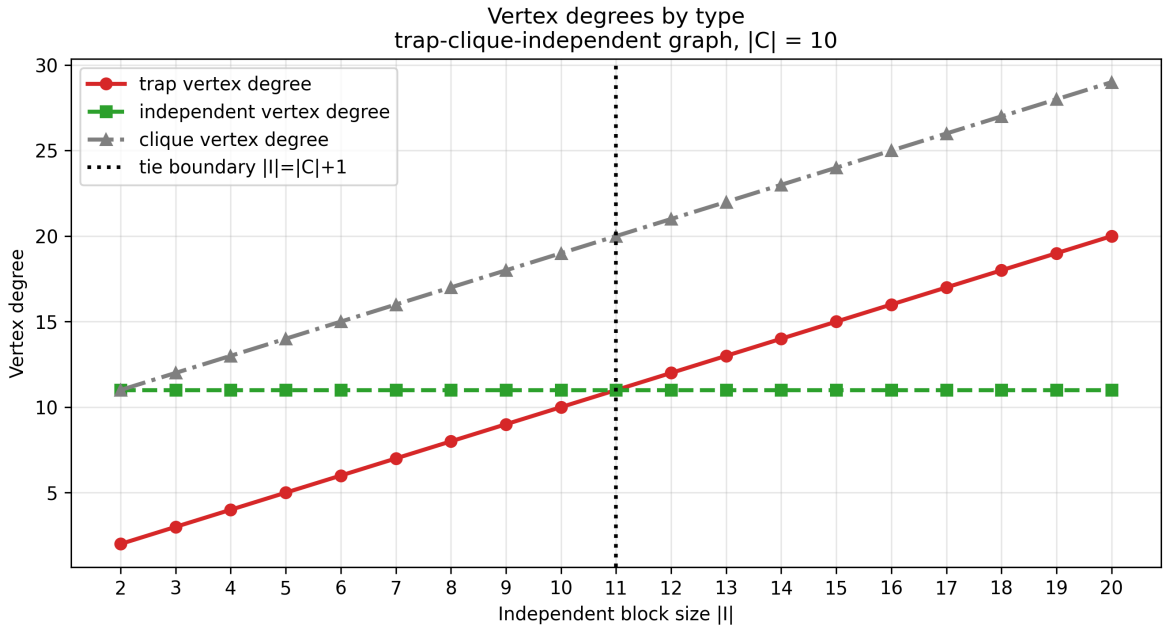


Figure D.2: Degrees of the three vertex types in the trap-clique-independent graph. The trap vertex has degree $|I|$, each independent-set vertex has degree $|C| + 1$, and each clique vertex has degree $|I| + |C| - 1$. Since Greedy selects a minimum-degree vertex, it chooses the trap vertex whenever $|I| < |C| + 1$.

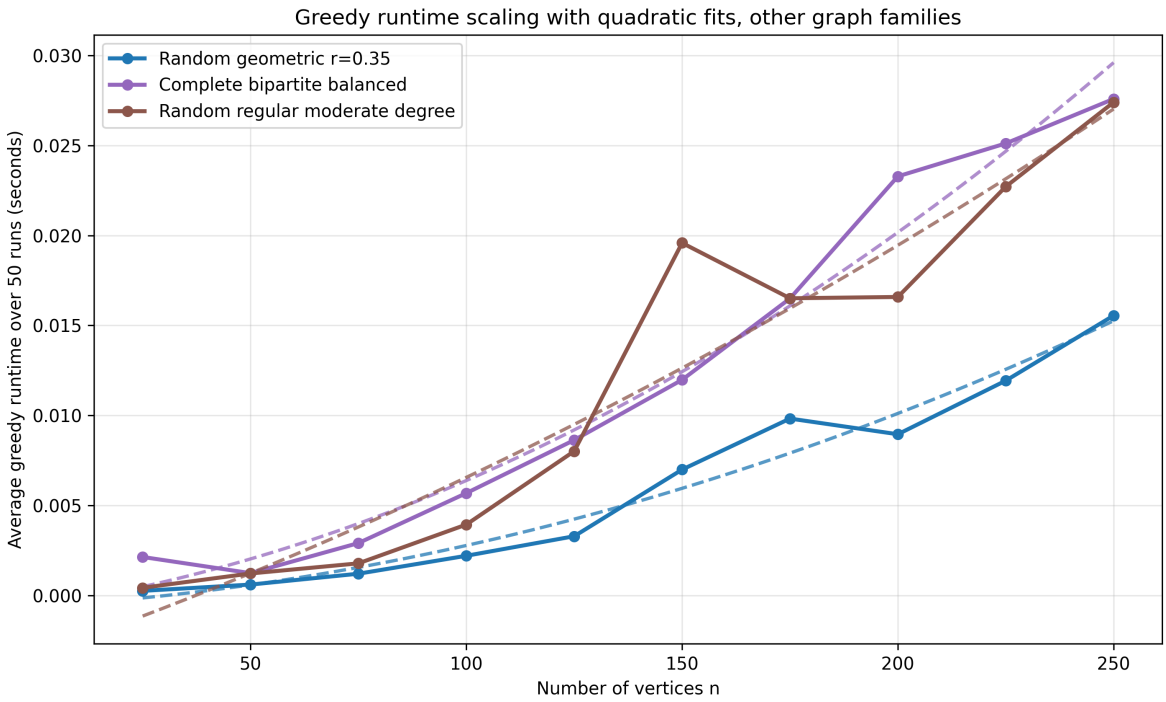


Figure D.3: Average runtime of the deterministic Greedy algorithm over 50 runs per graph size, shown for random geometric graphs with $r = 0.35$, balanced complete bipartite graphs, and random regular graphs of moderate degree. Solid lines show the measured mean runtime, while dashed lines show fitted quadratic runtime curves.

D.2. Lotka–Volterra Additional Results

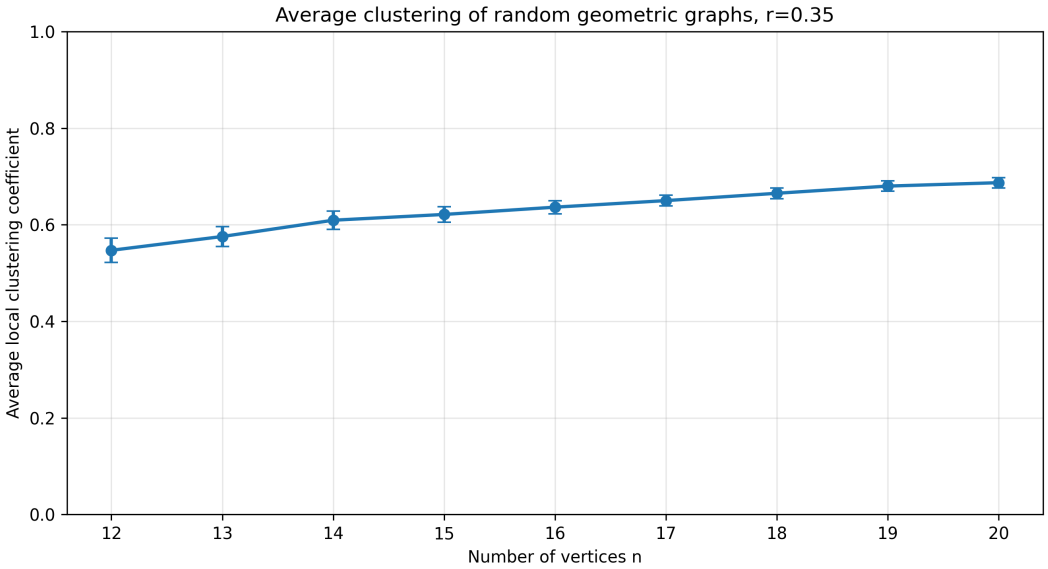


Figure D.4: Average local clustering coefficient of random geometric graphs with $r = 0.35$ for different values of n. The graphs have a relatively high average local clustering coefficient, around 0.6.

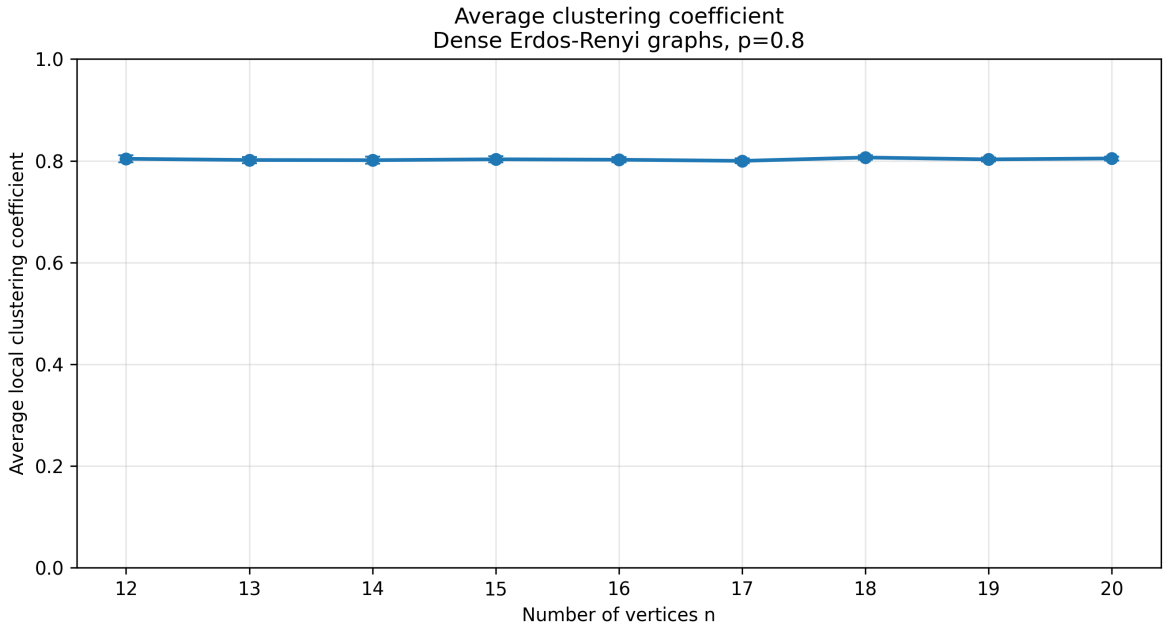


Figure D.5: Average local clustering coefficient of dense Erdős-Rényi graphs $G(n, 0.8)$ as a function of the number of vertices n. The clustering coefficient remains high across all tested graph sizes, reflecting the dense random structure where many pairs of neighboring vertices are also connected to each other.

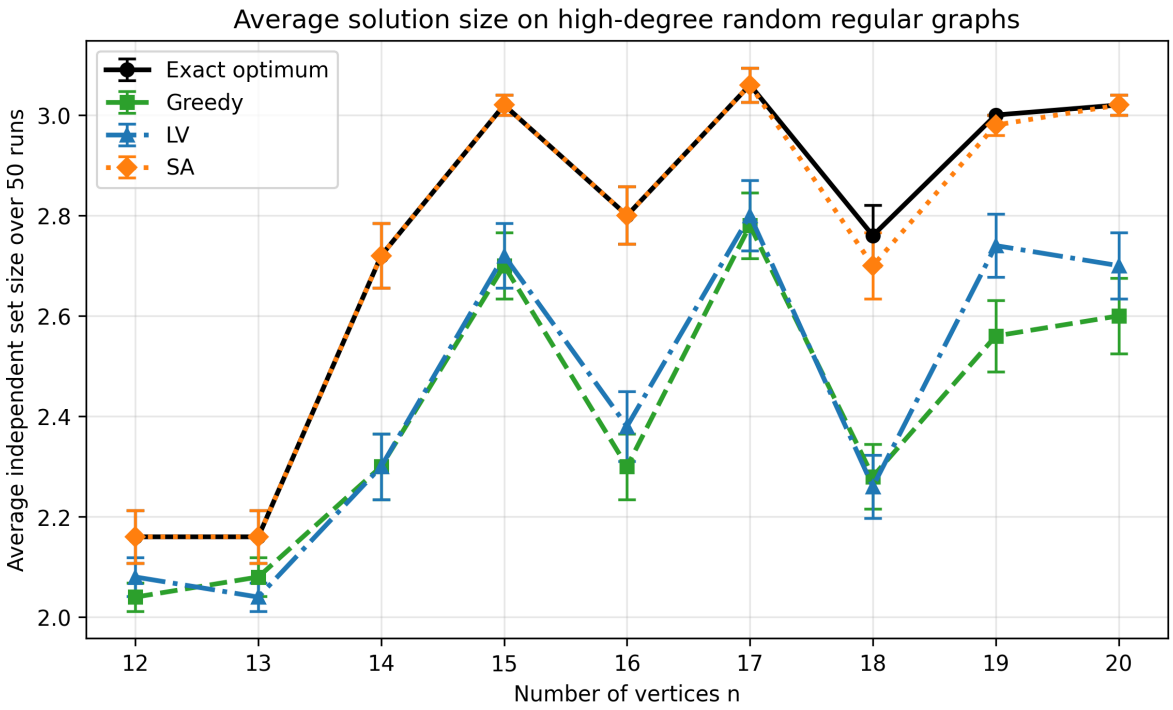


Figure D.6: Average solution size on high-degree random regular graphs for $n = 12, \ldots, 20$. The Lotka–Volterra algorithm finds smaller independent sets than the exact algorithm and the other heuristics.

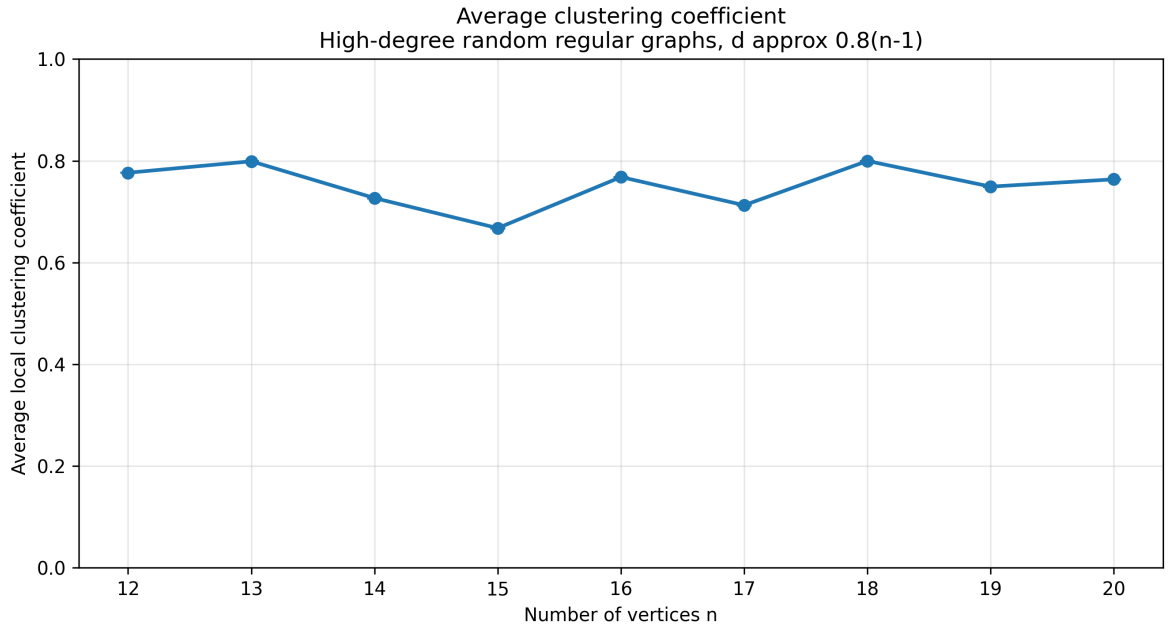


Figure D.7: Average local clustering coefficient of high-degree random regular graphs for $n = 12, \ldots, 20$, averaged over 50 graph instances for each n. The degree is chosen as a high fraction of $n - 1$, subject to the parity constraint that nd must be even. The degrees used are $d = 9, 10, 10, 10, 12, 12, 14, 14, 15$ for $n = 12, \ldots, 20$, respectively. The clustering coefficient remains high, roughly around 0.75, but shows small oscillations because the feasible regular degree does not increase smoothly with n. In particular, the downward jumps occur immediately after increases in the chosen degree d, showing that the oscillation is likely an artifact of the degree-selection rule rather than a structural trend in n alone.

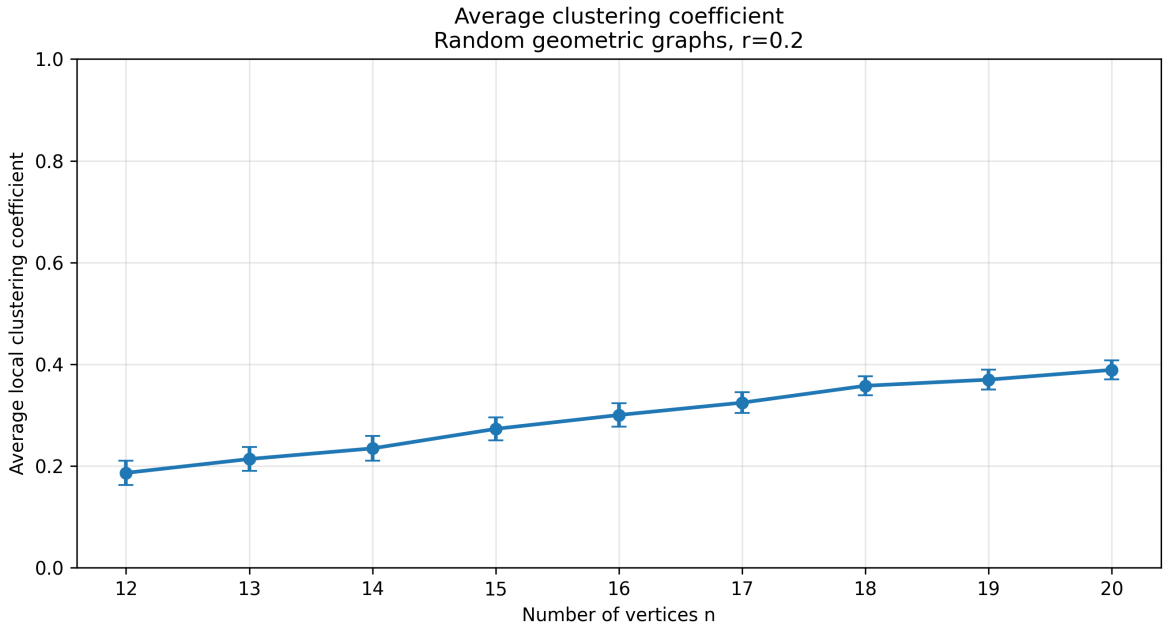


Figure D.8: Average local clustering coefficient of random geometric graphs with $r = 0.2$ for $n = 12, \ldots, 20$. Lowering the radius reduces the average clustering coefficient to about 0.3.

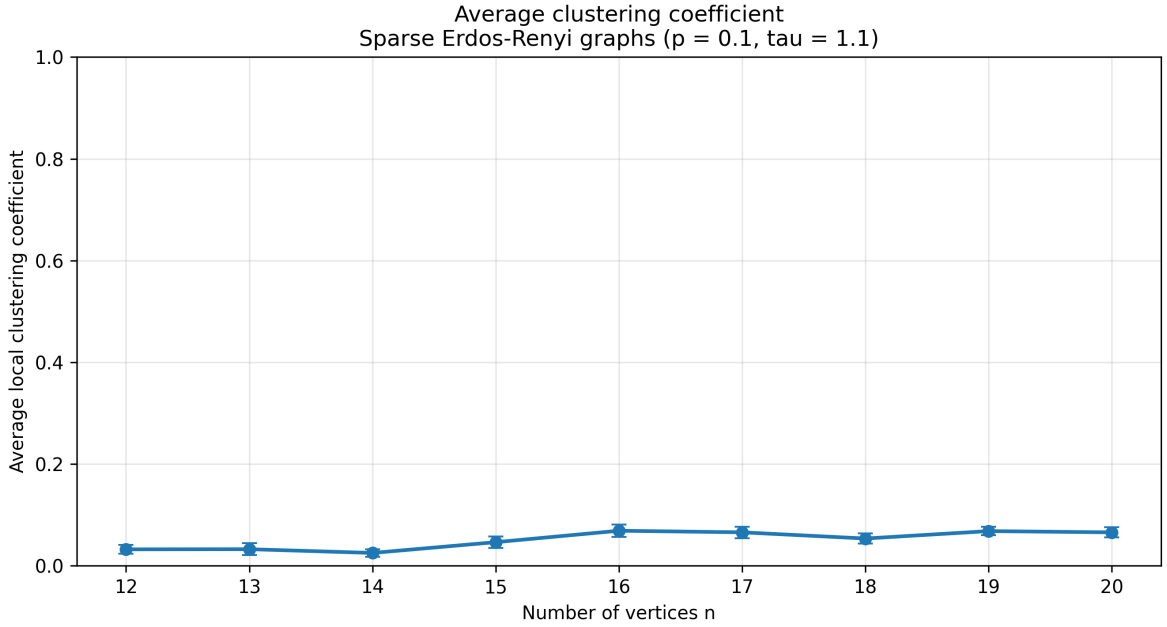


Figure D.9: Average local clustering coefficient of sparse Erdős-Rényi graphs with $p = 0.1$ for $n = 12, \ldots, 20$. The clustering coefficient is low, approximately 0.05, over the tested range.

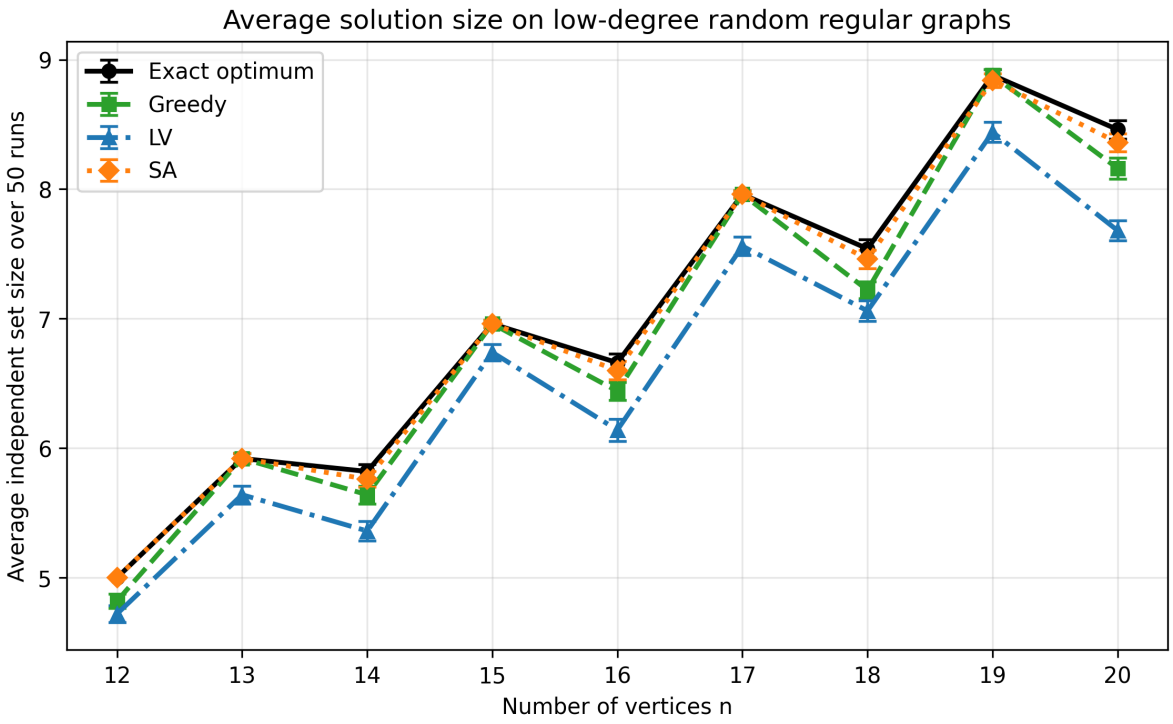


Figure D.10: Average solution size over 50 runs for each n on low-degree random regular graphs for $n = 12, \ldots, 20$. The intended degree is $d = 3$, but for odd n this is not possible because nd would be odd, so the degree is adjusted to $d = 2$. Thus the degrees used are $d = 3, 2, 3, 2, 3, 2, 3, 2, 3$ for $n = 12, \ldots, 20$, respectively. This parity adjustment explains the alternating behavior in the curves: the upward jumps occur when the graph switches back to $d = 3$, while the downward jumps occur when the graph switches to $d = 2$. The Lotka–Volterra algorithm performs worse than the exact algorithm and the other heuristics across most values of n.

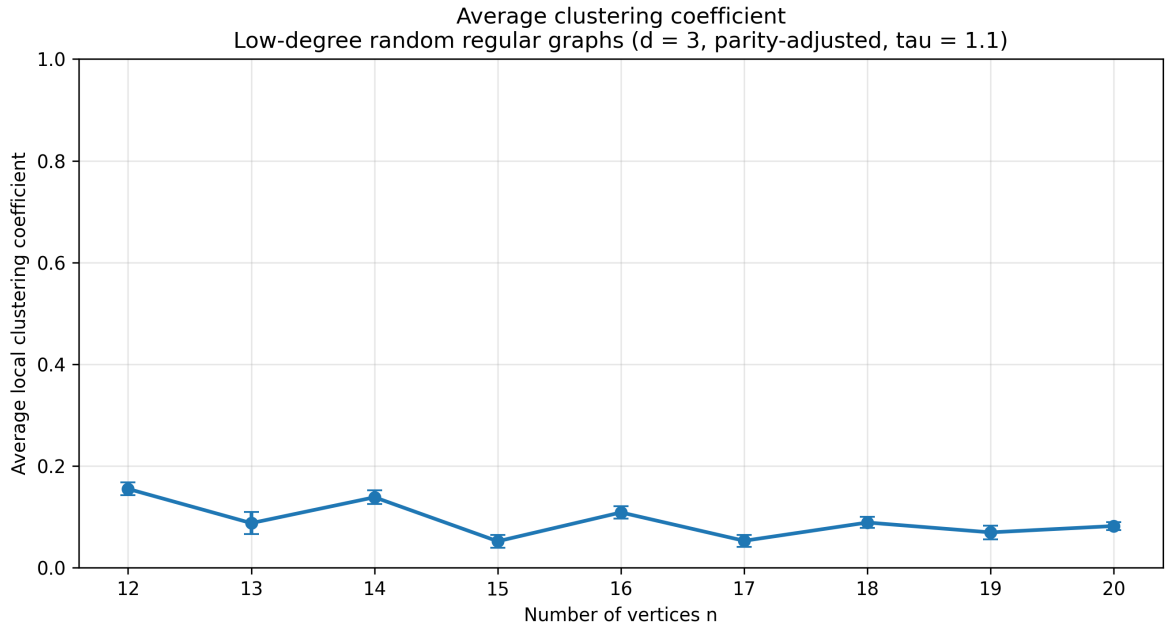


Figure D.11: Average local clustering coefficient of low-degree random regular graphs with degree $d = 3$ for $n = 12, \ldots, 20$. When needed, the degree is parity-adjusted so that nd is even and the random regular graph can be generated. The clustering coefficient is low, around 0.1.

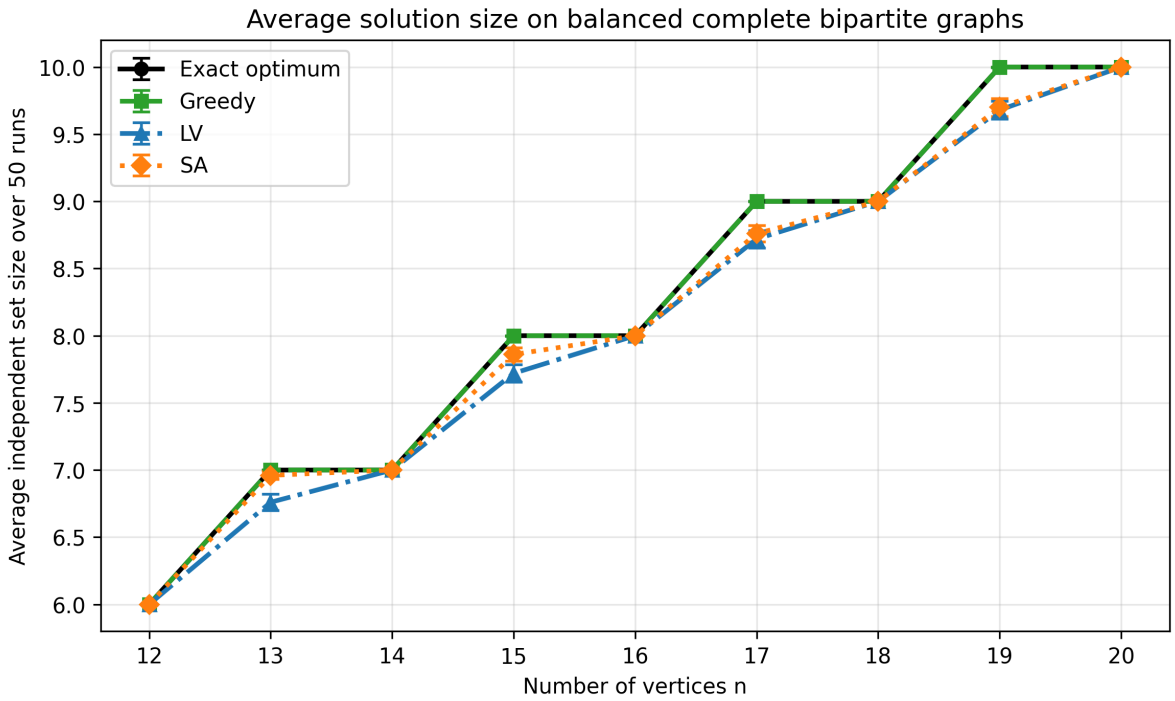


Figure D.12: Average solution size on balanced complete bipartite graphs for $n = 12, \ldots, 20$, using 10000 simulated annealing steps. In these graphs, the maximum independent set is one of the two sides of the bipartition. For odd n, the two sides differ by one vertex, while for the next even n, the sides become equal. Therefore the maximum independent set size is the same from each odd n to the next even n, which explains the flat segments in the plot.

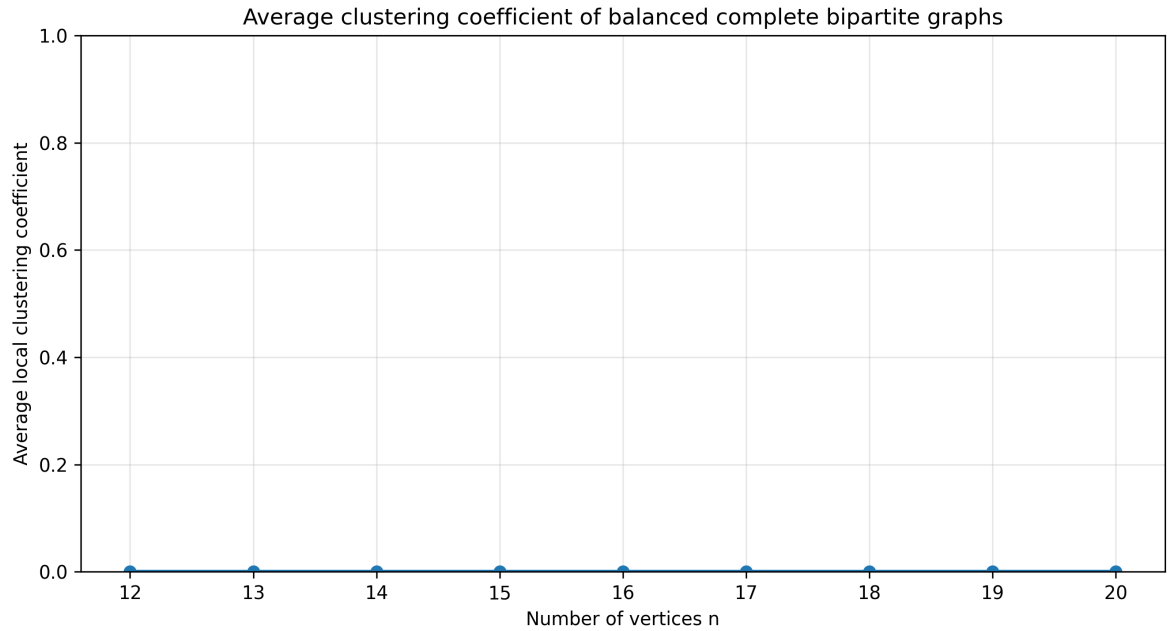


Figure D.13: Average local clustering coefficient of balanced complete bipartite graphs for $n = 12, \ldots, 20$. The clustering coefficient is 0, since bipartite graphs contain no triangles.

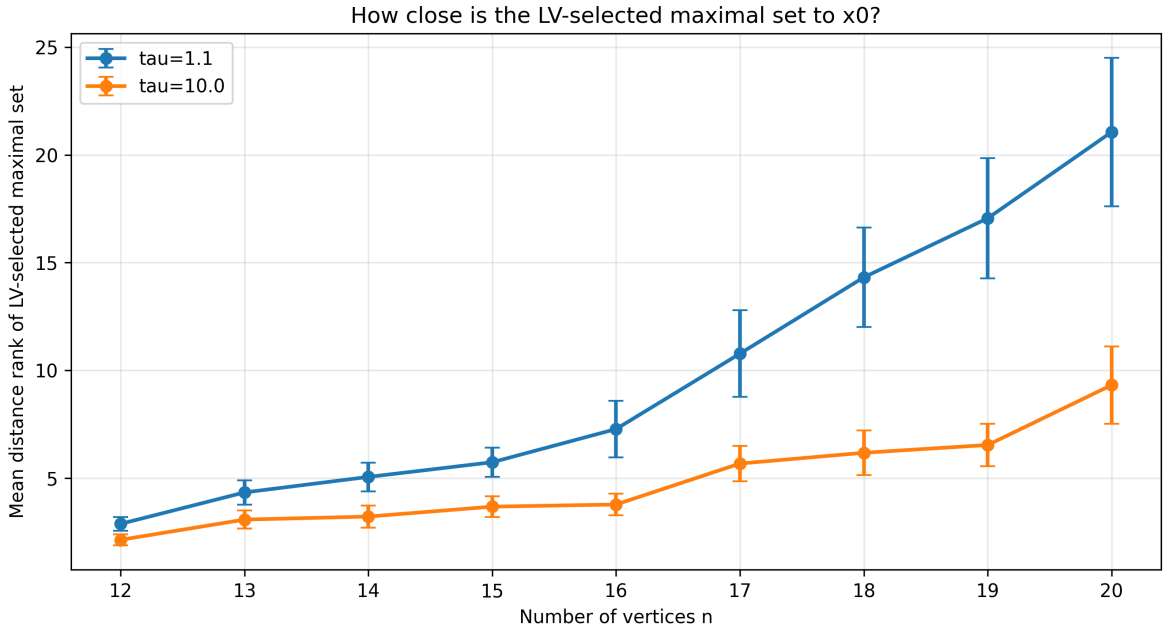


Figure D.14: Mean distance rank of the maximal independent set selected by the Lotka–Volterra algorithm for $n = 12, \ldots, 20$. Rank 1 corresponds to the maximal independent set closest to the initial condition x_0. Results are shown for $\tau = 1.1$ and $\tau = 10$, with error bars indicating variation over the runs.

D.3. Simulated Annealing Additional Results

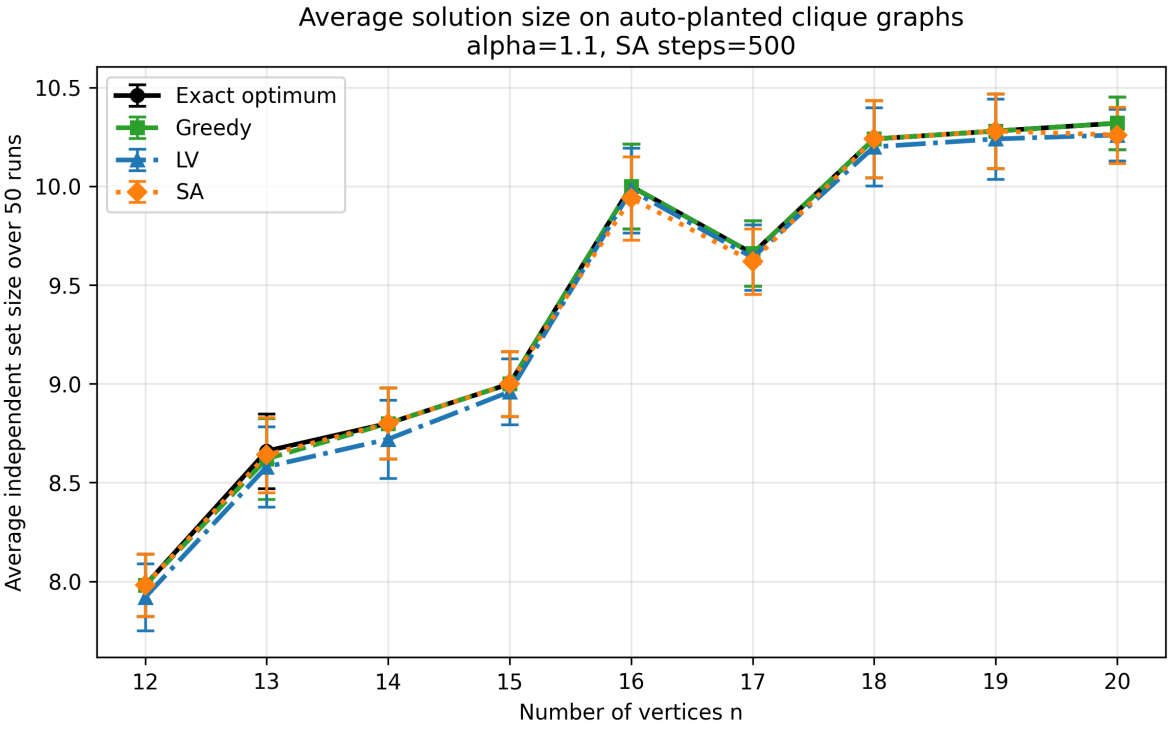


Figure D.15: Average solution size on auto-planted-clique graphs for $n = 12, \ldots, 20$, using $\alpha = 1.1$ and 3000 simulated annealing steps. The algorithm outputs are close to the exact optimum, showing that simulated annealing performs well in this small graph regime.

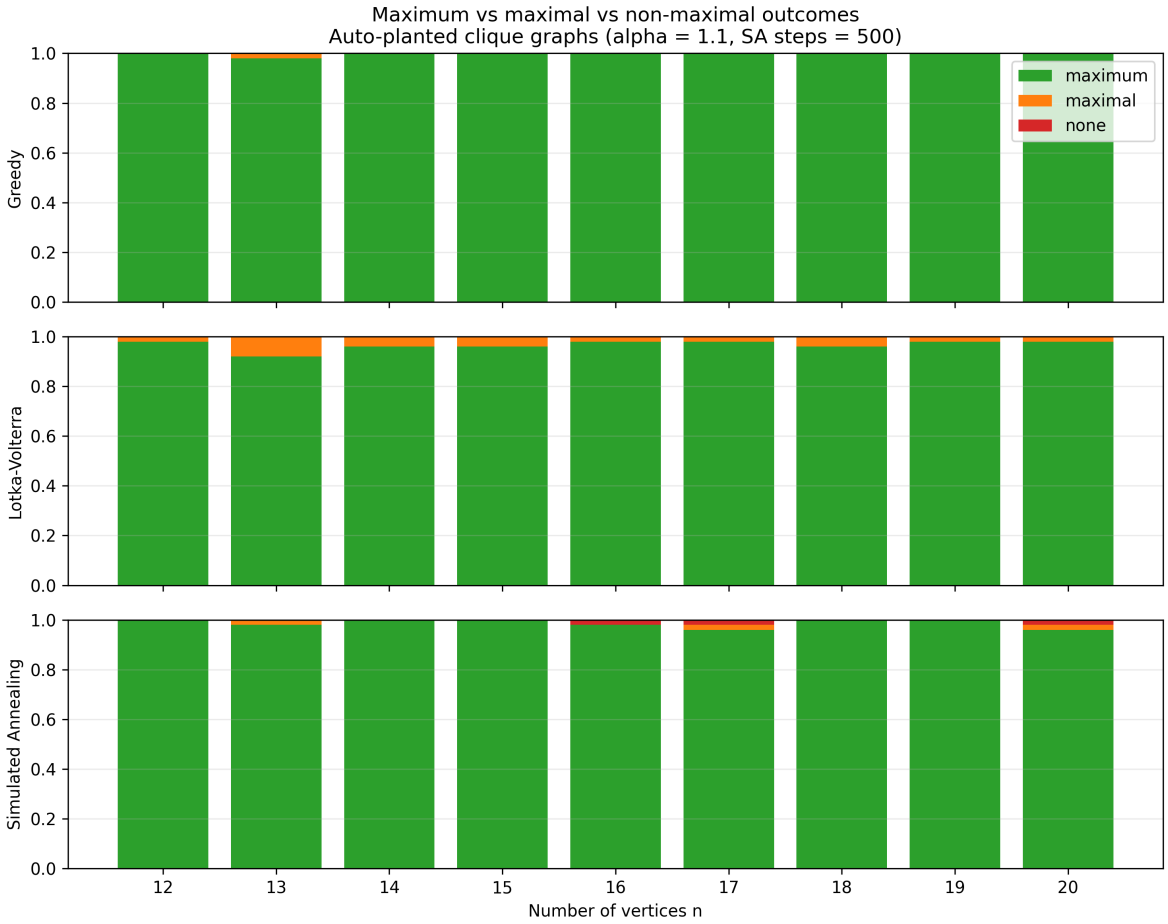


Figure D.16: Outcome classification for auto-planted-clique graphs with $n = 12, \ldots, 20$, $\alpha = 1.1$, and 3000 simulated annealing steps. The outcomes are classified as maximum, maximal but not maximum, or nonmaximal.

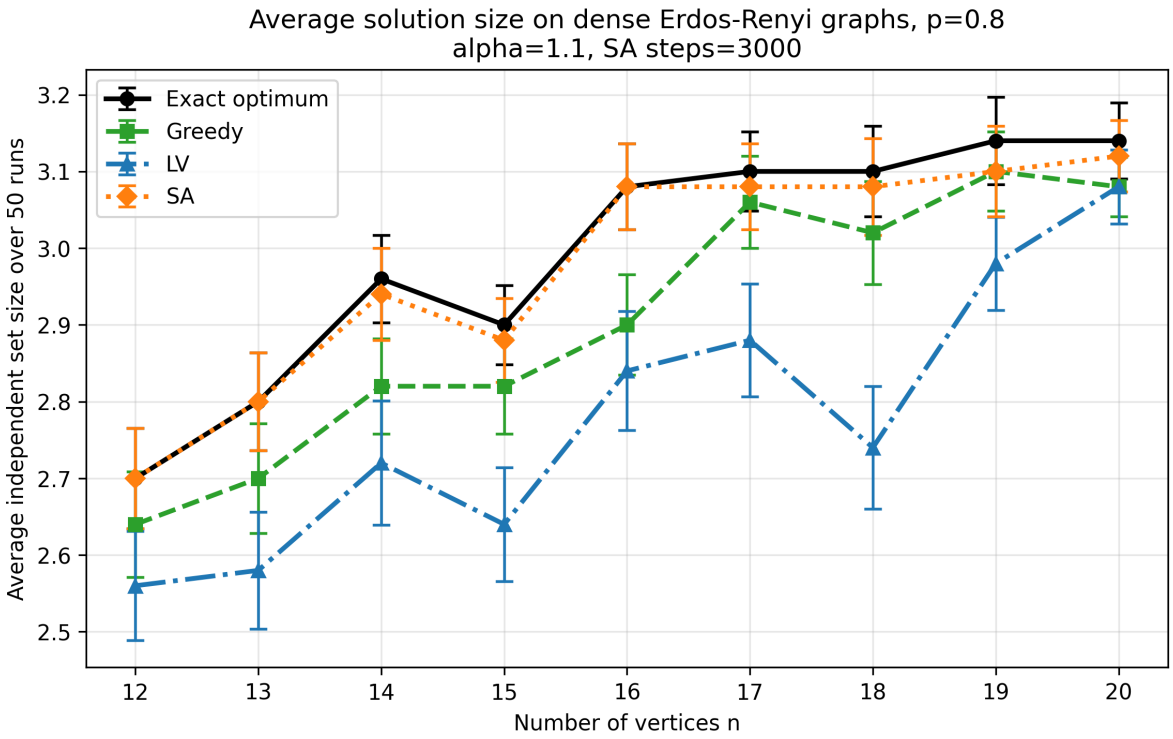


Figure D.17: Average solution size on dense Erdős-Rényi graphs with $p = 0.8$, for $n = 12, \ldots, 20$, using $\alpha = 1.1$ and 3000 simulated annealing steps. Simulated annealing remains close to the exact optimum, while the Lotka–Volterra algorithm performs worse on average. The small drop at $n = 15$ is caused by finite-sample variation in the random graph instances: the exact optimum also decreases slightly at this point. In this batch, fewer graphs at $n = 15$ have a maximum independent set of size 4, and slightly more have maximum independent set size 2, so the average is lower even though n is larger.

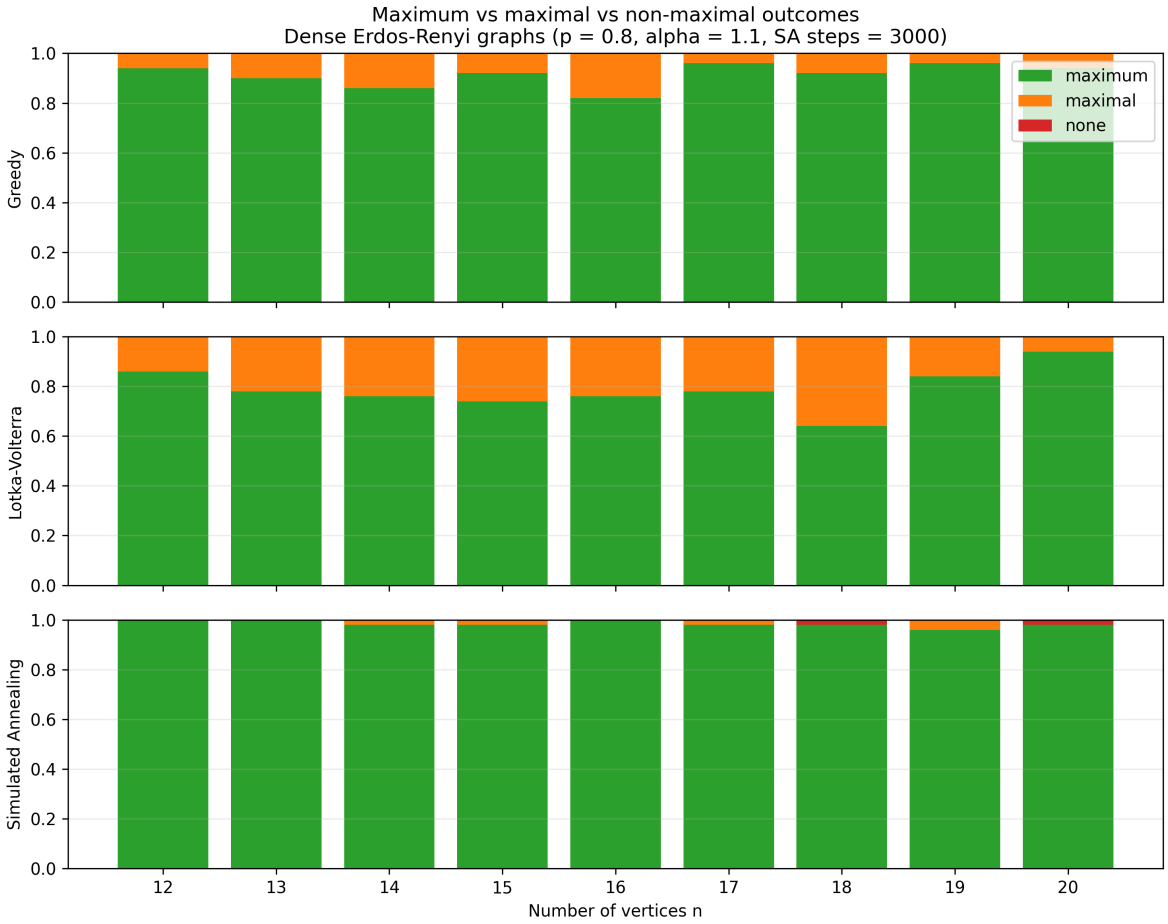


Figure D.18: Outcome classification for dense Erdős-Rényi graphs with $p = 0.8$, $n = 12, \ldots, 20$, $\alpha = 1.1$, and 3000 simulated annealing steps. The outcomes are classified as maximum, maximal but not maximum, or nonmaximal.

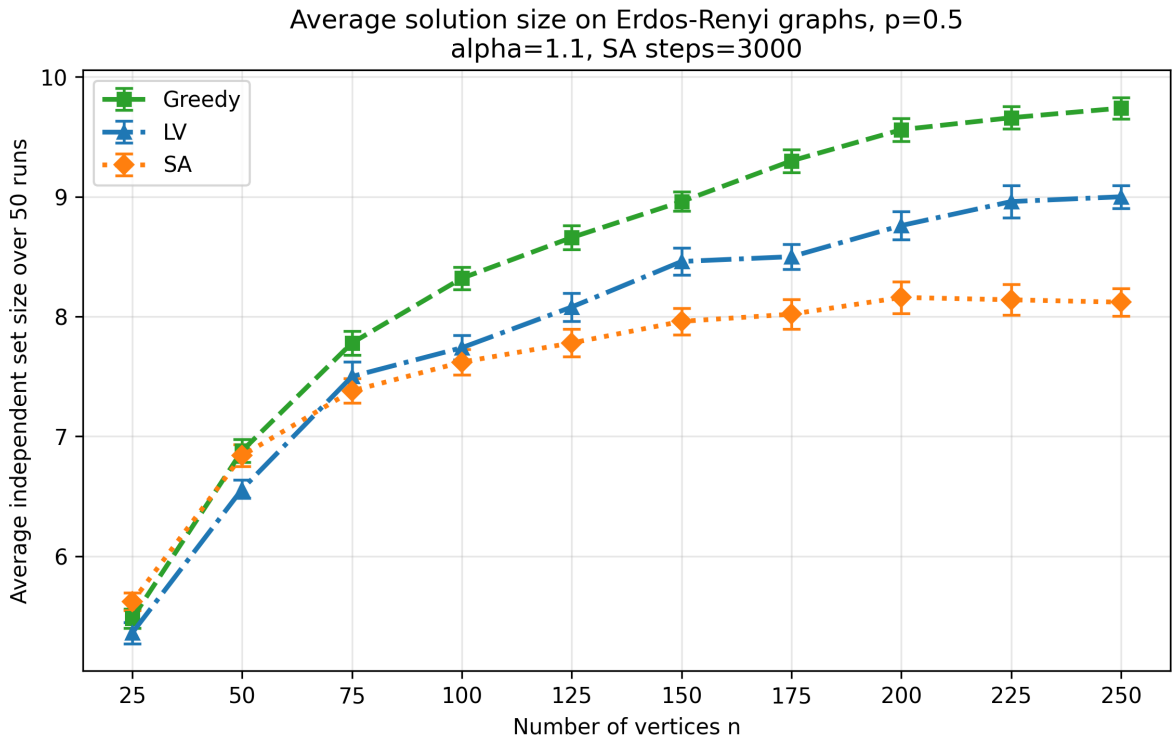


Figure D.19: Average solution size on Erdős-Rényi graphs with $p = 0.5$, using $\tau = 1.1$ for Lotka–Volterra and 500 steps for simulated annealing.

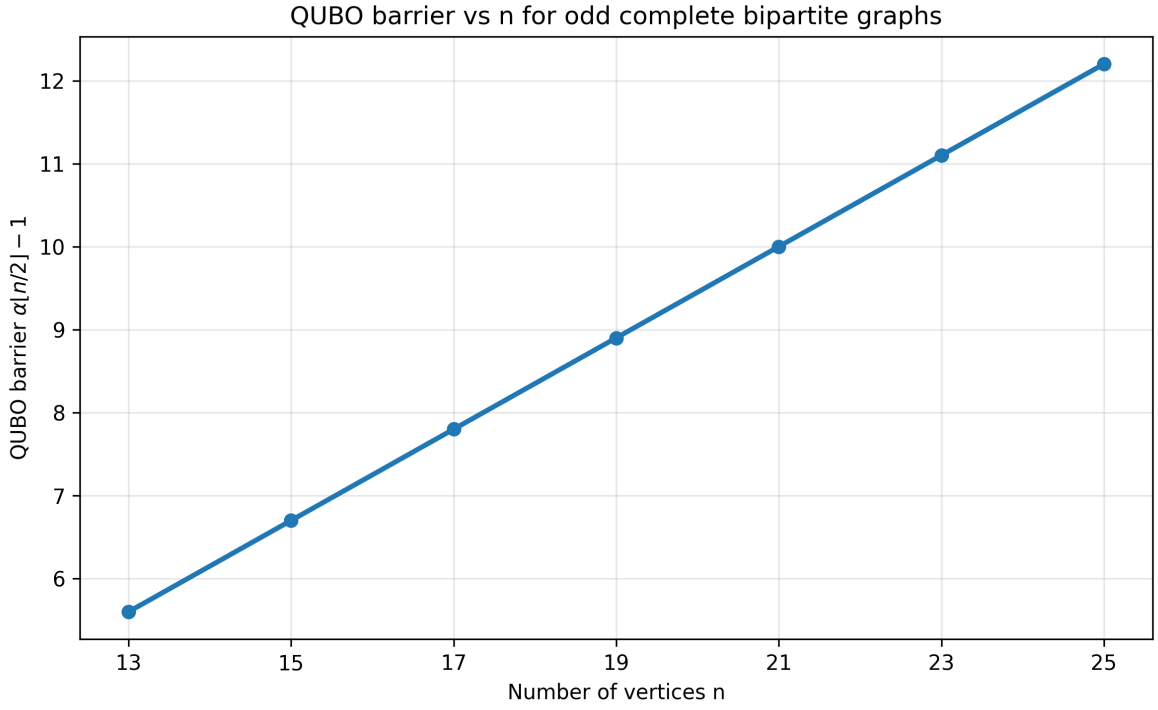


Figure D.20: QUBO barrier as a function of n for complete bipartite graphs $K_{\lfloor n/2 \rfloor, \lceil n/2 \rceil}$, using penalty parameter $\alpha = 1.1$.

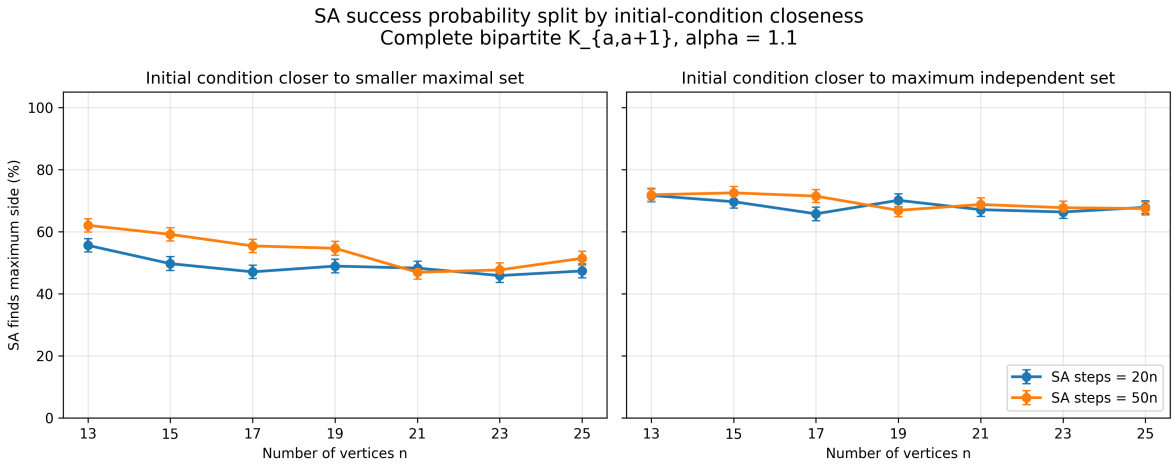


Figure D.21: Success probability of simulated annealing on complete bipartite graphs $K_{a,a+1}$, split by initial-condition closeness. The left panel includes runs whose initial state is closer to the smaller maximal independent set, while the right panel includes runs whose initial state is closer to the maximum independent set. The curves show the percentage of runs that find the larger side for two cooling lengths, $20n$ and $50n$, averaged over the conditional runs for each n. This split shows that recovery is harder when the initial condition is closer to the smaller maximal set, especially as n and the corresponding barrier increase.

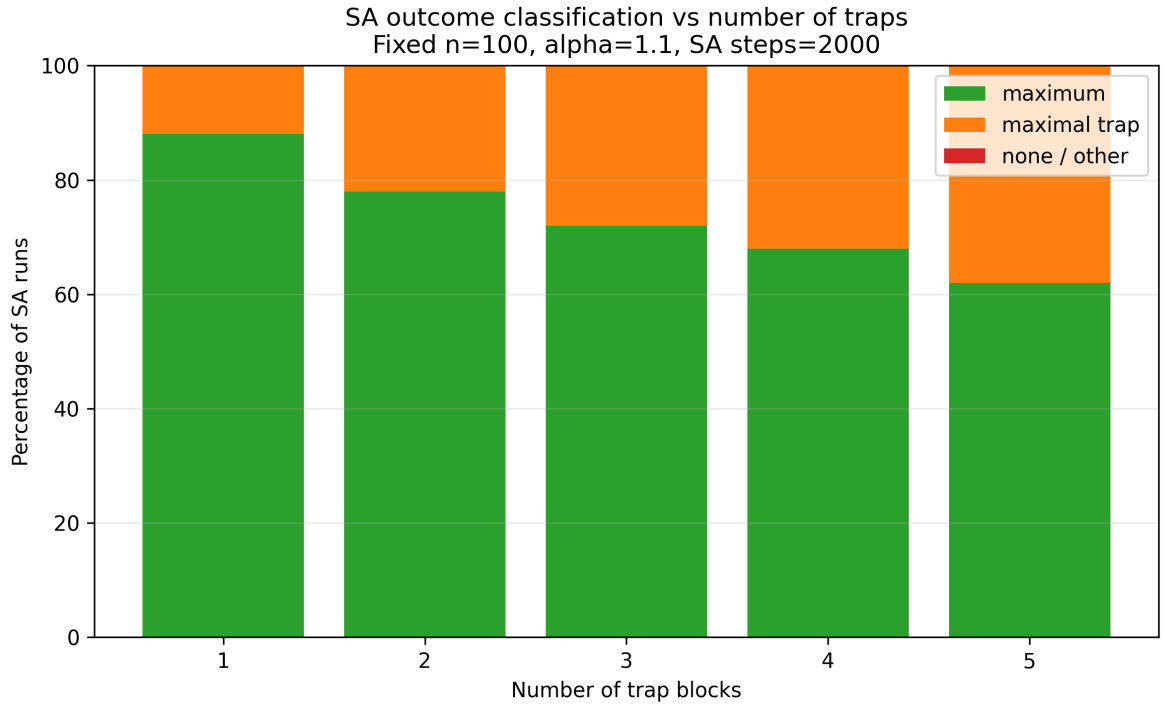


Figure D.22: Classification of simulated annealing outcomes as the number of trap blocks increases. Outcomes are separated into maximum independent set, trap maximal independent set, and none or other. As more trap blocks are added, simulated annealing more often converges to one of the trap maximal sets instead of the maximum, while nonmaximal outcomes remain absent.

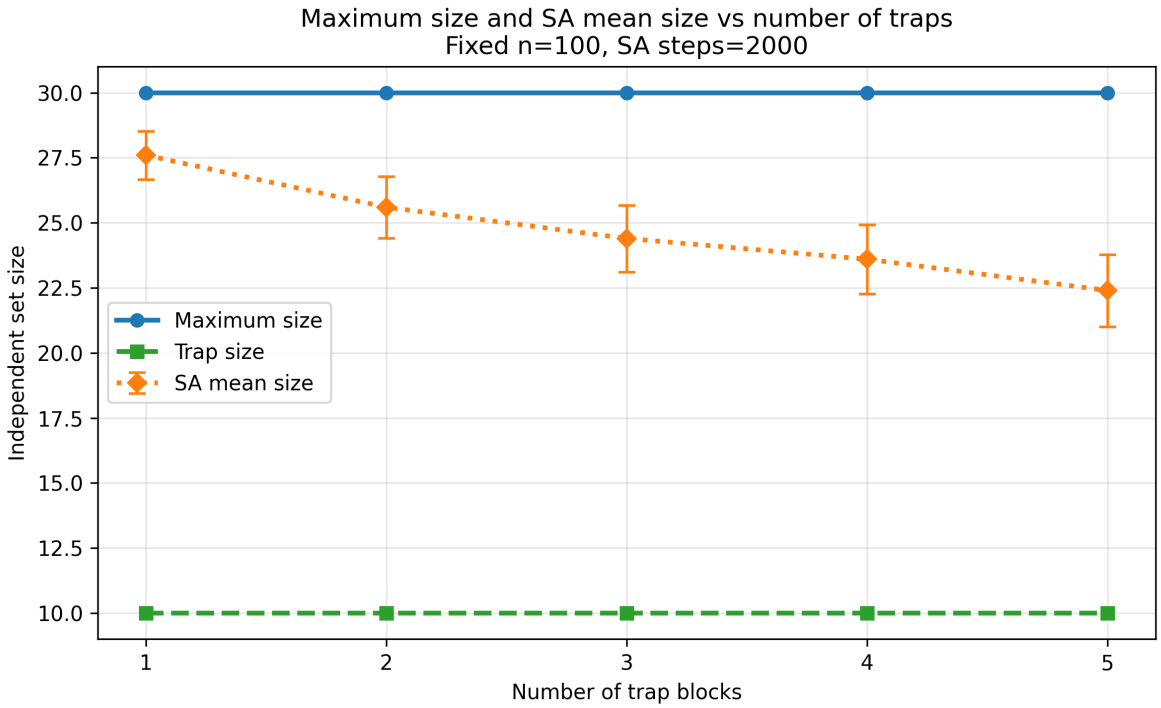


Figure D.23: Average solution size found by simulated annealing as the number of trap blocks increases, compared with the fixed maximum independent set size and fixed trap size. The maximum and trap sizes remain constant by construction, while the average simulated annealing solution size decreases, showing that the algorithm is increasingly pulled toward the smaller trap solutions.

E

Parameter Settings

The parameter settings used for the simulations are shown here.

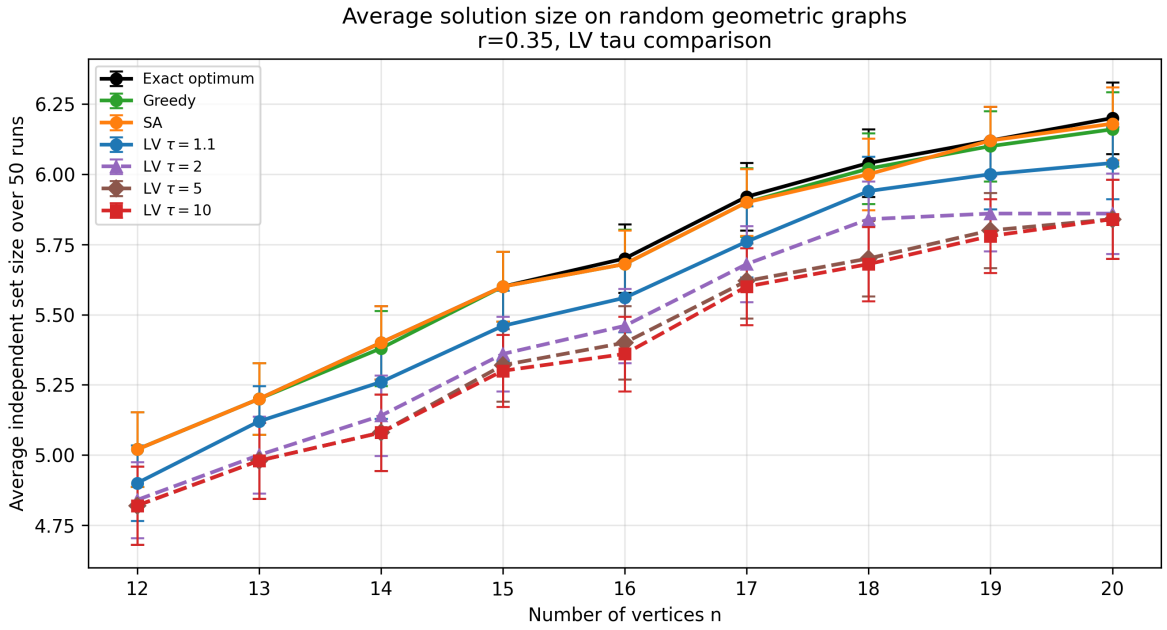


Figure E.1: Average independent set size over 50 runs for each n on random geometric graphs with connection radius $r = 0.35$. The plot compares the exact optimum, greedy algorithm, simulated annealing, and Lotka–Volterra with $\tau = 1.1$, $\tau = 2$, $\tau = 5$, and $\tau = 10$.

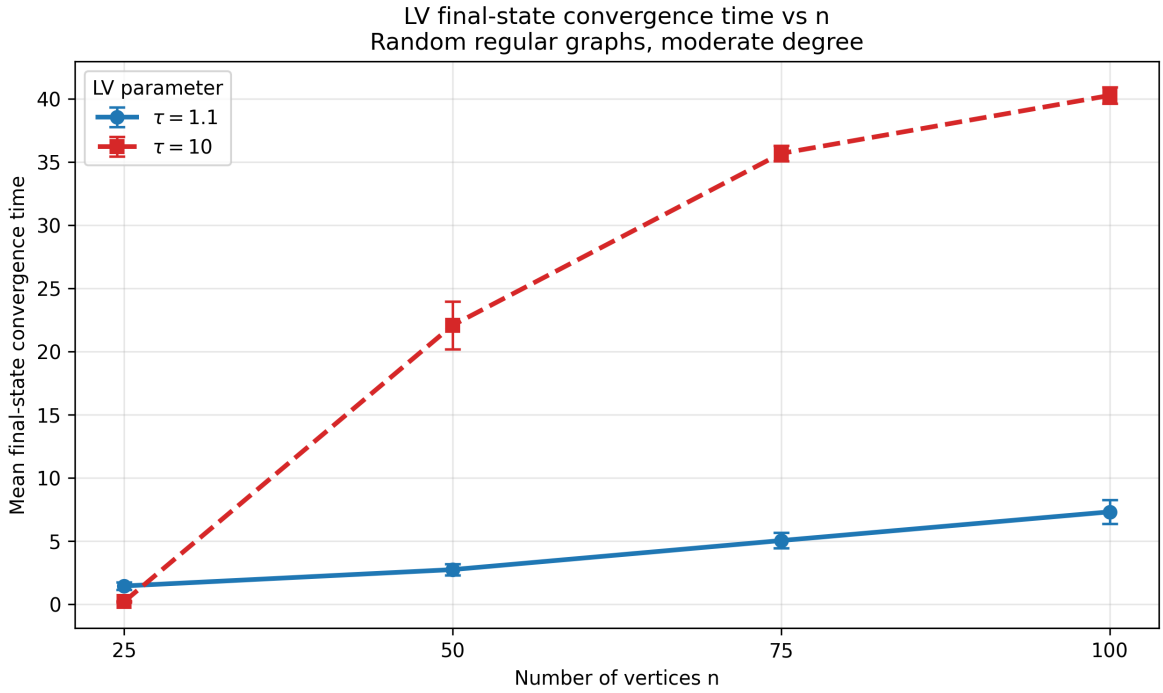


Figure E.2: Mean Lotka–Volterra final-state convergence time on random regular graphs of moderate degree, averaged over 50 runs for each n. Convergence time is measured as the first time after which the thresholded Lotka–Volterra-selected independent set remains equal to the final selected set. The two lines compare $\tau = 1.1$ and $\tau = 10$.

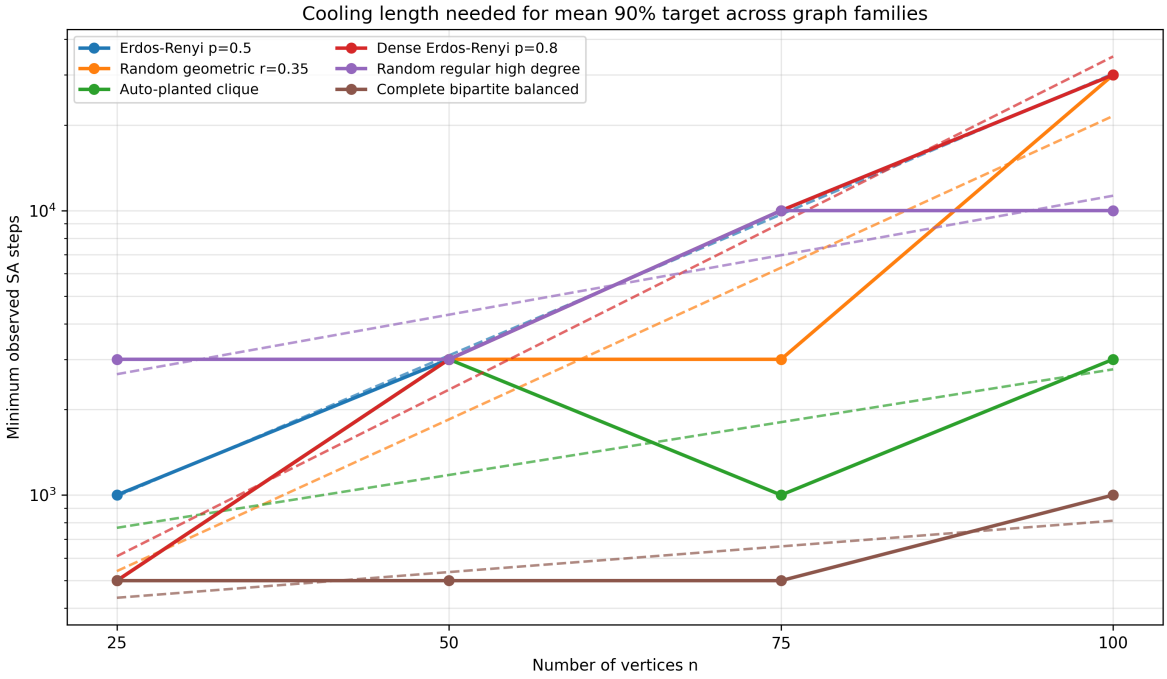


Figure E.3: Minimum number of simulated annealing steps required for the mean solution size to reach at least 90% of the best-known solution size, shown for several graph families with $n = 25, 50, 75, 100$. Solid lines show the observed required step counts, while dashed lines show exponential fits on a log-scaled y-axis. Dense Erdős-Rényi and random geometric graphs require the fastest growth in cooling length, while complete bipartite graphs remain comparatively easy.

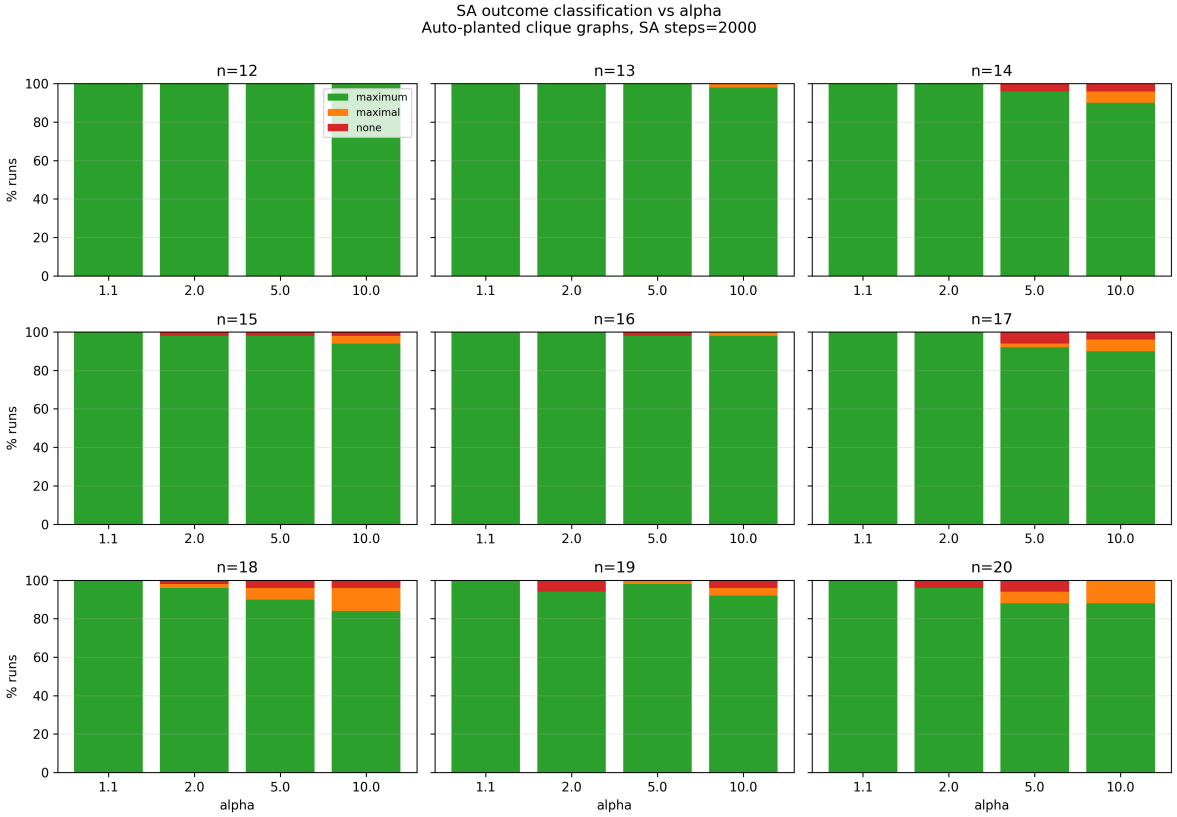


Figure E.4: Classification of simulated annealing outputs on auto-planted clique graphs for different penalty parameters α, using 2000 SA steps. For each n, the bars show the percentage of 50 runs classified as maximum, maximal but not maximum, or non-maximal after repair to an independent set. Smaller values of α generally recover the maximum more reliably, while larger penalties can reduce the acceptance of conflict-creating moves and lead to more maximal or non-maximal outcomes, especially for larger graphs.

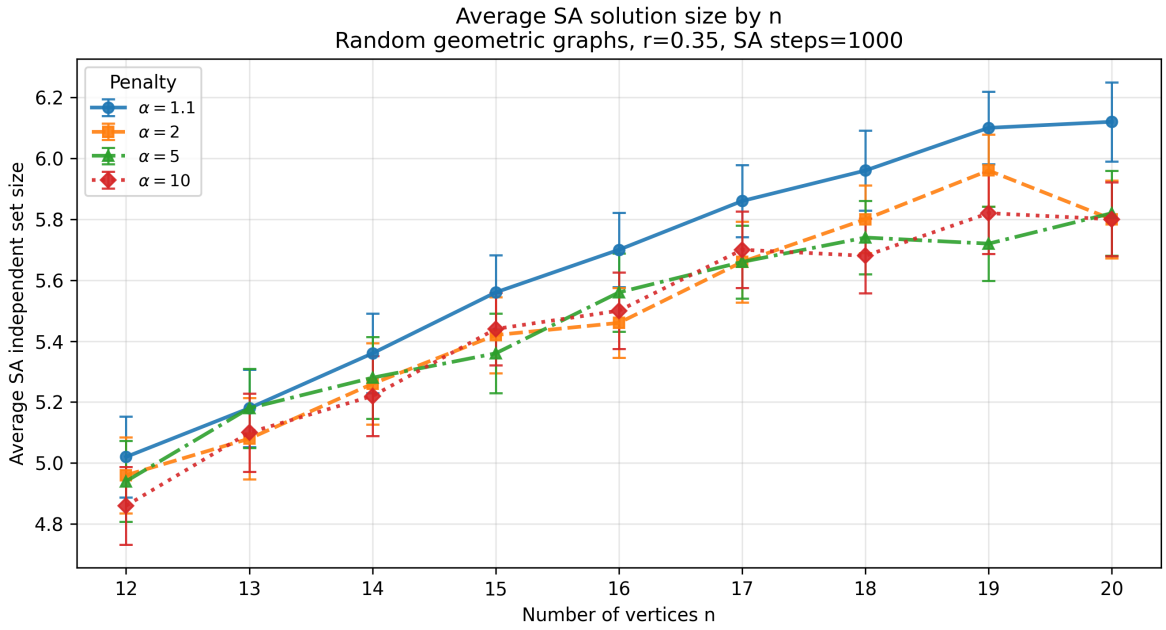


Figure E.5: Average independent set size found by simulated annealing on random geometric graphs with $r = 0.35$, for different penalty values α. The number of SA steps is fixed, so the differences between the lines show how changing α affects solution quality.

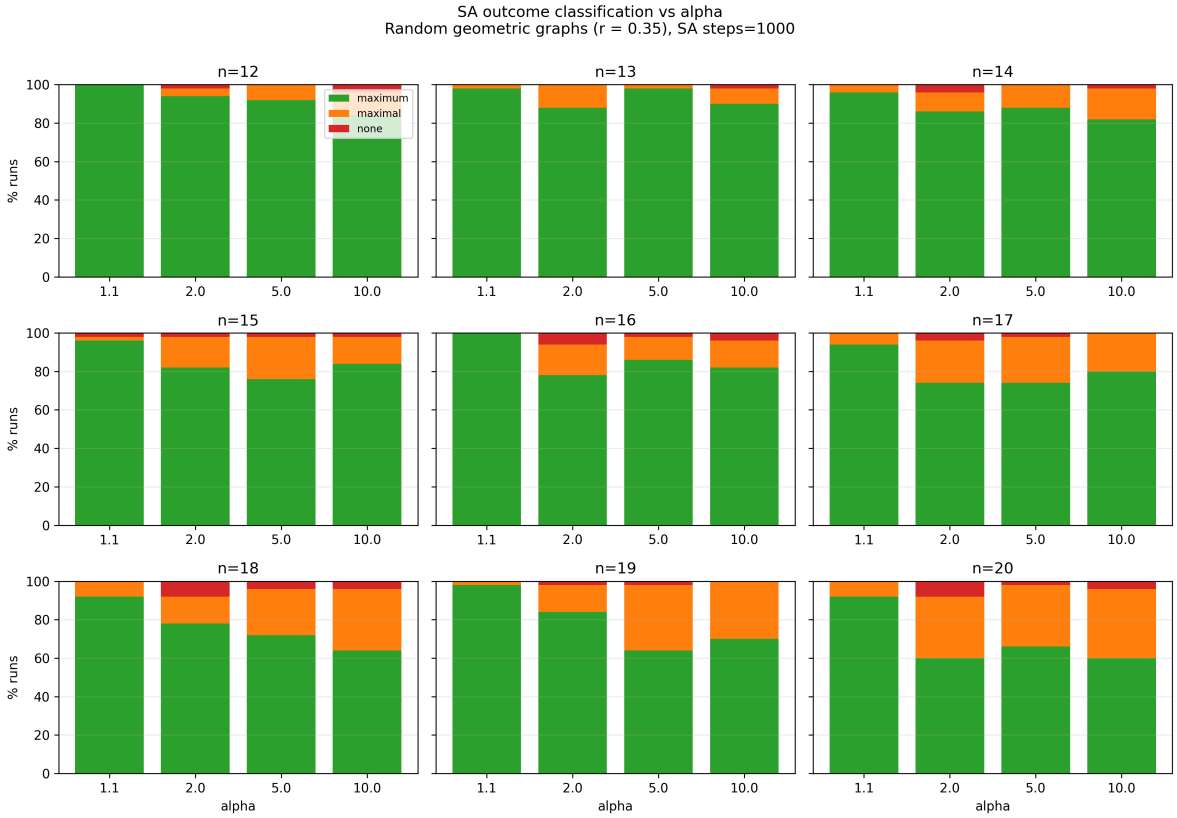


Figure E.6: Classification of simulated annealing outputs on random geometric graphs with $r = 0.35$, for different penalty values α. The outputs are classified as maximum, maximal but not maximum, or non-maximal.

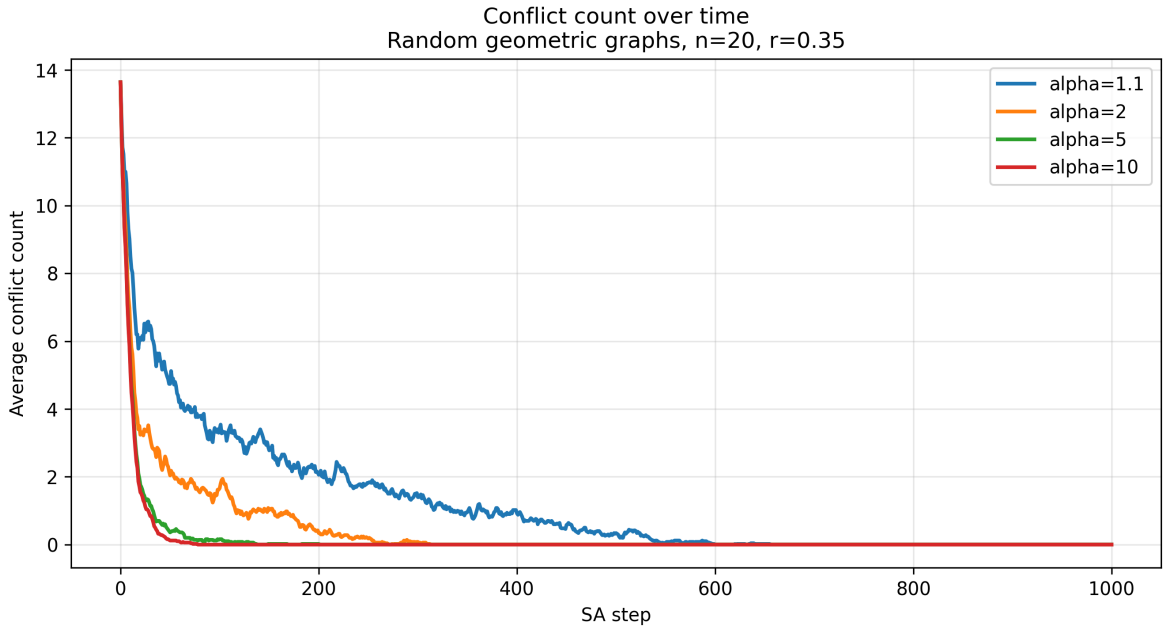


Figure E.7: Average number of edge conflicts during simulated annealing for different penalty values α. Larger values of α remove conflicts earlier, showing stronger pressure toward feasible independent sets.

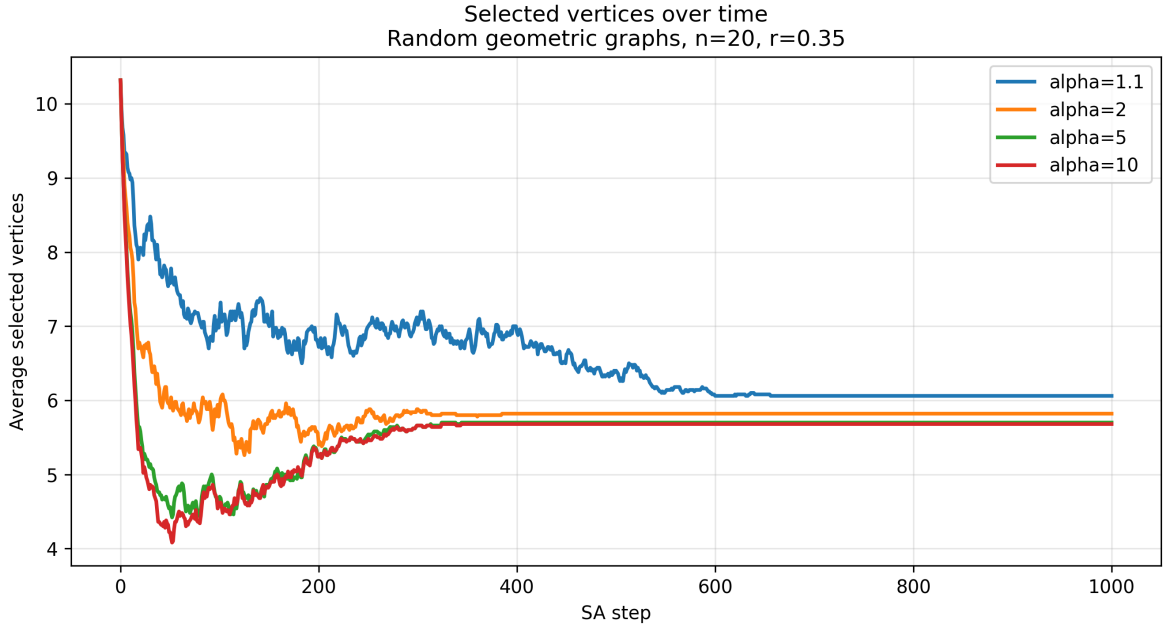


Figure E.8: Average number of selected vertices during simulated annealing for different penalty values α. Larger values of α lead to smaller intermediate candidate sets.

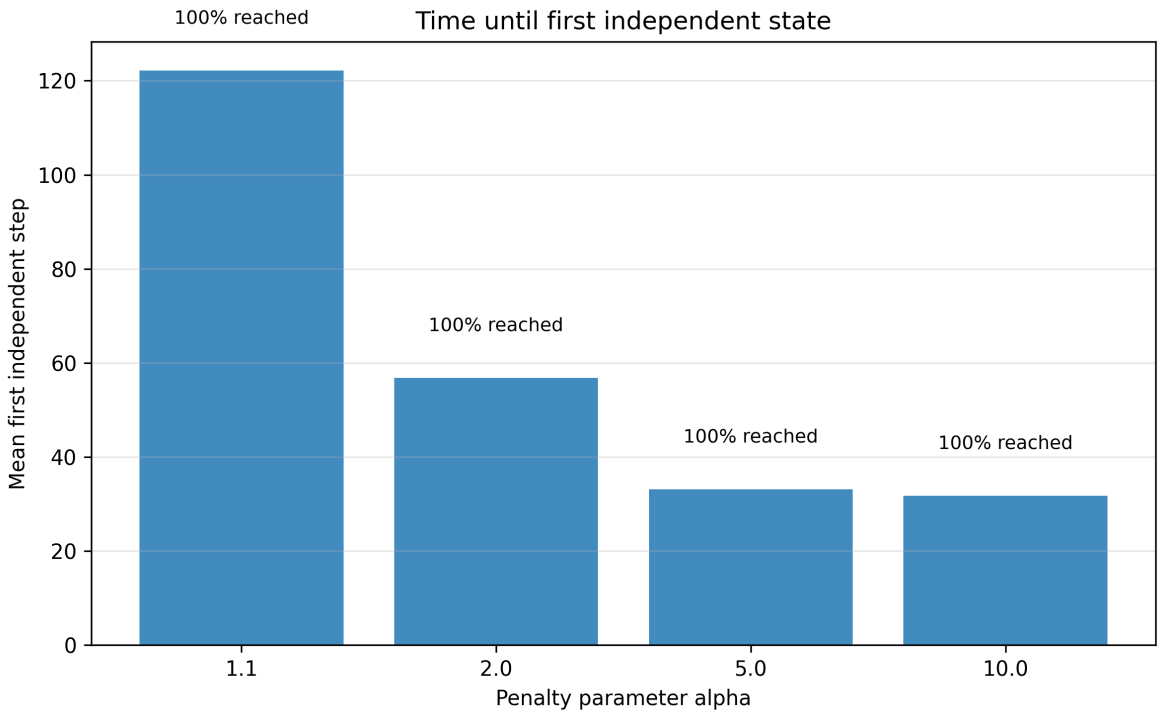


Figure E.9: First time step at which simulated annealing reaches an independent set, shown for different penalty values α. Larger values of α drive the system toward feasibility earlier.