UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RICARDO MATSUMURA DE ARAÚJO

Memetic Networks: problem-solving with social network models

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

Prof. Dr. Luis C. Lamb
Advisor

Porto Alegre, december 2010

CIP – CATALOGING-IN-PUBLICATION

Araújo, Ricardo Matsumura de

Memetic Networks: problem-solving with social network models / Ricardo Matsumura de Araújo. – Porto Alegre: PPGC da UFRGS,

.

135 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS,

. Advisor: Luis C. Lamb.

1. Social Computing. 2. Memetic Network. 3. Search. 4. Optimization. 5. Artificial Intelligence. I. Lamb, Luis C.. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Pró-Reitor de Coordenação Acadêmica: Prof. Rui Vicente Oppermann
Pró-Reitor de Pós-Graduação: Prof. Aldo Bolten Lucion
Diretor do Instituto de Informática: Prof. Prof. Flávio Rech Wagner
Coordenador do PPGC: Prof. Alvaro Freitas Moreira
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

AGRADECIMENTOS

Esta tese não teria sido possível se não fosse pelo meu orientador, prof. Luis Lamb. Mais do que pelas valiosas orientações, sugestões e correções, o agradecimento a ele é devido principalmente por ter sempre garantido que seus orientandos tenham a maior liberdade possível para trilhar seus próprios caminhos na pesquisa. Mostrou-me que o doutorado é o local onde devemos fazer nossas próprias perguntas, ao invés de apenas responder as dos outros. Espero que esta tese esteja a altura da confiança que me foi depositada.

A Raquel Recuero que, durante este processo de doutoramento, fez todo o caminho namorada-noiva-esposa (e futura mãe de minha filha Emilia), sempre sendo minha principal motivadora. É dela todo o crédito por ter-me apresentado à fascinante área das redes sociais. Esta tese, e minha vida, certamente seria completamente diferente (e pior) sem ela. Ao nosso cachorro, Chewie, por não ter pressionado acidentalmente (apesar das tentativas) nenhuma tecla que fosse fatal a esta tese.

Aos meus pais, Eduardo e Yayoi, e minha irmã, Márcia, por terem me acompanhado e provido tudo que precisei para chegar até aqui.

A Lada Adamic, da Universidade de Michigan em Ann Arbor, e seus orientandos, pelas valiosas discussões e apontamentos.

A meus colegas da UFPel e ex-colegas da FURG, por terem me permitido usar a carta "estou fazendo Doutorado", mais vezes do que seria aconselhável, para garantir o tempo livre para finalizar esta Tese. Garanto-lhes que o tempo foi bem utilizado.

Por fim, agradeço as agências de fomento, CNPq e CAPES, pelo suporte financeiro necessário, e aos inúmeros revisores anônimos dos periódicos e conferências onde parte dos resultados aqui descritos foram publicados ou submetidos para publicação.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

SWN	Small-World Network
SFN	Scale-Free Network
MNA	Memetic Network Algorithm
SIP	Social Information Processing
MAS	Multi-agent System
GA	Genetic Algorithm
HC	Hill-Climbing
LBS	Local Beam Search
VLSI	Very Large Scale Integration
DYN-MNA	Dynamic Memetic Network Algorithm
SWN-MNA	Small-World Network Memetic Network Algorithm
SFN-MNA	Scale-Free Network Memetic Network Algorithm

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Social systems are increasingly relevant to computer science in general and artificial intelligence in particular. Such interest was first sparkled by agent-based systems where the social interaction of such agents can be relevant to the outcome produced. A more recent trend comes from the general area of Social Information Processing, Social Computing and other *crowdsourced* systems, which are characterized by computing systems composed of people and strong social interactions between them. The set of all social interactions and actors compose a *social network*, which may have strong influence on how effective the system can be. In this thesis, we explore the role of network structure in social systems aiming at solving problems, focusing on numerical and combinatorial optimization. We frame problem solving as a search for valid solutions in a state space and propose a model - the Memetic Network - that is able to perform search by using the exchange of information, named *memes*, between actors interacting in a social network. Such model is applied to a variety of scenarios and we show that the presence of a social network greatly improves the system capacity to find good solutions. In addition, we relate specific properties of many well-known networks to the behavior displayed by the proposed algorithms, resulting in a set of general rules that may improve the performance of such social systems. Finally, we show that the proposed algorithms can be competitive with traditional heuristic search algorithms in a number of scenarios.

Keywords: Social Computing, Memetic Network, Search, Optimization, Artificial Intelligence.

Redes Meméticas: solução de problemas utilizando modelos de redes sociais

RESUMO

Sistemas sociais têm se tornado cada vez mais relevantes para a Ciência da Computação em geral e para a Inteligência Artificial em particular. Tal interesse iniciou-se pela necessidade de analisar-se sistemas baseados em agentes onde a interação social destes agentes pode ter um impacto no resultado esperado. Uma tendência mais recente vem da área de Processamento Social de Informações, Computação Social e outros métodos *crowdsourced*, que são caracterizados por sistemas de computação compostos de pessoas reais, com um forte componente social na interação entre estas. O conjunto de todas interações sociais e os atores envolvidos compõem uma rede social, que pode ter uma forte influência em o quão eficaz ou eficiente o sistema pode ser. Nesta tese, exploramos o papel de estruturas de redes em sistemas sociais que visam a solução de problemas. Enquadramos a solução de problemas como uma busca por soluções válidas em um espaço de estados e propomos um modelo - a Rede Memética - que é capaz de realizar busca utilizando troca de informações (memes) entre atores interagindo em uma rede social. Tal modelo é aplicado a uma variedade de cenários e mostramos como a presença da rede social pode melhorar a capacidade do sistema em encontrar soluções. Adicionalmente, relacionamos propriedades específicas de diversas redes bem conhecidas ao comportamento observado para os algoritmos propostos, resultando em um conjunto de regras gerais que podem melhorar o desempenho de tais sistemas sociais. Por fim, mostramos que os algoritmos propostos são competitivos com técnicas tradicionais de busca heurística em diversos cenários.

Palavras-chave: computação social, redes meméticas, inteligência artificial, busca, otimização.

1 INTRODUCTION

Social systems, those composed of many interacting actors involved in some sort of social relation, have gained increasing attention in the computer science and Artificial Intelligence (AI) communities for a number of reasons (LAZER et al., 2009; LEAKE, 2008). In one way, improved algorithms are needed to handle, analyze and extract knowledge from massive datasets generated by social interaction (e.g. citation of scientific papers, e-mail exchange, hyperlinks in websites). In that sense, just like the need to analyze DNA and other biological structures spawned the area of Bioinformatics, the need to analyze complex social structures is giving birth to the field of Computational Social Science (LAZER et al., 2009), which uses computers and algorithms to analyze and understand social systems. Hence, this area include social simulation of real social systems, algorithms to extract patterns from social interactions, database structures to store social relations and so on.

Running in the other direction, there is a growing interest in doing exactly the opposite: to use social systems to *produce* computation. This approach can be broadly divided in two main types. In the first, social structures are used to compose artificial systems that aim at solving problems. This is the case of Multi-Agent Systems (MAS), which are composed of several independent and semi-autonomous agents that collaborate (or compete) to solve a given task; insights on how real social structures emerge in the real world can then be applied to such MAS in order to improve its performance or reduce its costs of operation.

The second approach towards generating computation from social systems is composed of using real social systems, with real people, and leveraging such systems so that humans, and not computers, are responsible for producing reliable computation (AHN; DABBISH, 2008; AHN; LIU; BLUM, 2006). This is the case of *crowdsource*, Wisdom of CrowdsWisdom of Crowds, Social Computing and Social Information Processing systems, all of which leverage the interactions of humans in order to solve problems in ways that are almost algorithmic (BRABHAM, 2008; SUROWIECKI, 2005; PARAMESWARAN; WHINSTON, 2007; LERMAN, 2007). The main idea behind this approach is that there is a great potential to solve complex problems by requiring very little from a great number of people.

This latter approach can be further divided in different perspectives towards how a problem is distributed to a social system and how a result is extracted from it. In one end of the spectrum, *crowdsourcing* is concerned with creating an open call for contributors to solve a problem and once a volunteer takes up the task, he or she will solve it without the help of others. On the other end, Social Information Processing (SIP) uses the interaction of many actors to collaboratively and iteratively create a solution.

All of these social systems have in common the need for social interactions between actors. By combining the set of all actors and their social interactions, we can compose a *social network*. It is clear that such network has a central role in any social system, since it represents and defines how actors interact with each other, by setting specific network properties (such as the network topology).

One could ask how does these network properties relate to the system's performance at solving some task. In this thesis, we report on a number of algorithms, experiments and results that tackle precisely this question. We do so by using a methodology that fits in between the two approaches mentioned above: we propose a model that mimics many aspects of social interactions and the exchange of information in social networks, but that aims not at simulating these processes, but rather at *producing computation in the form of problem-solving*, so that there is an objective definition and measurement of performance.

Hence, as we will argue, this model, named Memetic Network, is close in its nature to e.g. Genetic Algorithms (GA), which mimics a natural process (natural selection and evolution) but aims at solving problems in general by automatic means (without using actual genes or DNA). Likewise, we make use of concepts of social networks to produce algorithms that are able to handle specific tasks in an automatic fashion, without actual social actors being involved. Nonetheless, we argue that the model and its associated algorithms are also useful to provide insights and understandings of real social systems and that its greatest value lies precisely in that.

In order to pursue our objectives, we make use of the concept of *memes* - pieces of information that propagate in a social network by copy - proposed by (DAWKINS, 1976). We also draw several concepts and tools from Network Theory (or the "Science of Networks") (NEWMAN; BARABÁSI; WATTS, 2006), both in the form of specific network topologies and properties to apply in our algorithms and in the form of analysis tools that are used to extract the results.

The act of solving problems is framed in this thesis as a *search problem*, where the task is reduced to finding a particular desirable state (a solution) in a possibly very large state space. Framing the problem as such allows for a very objective measure of a system's performance, which is one of the main advantages of the proposed model. The proposed algorithms are therefore search algorithms which, as we will show in Chapter 8, can be very competitive with traditional search techniques (such as Hill-Climbing, GA).

This work is organized as follows.

Chapter 2 summarizes aspects of the area of search and optimization, defining what we consider to be a search and optimization task and presenting and discussing several search algorithms commonly used in the literature, from Hill-Climbing algorithms to Genetic Algorithms.

Chapter 3 presents a revision on basic concepts, terms and previous results on the area of Graph, Network Theory and Social Computing. The basic network models used in this thesis are shown, along with the most relevant network properties.

Chapter 4 defines our general and specific goals and presents the methodology used in this thesis. We detail how the model is built, how the experiments are conducted and analyzed and other relevant details.

Chapter 5 presents the Memetic Networks model, a basic framework that allows experimenting with the influence of network properties in search. We introduce the concept of *memes* and how it is used to compose our model, along with several alternatives to aggregate information from multiple vertices in a network. We also present initial experiments with basic instantiations of the model, in order to better discuss its various parts.

Chapter 6 report on experiments with networks that are static in nature, so that the topology is fixed *a priori*. We focus on the ubiquitous Small-World and Scale-Free classes of networks, showing how different networks from these classes affect optimization tasks.

Chapter 7 deals with dynamic networks, where we let the network adapt to the problem at hand and reconfigure itself on the fly. We show how properties of this type of network influence the quality of the emerging solutions.

Chapter 8 compares the proposed algorithms with each other and with traditional search techniques, showing that they can be competitive with other algorithms and potentially useful as a general-purpose optimization algorithm.

Chapter 9 concludes this thesis, discussing the results and proposing future lines of research.

Much of this work has appeared in a number of papers. The results present in Chapter 5 and Chapter 7 include those published in (ARAUJO; LAMB, 2008a,b,c). Some of the results in Chapter 8 have appeared in (ARAUJO; LAMB, 2008d).

The concepts and ideas proposed in this thesis are a greatly motivated by our previous works on synchronization of competitive agents in markets (ARAÚJO; LAMB, 2007; ARAUJO; LAMB, 2009). In these works, each actor in a group compete to maximize her own utility function, while the system is designed so that only a minority of the players can win; hence herd behaviors are always prejudicial and players must be able to differentiate from each other as much as possible.

However, no communication is allowed between actors and considering systems where such communication is possible and some cooperation is allowed was, and still is, a matter for future work; but the pursue of such line of thought lead to the necessity of creating a model of communicating agents, which end up becoming the central idea in this thesis and springing the results we present hereafter.

2 BASICS OF SEARCH TASKS AND ALGORITHMS

In this chapter we introduce the general problem we tackle in this thesis, namely problem-solving through search. We describe how problem-solving can be posed as a search problem and proceed to describe several successful search algorithms often used for this purpose, focusing on the so-called *population-based* search, which uses communicating parallel searches.

2.1 Search Tasks

A problem solving task can be understood as a search for a specific state (or a sequence of states) in a solution space, satisfying the problem's requirements (RUSSELL; NORVIG, 2002). For example, the problem of designing an airplane wing can be reduced to the problem of finding an optimal (or good enough) set of wing's parameters that leads to the desirable performance according to some evaluation methodology. The problem of winning a chess game can be seen as finding a sequence of moves that lead to the desired winning configuration. Each combination of parameters in the wing or possible board configurations can be regarded as a state and the problem becomes searching in a search space for a particular solution that contains the desirable parameters.

What are the requirements to perform a search as a way to solve some problem? Three basic requirements can be devised (MICHALEWICZ; FOGEL, 2004; RUSSELL; NORVIG, 2002; MITCHELL, 1997). The first requirement is a way to provide a representation for possible solutions to the problem. This representation will configure a search space, each point consisting of a candidate solution, and should be defined so as to allow the inclusion of the goal state in this space. This is not always a straightforward task, since the goal state may not be known in advance for some problems (and often is not). If the goal state is not included in the search space, then certainly the solution obtained will be sub-optimal.

The second requirement is a way to enumerate all possible states. That is, we must be able to systematically generate any possible solution allowed by the chosen representation. Moreover, it is interesting to allow for generating a "next" state from a current one, as this allows for visiting one state at a time, in a defined and repeatable sequence.

The third requirement to perform search as problem solving is a way to evaluate each candidate solution in the search space. At the very least, there must be a way to know when a goal state is reached. However, allowing for an evaluation function that ranks higher solutions that are closer to the goal state allows for more efficient searches. Indeed, if all we know is whether a state is the goal state or not, we can hardly do better than evaluate all possible states.

While there are several types of search problems, this thesis focuses on *optimization* problems. An optimization problem can be seen as a search problem where:

- The sequence of states that lead to a solution is not important. This is in opposition to, for example, a chess game, where the sequence of moves that lead to a winning state must be known (i.e. there is little use in knowing how a winning state looks like without knowing how to get there). However, even in this latter case it is possible to understand a sequence of states as a single meta-state (e.g. a sequence of moves in a chess game may be regarded as a single state); this would lead to a much larger search space and consequently to a harder search problem. Nonetheless, while conceptually the two problems are not very different, the distinction is useful in practice.

- Each solution can be evaluated and ranked, so as to have a preference ordering over solutions in the search space. In contrast, consider the SAT problem (GAREY; JOHNSON, 1979), where one desires to find a set of values for boolean variables in a logic statement such that the statement is made true; in this case, either a solution satisfies the statement or it does not - it is not possible to directly rank two “false” solutions[1].

- There may not be a known goal state or its evaluation. In an optimization problem, one may want to find the best solution, but typically there is no way to know whether a solution is the best one (otherwise, the solution would be known in the first place). Hence, the goal is to find the best solution possible given some resource constraint (e.g. time or space). An alternative is to set a threshold over the evaluation function that yields a sufficiently good solution and stop the search once such solution is found. Still, the goal state is not known, only its desired evaluation, but this provides for an additional stop condition for a search algorithm.

2.2 Search Algorithms

For very simple problems, it is possible to perform an exhaustive search - see e.g. (RUSSELL; NORVIG, 2002; SPALL, 2003), where an algorithm enumerates all possible states and evaluates each one, returning the best found (if any). The starting state is not important, since all states will be visited in the end. *Exhaustive search algorithms*

[1]Heuristics are often applied to add more information to the SAT problem, so that it can be treated as an optimization problem. See (MICHALEWICZ; FOGEL, 2004) for examples.

are always complete and optimal. A complete search algorithm always finds a solution if one exists. An optimal search algorithm always returns the optimum solution (i.e. the solution that has the best evaluation among all possible solutions).

Completeness in optimization problems may not seem an issue, since - in principle - every state is a possible solution. However, in constrained optimization a solution may not always be viable, in the sense that it does not satisfy a set of constraints even though it may be allowed by the chosen representation. Hence, an algorithm that is not able to find a viable solution is said to be incomplete. Likewise, if a threshold is set specifying a minimum evaluation value to be met, then an algorithm that is not able[2] to meet this threshold is also incomplete.

As problems become more complex, in the sense that the number of states and the cost of evaluating a state grow, exhaustive search can become computationally intractable - typically time and/or space requirements will surpass any existing hardware specification. In many cases this can happen even for problems with moderate sizes.

Take for example the Traveling Salesman Problem (TSP) (APPLEGATE et al., 2007), consisting of a set of cities with roads with different lengths connecting pairs of cities and where one wants to find the shortest path that visit each city exactly once. Each combination of cities specifying the order that they must be visited is a possible solution for the problem, but one aims at finding the best solution. This problem was proved to be NP-Hard (APPLEGATE et al., 2007; MICHALEWICZ; FOGEL, 2004) and the only way to guarantee an optimal solution in the general case is by enumerating all possible solutions. Small instances of such problem (with a few cities) are tractable in today's computers. However, the number of possible solutions grows as a factorial function of the number of cities, which is worse than exponential growth, quickly making even moderately large instances of the problem intractable.

Hence, in these cases, it is no longer possible to generate and evaluate each possible solution and a careful decision must be made towards choosing which solutions will be evaluated. Moreover, for the general case, search algorithms that use a reasonable amount of resources do not guarantee that the best solution will be found, or that any reasonable solution will be found at all (even if it does exist). At best, it may be possible to reduce arbitrarily the probability of not finding a solution (or the best solution) and/or specify a lower bound for the solution's quality (SPALL, 2003).

Non-exhaustive searches can be of two basic types (RUSSELL; NORVIG, 2002). The first type simply starts an exhaustive search and stops whenever a resource is depleted (e.g. the algorithm runs out of time or space), returning the best solution found so far. This type is said to perform search without additional information, since a search is not guided in any way - it is the same exhaustive search algorithm being used.

The second type of non-exhaustive search tries to use information from the problem to guide the search, thus providing a preference ordering over which states are to be evaluated next. This second type is said to perform search with additional information.

[2]The algorithm is not able to meet the threshold in the sense that the probability of reaching a solution evaluated above the threshold is zero.

Such information is typically provided by a heuristic (GLOVER; KOCHENBERGER, 2003; PEARL, 1984). A heuristic is a rule that provides some knowledge about the problem that can help discarding bad solutions or favoring potentially good ones. For example, an actual heuristic for the TSP takes the form “from the current city, visit the nearest neighbor city first”, which seldom returns the optimum path, but under certain conditions is able to find reasonable solutions with drastically reduced resource requirements (GLOVER et al., 2000).

2.2.1 Local search

A heuristic can use existing knowledge about the problem or use information collected during an ongoing search. The latter is the case of *local search algorithms* (RUSSELL; NORVIG, 2002; GLOVER; KOCHENBERGER, 2003; HART, 1994). These algorithms try to use local information about the solution space itself. Typically, they start from a single state and decide which state will be visited next by using a heuristic. This heuristic may not be problem-specific because it is not about the problem, but rather about the way a search is performed in general.

A well-known local search algorithm is the *hill-climbing search*. This algorithm starts from a randomly chosen state S_0 and proceed by enumerating only the neighbor states of S_0. A state is said to be the neighbor of S_0 if it can be reached from S_0 by making a limited, small number of changes to it (RUSSELL; NORVIG, 2002). Whenever a better state is found in this neighborhood, it becomes the current state and the process is repeated. The heuristic in this case is “search in the direction of better evaluated states”. If the considered state neighborhood is kept small, then this algorithm is very resource efficient, as only a fraction of the solution space will be explored (of course, if the neighborhood includes all possible states, then the algorithm performs an exhaustive search). In this type of algorithm, it is said that solutions are being *exploited* - i.e. information about previously evaluated solutions are being used to guide the search.

Exploiting information from the search space is only useful when there is information to be exploited. Hence, the evaluation function must yield a search space that provide such information. The topography or landscape of a search space is a graphical visualization of the evaluation of each state, plotted for each possible state. Figure 2.1 depicts simple examples of topographies. For an evaluation function to provide useful information for local search, it must typically generate a smooth topography, so that similarly evaluated states are close together in the topography (i.e. the “geographical” location of states is correlated to their evaluation). In an extreme case, the evaluation function assigns random values to each state, hence completely removing any information from the topography.

Several issues may arise when using local search algorithms and different heuristics have been proposed to alleviate such problems. The basic hill-climbing algorithm, and its derivatives, are prone to stagnate the search in local optima. This happens when no neighbor is better than the current solution, leading the search to a halt. However, it is not always the case that such solution is the optimal one. Consider the plots in Fig. 2.1,

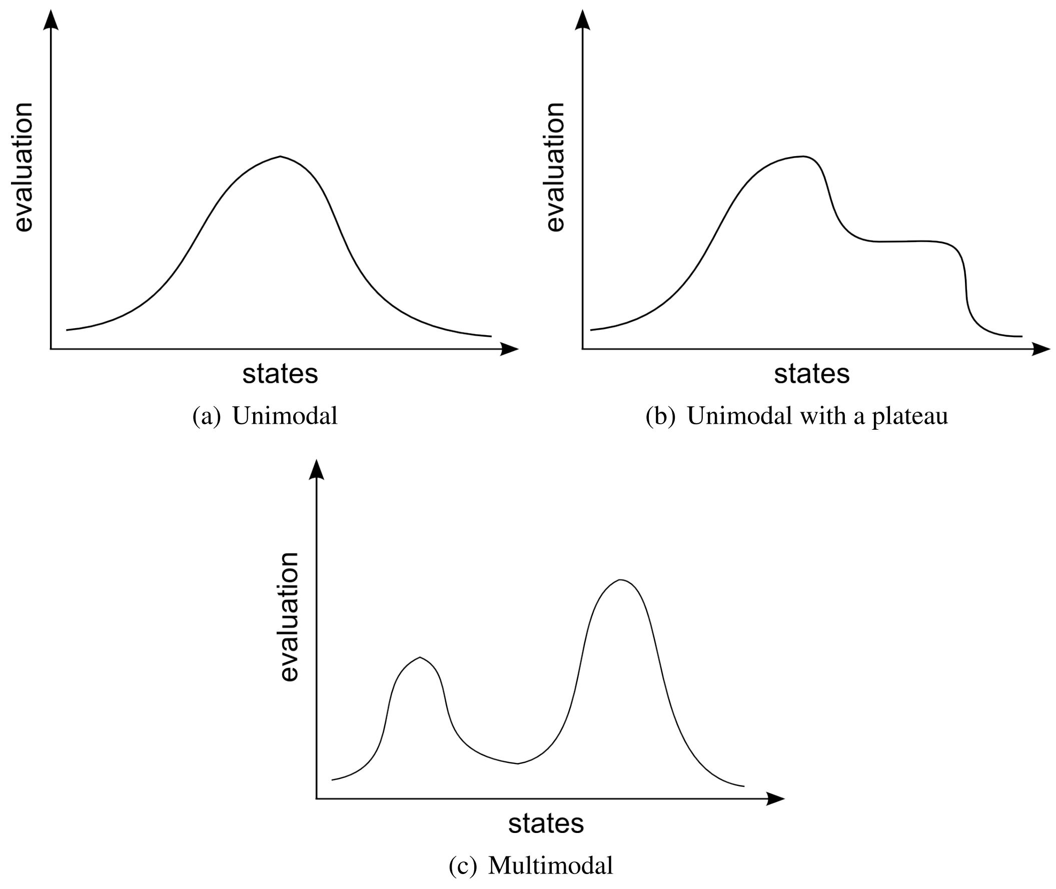


Figure 2.1: Different possible scenarios for optimization problems: (a) only a single optimum; (b) single optimum with a plateau; (c) multiple optima.

where the topographies of fictitious function evaluations are shown. The topography in Fig. 2.1(a) is said to be unimodal, as there is only one optimum. Hill-climbing provides the best results on unimodal problems, because when they do converge, they converge to the best possible solution.

Another instance where hill-climbing performs poorly is depicted in Fig. 2.1(b). In this problem, there is a "plateau" where the evaluation of states do not differ from the evaluation of its neighbors. A hill-climbing search will be unable to continue a search, returning a solution that may not even be a local optima.

The topography depicted in Fig. 2.1(c) is said to be *multimodal*. Multimodal problems (i.e. those that induce multimodal, or rugged, topographies) are characterized by multiple local optima, from which only a few may be global optima (i.e. provide the best possible solutions). A hill-climbing search will stagnate in a local optima and there are no guarantees that such local optima will also be a global one. The algorithm's outcome will depend on where it starts (i.e. the initial state). If it happens to start near a global optima, then it will converge to this global optima, providing the best solution. Otherwise, only a local optima will be returned, which may or may not be adequate for the problem at hand.

2.2.2 Avoiding local optima

In real-world problems, local optima are often present and the solution provided at these points are not adequate (MICHALEWICZ; FOGEL, 2004). A considerable amount of efforts has been put into developing algorithms that are able to avoid or escape local optima. Some noteworthy examples are considered in this section.

Search algorithms that employ some technique to avoid being trapped in local optima are often called *global search algorithms*, even though only local information about the search space is still used.

One of the most well-known search algorithms that try to escape local optima is the *simulated annealing*[3] algorithm (KIRKPATRICK; GELATT; VECCHI, 1983). This algorithm also starts at a single initial state and proceed like a hill-climbing search. Whenever a better state is generated from the current state, this better state becomes the current one. However, if a worse state is generated, the simulated annealing may accept it (i.e. making it the current state) with some probability - the worse the state is, the less likely it is to accept it. This provides a way for the algorithm to search "downhill" and effectively abandon a local optimum or leave a plateau. To allow for convergence, the probability of accepting worse solutions decreases with time, so that eventually no worse solution is accepted and the algorithm converges to some solution.

The idea of the simulated annealing algorithm is to allow for some exploration of the solution space instead of only exploiting the current solution (as is the case of hill-climbing). The algorithm initially explores heavily the space, accepting frequently worse solutions. In later rounds, the algorithms prefer to exploit known solutions, favoring convergence speed.

Another approach towards global search is performing multiple restarts of the hill-climbing algorithm (MUSELLI, 1997; MAGDON-ISMAIL; ATIYA, 2000). This meta-algorithm then returns the best solution found over all runs. By restarting the hill-climbing process at different points in the search space, the algorithm can potentially find multiple local optima, improving the probability of one of them being the global optima. Each run is completely independent and can be executed either in parallel or sequentially, without altering the results.

2.2.3 Population-based search

Running multiple independent instances of a hill-climbing (or similar) algorithm has often been applied in several search task with success (MUSELLI, 1997; MAGDON-ISMAIL; ATIYA, 2000). However, one of the problems with this approach is that there can be an inefficient use of computational resources. This can happen due to two main reasons: resampling of the same solutions and exploration of unpromising regions of the search space. If we allow for random initial states, there is a chance that many hill-climbers will converge to the same local optima, returning identical results. On the other hand, some hill-climbers may get stuck in local optima or plateaus even if other

[3]The term simulated annealing comes from the annealing process of materials, where they are heated and slowly cooled so that large crystals are formed, avoiding defects.

searches have found better solutions, thus wasting resources by exploring solutions that are known to be sub-optimal.

Clearly these shortcomings can be improved by allowing some sort of communication between parallel searches. Several studies have tackled this problem, resulting in different approaches and algorithms. They are usually classified as population-based, indicating the use of a "population" of searches running in parallel and with communication or interaction capabilities.

2.2.3.1 Local Beam-Search

One of the simplest forms of population-based search is local beam-search (RUSSELL; NORVIG, 2002). In this approach, a number of initial states are generated and evaluated. The next step is generating neighbor states for each state. For example, if we start with N states and for each state we generate one neighbor, we have in total $2N$ states. Then, the N best states among all $2N$ states are kept and the remaining discarded. The process is then repeated. This is a case of *best-first search* with limited memory requirements.

In local beam-search, we are effectively guiding our population based on results of previous samples. Hence, it is a form of communication between searches. The effect is that "promising" areas of the search space will receive greater attention (i.e. more computational resources), while less promising areas are abandoned. Hence, visited solutions are being exploited and exploration is reduced.

The term "promising" assumes that there is a strong correlation between the location of solutions in the search space and their evaluation, so that in the neighborhood of good solutions we expect to find other similarly good (and hopefully better) solutions. Once the algorithm finds a good solution, it expects to find other good solutions around it. Therefore, it makes sense to direct searches that were exploring worse solutions to these areas. However, even such smooth topographies can be multimodal and deceptive - the global optima may be surrounded by very bad solutions. Compared to the hill-climbing algorithm, local beam search allows for more exploration, since not only the very best state is kept, but a few worse ones are still allowed to remain in the population.

Local beam-search implements one type of interaction between searches, but only information about the states' evaluation is used for that interaction. Information about the state itself (i.e. its location in the search space) is only indirectly transmitted. Another approach comes from the area of Evolutionary Computation, that includes many successful algorithms such as Genetic Algorithms, Genetic Programming and Evolution Strategies. In these algorithms, information on the states' locations is transferred between searches.

2.2.3.2 Genetic Algorithms

The best and most widely used example of Evolutionary Computation is arguable Genetic Algorithms (GA) (HOLLAND, 1992; FOGEL, 2000; MICHALEWICZ, 1996;

MICHALEWICZ; FOGEL, 2004; MITCHELL, 1997). The term GA has been appropriated and used in many different ways in the literature and is often used to encompass the whole field of evolutionary computation, but we discuss it here as a specific algorithm.

A GA is a population-based search algorithm that is loosely based on the idea of genetic evolution and natural selection and originally proposed by John Holland (HOLLAND, 1992, 1986). In a GA, a number of individuals represent states in the search space, each individual representing a state. Typically, states are represented as binary strings, called *chromosomes*. In a chromosome, there is no distinction of individual parameters of a state, so that if a state is composed of a set of real numbers, these are coded in a binary form and stitched together in one single contiguous chromosome.

The algorithm starts by randomly initializing N individuals (i.e. randomly assigning a state to each individual) and evaluating them using an evaluation function (often called *fitness function*). Then, a new population is created from this population as follows. A selection algorithm picks pairs of individuals from the current population so that the probability of selecting any individual is proportional to its fitness (evaluation) - the more "fit" the individual, the greater the chance it has of being selected. Several selection algorithms have been proposed. The original algorithm was called *roulette wheel selection*, which assigns a larger "share" of selection probability to individuals with higher fitness.

With probability $(1 - p_c)$, the selected individuals are put in the new population without changes. With probability p_c, the individuals go through a crossover operation. The most common type of crossover operation is the the single-point crossover, where a random bit in the chromosome is selected and the two halves of the chromosomes of each individual are exchanged, as shown in Figure 2.2, forming two new individuals which are put into the new population.

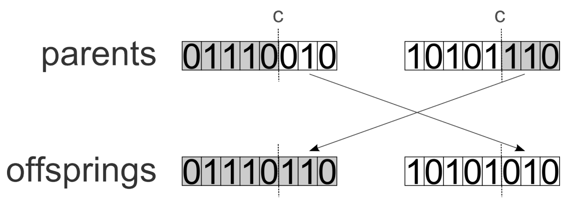


Figure 2.2: Example of a crossover being performed over 8 bit strings. Two offsprings are created by swapping bits from two parents at point c.

This selection-crossover process is repeated until the new population is filled with N individuals. The last step of the algorithm applies a *mutation* to each individual in the new population: each bit in each individual is flipped with probability p_m. Then, this new population becomes the current population, the individuals are evaluated and the whole process is repeated.

To some extent, a GA resembles a local beam-search: the selection algorithm discards most of the worse-performing individuals, favoring better solutions. However, this selection is now stochastic, hence even the worse evaluated solution has some (low) probability of being selected (just like simulated annealing). This reduces the pressure on exploiting only the very best solutions. Nonetheless, the main difference is the crossover operation, which allows for searches to effectively and directly communicate information about their states with each other. Russell and Norvig (RUSSELL; NORVIG, 2002) note that "the primary advantage, if any, of genetic algorithms comes from the crossover operation". Such recombination is used to generate "neighbor" solutions, but using information from two solutions instead of just one. Of course, this is inspired in sexual reproduction in nature and justified by the belief that it provides an evolutive advantage over asexual reproduction. Under this light, local beam-search can be understood as an asexual GA with a very strong selection algorithm.

There is a long and on-going debate in the literature about how exactly crossovers can really help a search (and whether it helps at all), but many, if not most, researchers in the area (HOLLAND, 1992; FOGEL, 2000; MICHALEWICZ; FOGEL, 2004; GOLDBERG, 1989) believe that this operation does provide benefits. There are evidences that the crossover operation works by modifying the granularity of the search, which is possible by combining large chunks of useful information. This theory was formalized by the notion of *schemes* as the supporting representation.

Several alternatives to each element of a Genetic Algorithm have been proposed. For instance, it is now common to use any representation that is more natural to the problem at hand, in contrast to using only binary strings (MICHALEWICZ, 1996); it has been shown that the representation should not influence the overall search performance (FOGEL, 2000). Different selection and crossover methods have been proposed, including the recombination of more than two individuals and spatially structured selections (we will return to these particular cases later in this thesis).

2.2.3.3 Ant Systems

Another interesting example of population-based search is the so-called Ant Systems (AS) (DORIGO; COLORNI, 1996) . These algorithms mimic a colony of ants (or other social insect) in order to transfer information between searches and that have been successfully applied to combinatorial optimization, such as the Traveling Salesman Problem (DORIGO; GAMBARDELLA, 1997). The inspiration is justified due to observations that ant colonies are able to find the shortest route to a food source without centralized control. Each search is thus seen as an ant foraging for good solutions

For example, in an Ant Colony Optimization (ACO) (DORIGO; BIRATTARI; STÜTZLE, 2006) algorithm applied to the TSP, a population of ants try to find a Hamiltonian cycle - a cycle that goes through each city exactly once - with minimum cost in a graph. Initially, ants wander through the graph randomly. Each ant that completes a full cycle leave a "pheromone trail" over the found path. This trail is a stigmergic[4] indication of the quality of the found path, and the amount of pheromone laid on the path is proportional to the quality of the solution. Other ants are probabilistically attracted to the

[4]Stigmergy is a mechanism of indirect coordination between agents, which exchange information by modifying some shared environment (BONABEAU; DORIGO; THERAULAZ, 1999).

pheromone, and stronger pheromone trails end up attracting more ants. Additionally, pheromone "evaporates" with time, so that a constant flow of ants is necessary to keep a trail's pheromone level strong.

Searches communicate with each other through pheromones, probabilistically influencing each other towards certain areas of the search space. In effect, a single ant is influenced by all ants that have performed searches before it. Hence, a new solution is formed by combining several previous solutions, with the addition of some random decisions due to the probabilistic attraction towards pheromone trails (which guarantees some exploration). The result is that the surroundings of good paths in the search space are explored more often (i.e. they are exploited), while bad paths receive less attention.

2.2.3.4 Other population-based algorithms

Several other population-based search algorithms have gained attention in the last few years. Particle Swarm Optimization (PSO) perform a search with a population of "particles" that constantly move on the search space and have its direction and velocity determined by other particles (EBERHART; KENNEDY, 1995; OLORUNDA; ENGELBRECHT, 2008). All particles may influence each other, or a specific structure may be set to restrict influence to a particular neighborhood. A particle knows its own position (in the search space), the position of the best particle in its neighborhood, the best particle in the whole system and the best particle it has seen during a run and adjust its direction and velocity by combining these information.

Artificial Immune Systems (AIS) is a more recent class of search algorithms, inspired by the immune system of vertebrates (DASGUPTA, 1999; FARMER; PACKARD; PERELSON, 1986). It is presented in several forms, each adapted to some specific task. Most of these forms are based on the selection of specific structures (antibodies) based on their "affinity" to some problem. Hence, a process of natural selection is also in place. AIS algorithms are also population-based, even though the selection algorithms used in AIS are typically more involved than that of evolutionary algorithms.

2.3 Information Exchange

In population-based algorithms, information collected during a search can influence and guide how it is conducted. Since multiple searches are performed, either sequentially or in parallel, they can influence each other. We can understand this influence as a form of communication. Each search communicates some information to other searches, regarding the value of the region being explored or their raw position in the search space.

This communication may be very direct, as is the case of Genetic Algorithms, where pieces of solutions are effectively exchanged. On the other hand, in Local Beam Search, this communication is very indirect, as searches do not exchange parts of the solution, but rather "broadcast" their evaluation. Table 2.1 summarizes the discussion so far, showing how communication takes place in each mentioned population-based algorithm.

Search algorithm	Individual search	Communication method
Local Beam Search	state	selection by evaluation
Genetic Algorithms	chromosome	crossover, selection by evaluation
Ant Systems	ant	pheromone trail
PSO	particle	position and velocity

Table 2.1: Short summary of population-based search algorithms.

For all non-exhaustive search algorithms, there is a trade-off between exploiting known solutions and exploring unvisited solutions. Random sampling of the search space provides maximum exploration, as all solutions have the same probability of being selected at any time. On the other hand, a steepest ascent hill-climber will exploit the initial solution by only exploring its best neighbor.

When using a population-based search, increasing exploration means that less resources will be allocated for promising areas of the search space (i.e. for areas where good solutions have already been found). Hence, if indeed better solutions are to be found near good solutions, convergence towards these better solutions will be slower.

Increasing exploitation of found solutions have the reverse effect and improves convergence speed towards better solutions that can be found near the good solutions already found. However, since less resources are dedicated for exploration of unknown areas of the search space, there is a higher probability that the algorithm will miss the global optimum or better local optima possibly located at these unvisited areas. The trade-off is, then, between faster convergence speed towards local optima and lower probability of missing global optima.

All population-based algorithms allow for some control of exploitation and exploration. In Evolutionary Algorithms, increasing selection pressure increases exploitation. This control can be attained, for example, by using tournament selection. Tournament selection randomly selects K individuals in the population and selects the best one to compose the next generation. By tuning K, one can control how much exploitation is being performed - if the group is of the same size as the population, exploitation is at a maximum (the best individual is always selected, limiting exploration to its surroundings only).

In Ant Systems, exploitation is controlled mostly by the pheromone trail - faster "evaporation" leads to higher exploration; more attraction towards pheromone leads to higher exploitation. In Particle Swarm Optimization, the trade-off can be controlled by setting the weight given to how much a particle is influenced by others.

These algorithms provide such control because different problems require different trade-offs. A problem whose topography is highly multimodal will require more exploration, while a less rugged topography may allow for more exploitation to speed convergence up. Of course, if time is not an issue, we can always allow for as much exploration as possible or, even better, simply perform an exhaustive search. Since time is very often an issue, we need a balance between exploration and exploitation when solving the problem, so as to guarantee a good enough solution in a reasonable amount of time.

We can thus conclude that knowing how to control the trade-off between exploitation and exploration is crucial for any search algorithm and, hence, for problem-solving techniques. In order to provide solutions with a high quality in a limited amount of time, one must know how "hard" the problem is and adjust the algorithm accordingly. This is important because in this thesis we examine the relationship between network properties and the ability of the algorithms described in the next chapter to perform well in diverse tasks.

3 GRAPHS, NETWORKS AND SOCIAL COMPUTING

3.1 Graph Theory

A network is a general term used to specify any set of objects with relations, or interconnections, between them (NEWMAN; BARABÁSI; WATTS, 2006; BARABÁSI, 2003). These objects may not be physical objects, but can also be concepts or abstractions. Connections are representations of any arbitrary kind of relation between two objects. Hence, the term network can be applied to a variety of systems. For example, a computer network is a collection of computers that are able to communicate with each other through physical connections. On the other hand, a *social network* is a collection of people with relations that can be more intangible, such as friendship and trust bonds.

Graphs are used as abstract representations of networks. Formally, a graph is usually represented by an ordered pair $G = \langle V, E \rangle$, where V is a set of vertices, or nodes, and $E \in V \times V$, a set of edges. A vertex v is said to be adjacent to another vertex v if $(u, v) \in E$ - i.e. there is a connection from v to u. The neighborhood of a vertex u is the set of all vertices adjacent to u.

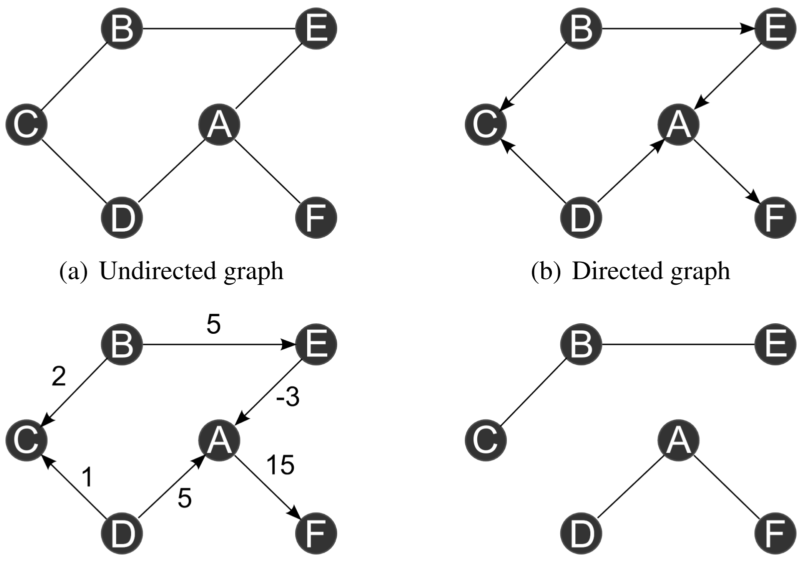
Several types of graphs can be devised based on properties of its elements. The most relevant to this study are very briefly detailed below. For the explanations that follow, u and v are vertices in a graph G. Figure 3.1 depicts examples of graphs. The definitions below follow (NEWMAN; BARABÁSI; WATTS, 2006).

Directed and undirected graphs An undirected graph has reflexive relations, so that $(u, v) \in E \Rightarrow (v, u) \in E$. A directed graph, on the other hand, has no such restriction. Directed graphs are often called digraphs.

Weighted and unweighted graphs A weighted (or valued) graph associates a number or symbol to each edge. This allows for representing additional information about the connections, such as costs or lengths.

Connected and disconnected graphs An undirected graph is said to be connected if there is a path from any vertex u to any other vertex v. Otherwise, the graph is said disconnected. For directed graphs, if the statement holds then the graph is said to be strongly connected. It is weakly connected if it is connected when edges' directions are discarded.

Cyclic and Acyclic graphs A graph is said to be acyclic if for any vertex v there is no path that starts and ends on v. If any such cycle is present, then the graph is said cyclic.



(a) Undirected graph (b) Directed graph

(c) Weighted graph (d) Disconnected graph

Figure 3.1: Examples of different types of graphs.

A number of properties are useful to describe graphs and its components. We recall those relevant to this thesis below.

Degree In an undirected graph, the node's degree is the number of edges that the node is part of, i.e. the number of nodes it is connected to. For directed graph, two types of degrees can be defined. The in-degree of a node is the number of incoming connections (number of nodes that "point" to the referred node). The out-degree is the number of outgoing connections (number of nodes that the referred node "points" to). For example, in the graph depicted in Fig. 3.1(a), the node A has degree 3, while in the graph in Fig. 3.1(b), node A has an in-degree of 2 and out-degree of 1.

Path A path exists between two nodes, u and v, if it is possible to reach u from v through a succession of edges. If a path exists between these nodes, then it is said that u is reachable from v. In undirected graphs, paths are always symmetric, but that may not be the case for directed graphs. The s between two nodes is the number of nodes that must be traversed from a starting node to reach a target node.

Diameter The diameter of a graph is the longest shortest path between any two nodes. In cyclic graphs, cycles must be excluded to calculate the diameter (otherwise,the diameter is infinite).

Hamiltonian cycle A Hamiltonian cycle, or Hamiltonian circuit, is a cycle that visits every node in the graph exactly once. That is, starting from any node, we return to the same node by creating a path that visits each node once. No particular order is imposed on the order of the visits.

Graphs are typically used to represent networks, and tools from Graph Theory can be used to analyze such graphs and, thus, the networks they represent. Nonetheless, Graph Theory may fall short when used to analyze certain types of networks, particularly massive networks that are complex and dynamic.

3.2 The Science of Networks

The study of networks has gone through a major paradigm shift over the last few years. The previous dominating paradigm viewed networks as mostly static structures, representing static models. The pioneer work of the mathematician Leonhard Euler in Graph Theory illustrates this line of thought. In such work, Euler was posed with a simple problem, known as the Seven Bridges of Königsberg: given a set of islands in the city of Königsberg and a set of bridges connecting these island, is it possible to perform a complete walk such as every bridge is crossed exactly once? Euler introduced the graph concepts to model the problem and proved that the task for the city of Königsberg was an impossible one. Euler proved a theorem stating that for any graph modeling the defined problem, if the graph had a specific property, then the task was impossible. This property stated that there could be at most two vertices (i.e. islands) with an odd number of bridges connected to it.

The problem, and its solution, is obviously static in nature, as neither bridges nor islands would change, and the graph used as a model was custom-tailored to the problem at hand. Moreover, the problem statement (and its solution) says nothing about how the actual bridges and islands are used (e.g. by the population). Graph Theory is mostly concerned with graphs as pure structures (WATTS, 2003) that are static in time. As dynamic aspects are introduced, either by allowing the network itself to change or by having dynamic processes making use of the network, Graph Theory fails to provide the necessary tools to analyze such aspects.

The so-called *new science of networks* was devised as a new research area with a research agenda aimed at tackling the shortcomings of Graph Theory. While it can be seen simply as an extension of Graph Theory, it has been argued (NEWMAN; BARABÁSI; WATTS, 2006) that the agenda of this new research area is different enough from Graph Theory to make it reasonable to call it by a different name. This area evolved from the need to understand real complex network structures, that are often massive in size, without applying too much abstractions, which could hide the complexities that make the problem interesting in the first place. Several distinguishing items in this research agenda can be identified, according to (NEWMAN; BARABÁSI; WATTS, 2006):

- *Focus on modeling real-world networks*. While most previous studies on graphs focused on artificial constructs that were amenable for analytical analysis, the New Science of Networks is mostly concerned with providing the tools and means to tackle real-world networks, with as little abstractions as possible. This include social networks, biological networks, citation networks, transportation networks and so on.

- *Networks are assumed to be evolving structures.* Graph theory is mostly concerned with networks that are static in nature, or representing a snapshot in time of a dynamic network. The New Science of Networks, on the other hand, recognizes that most real-world networks evolve over time. For example, the network composed of all websites and the hyperlinks between them changes every second, with new websites entering and leaving the network and new hyperlinks being created or deleted. Hence, a major area of research focus on understanding how structure at a global scale emerges from local interactions.

- *Networks can be understood as dynamic systems.* The traditional approach towards modeling using networks oversimplified the relationship between what happens over a network and its structural properties. The New Science of Networks, in contrast, is much concerned with understanding networks as part of dynamic systems, in which vertices are entities that are tightly coupled by their interconnections. Hence, dynamic processes may take place over a network and be influenced by its topology, but also the reverse is true and the topology may be modified by the interaction of the vertices.

In practice, one of the main distinguishing differences of the new science of networks is its statistical approach towards networks. Instead of defining a network by its structural properties, this approach is interested in its statistical properties, which emerge only from a macro-view of the network as a whole. Such properties are then used to create classes of networks. A major line of research in this area is concerned with classifying real-world networks and creating theoretical generative models that induce the same properties.

This new Science of Networks spawned several modern studies on all types of networks, from biological networks to computer networks. The tools provided by this area are central to understand modern issues such as disease epidemics (MOORE; NEWMAN, 2000), information cascading (LESKOVEC; ADAMIC; HUBERMAN, 2006), web search (ADAMIC; ADAR, 2005), computer network robustness to attacks (ALBERT; JEONG; BARABÁSI, 2000), possible outcomes of trading networks (KLEINBERG; TARDOS, 2008) and many others. This new Science of Networks has been mostly useful for Artificial Intelligence, in different sub-areas, such as semantic analysis, natural language processing and collaborative web search (MITCHELL, 2006; MENCZER; WU; AKAVIPAT, 2008; BERNERS-LEE; KAGAL, 2008; RADEV; MIHALCEA, 2008).

There are several statistical properties that are used to analyze and classify a network and that takes into account its evolving and dynamic properties. Three of them are central to the discussion in this thesis, which are described below, following (WATTS, 1999).

Degree distribution The degree distribution is the histogram of degrees for each node in the graph. It measures how degrees are distributed among nodes. A regular network is one where all nodes have the same degree (thus, the histogram is flat). A random network, on the other hand, is characterized by a Poisson distribution.

Characteristic Path Length The Characteristic Path Length (L) of a network G is the median of the means of the shortest path lengths between all pairs of vertices. Hence, it can be found by first calculating, for each vertex $v \in V(G)$, the average over all shortest path lengths to all other vertices; then, we calculate the median over all $|V(G)|$ means, yielding L. This represent how "close" vertices are from each other - the lower L is, the shortest are the average distance between vertices.

Clustering Coefficient Let C_v be the *clustering coefficient* of vertex v and Γ_v be the set of adjacent vertices to v and $k_v = |\Gamma_v|$. Then, let:

$$C_v = \frac{|E(\Gamma_v)|}{\binom{k_v}{2}}$$

where $|E(\Gamma_v)|$ is the number of edges between vertices in Γ_v and $\binom{k_v}{2}$ is the total number of possible edges in Γ_v. Hence, C_v measures for a single vertex the extent to which vertices adjacent to v are also adjacent to each other. The clustering coefficient for an entire graph G is then defined as C_v averaged over all $v \in V(G)$.

3.3 Small-World Networks

The class of Small-World Networks was first proposed by Duncan Watts (WATTS; STROGATZ, 1998; WATTS, 1999, 2003), in an attempt to provide an explanation for the so-called Small-World Phenomena. This phenomena states that every person in the world is connected to every other person by a very short path of intermediate friends and acquaintances. The phenomena became first known in the scientific community due to a series of pioneering experiments conducted by Stanley Milgram (MILGRAM, 1967)[1]. In his most well-known experiment, Milgram sent letters to several random people; these letters roughly stated that he was trying to reach a final specific individual in a US city and that the recipient should forward the letter to whomever they thought could bring the letter closer to the final individual (or deliver to the individual himself, if the recipient knew the person). Before forwarding the letter, however, the recipient was asked to add his or her name to the end of the letter, so that Milgram could trace the path

[1]It then became known for the general public mostly due to the popular website *The Oracle of Bacon*, which proposed that any actor in the world has a path to the actor Kevin Bacon in a network where an edge connects two actors if they starred together in a movie. See http://oracleofbacon.org.

to the destination. Milgram showed that, for the letters that did reach the destination, the average path length was between 5.5 and 6.0, which later lead to the popularization of the term “six degrees of separation”.

Even though Milgram’s experiments were subject to criticism, mostly due to the fact that it did not account for letters that never reached the destination (KLEINFELD, 2002), other experiments showed similar results. The puzzlement over this problem comes from the fact that such short path lengths are expected for random networks, but the social network connecting people is hardly random - e.g. there is certainly a higher probability of knowing a neighbor than knowing someone in a far country. Moreover, the probability of two people with a common friend knowing each other is much higher than the probability for two random people. Social networks have some inherent structure. However, if we do accept that social networks are structured (and thus highly clustered), like a geographical grid for example, in order to account for such non-randomness, then the short path lengths should not exist.

Watts developed a model that accounted for both the apparent structure and short path lengths of social networks (even though the origin of the model is related to the study of synchronization of cricket chirps (WATTS, 2003; STROGATZ, 2003)). This model is known as the Small-World Network (SWN) model. The central idea is to start with a very regular and clustered network and then add a few shortcuts between random nodes. Such shortcuts drastically reduce the otherwise long path lengths of the otherwise structured network.

In its most common form, the SWN model starts with a simple regular ring network (a 1-lattice network) with N nodes and each node is connected to only a few (K) immediate neighbors. Then, each edge in the network, with probability β, is reassigned to a random node. More precisely, the algorithm to perform such rewires is described in (WATTS; STROGATZ, 1998) as follows:

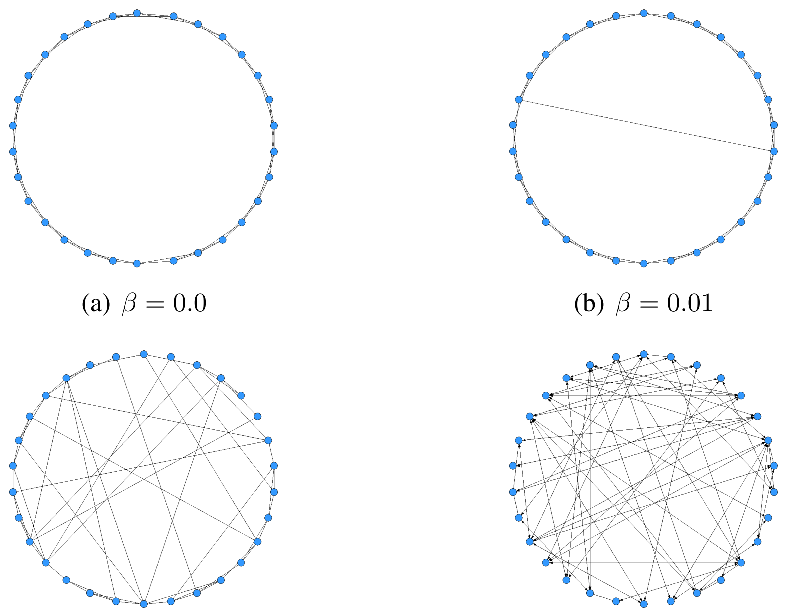
1. Each vertex i is chosen in turn, along with the edge that connects it to its nearest clockwise neighbor $(i, i+1)$.

2. A uniformly random real number r is generated. If $r \geq \beta$, then the edge $(i, i+1)$ is unaltered. If $r < \beta$, then $(i, i+1)$ is deleted and rewired such that i is connected to another vertex j, which is chosen uniformly at random from the entire graph (excluding self-connections and repeated connections).

3. When all vertices have been considered once, the procedure is repeated for edges that connect each vertex to its next-nearest-neighbor (that is, $i+2$), and so on. In total $k/2$ such rounds are completed, until all edges in the graph have been considered for rewiring exactly once.

The above algorithm describes the *SWN generative model*, which yields networks that possess characteristics of SWNs for a range of (relatively low) values of β. In particular, such networks are characterized by a characteristic path length that grows in $O(logN)$, i.e. logarithmically with the number of nodes, while presenting a high clustering coefficient. In contrast, a random network have short path lengths, but also a very low clustering coefficient; on the other hand, a regular ring network is highly clustered, but have a long characteristic path length (that grows in $O(N)$). This is the reason why SWNs are often described as being "between order and randomness" (WATTS, 1999). Indeed, β controls how much randomness is introduced into the network.

Figure 3.2 shows different networks generated using the SWN generative model and different values of β. For $\beta = 0.0$, the ring network is unchanged and remains perfectly structured and regular. For $\beta = 1.0$, all edges are rewired and the network becomes completely random. Intermediate values of β generate networks that are SWNs. Figure 3.3 shows how characteristic path length (L) and clustering coefficient (C) scales with β.



(a) $\beta = 0.0$ (b) $\beta = 0.01$

(c) $\beta = 0.3$ (d) $\beta = 1.0$

Figure 3.2: Examples of networks with different rewiring probabilities. The starting topology is a regular ring network with neighborhood size of four.

Small-World Networks were found to be quite ubiquitous and it is conjectured that most naturally occurring, as well many men-made, networks have the small-world property (WATTS, 1999). For instance, electric power grid networks, protein interaction networks, the neural network of the *C. Elegans* worm, social influence networks, e-mail exchange networks, scientific paper citation networks, all have characteristics of a SWN. This model became a more realistic alternative to model systems which otherwise would be modeled by either structured or random networks (WATTS, 1999, 2003; NEWMAN; BARABÁSI; WATTS, 2006).

SWN are not only distinguished by their topology, but also by the effects such topology has on dynamic processes taking place on the network. A disease that spreads through a SWN will spread faster than a structured network due to its short characteristic path length (KEELING, 1999; KUPERMAN; ABRAMSON, 2001). Global computations in celullar automata were also found to be possible by allowing cells to

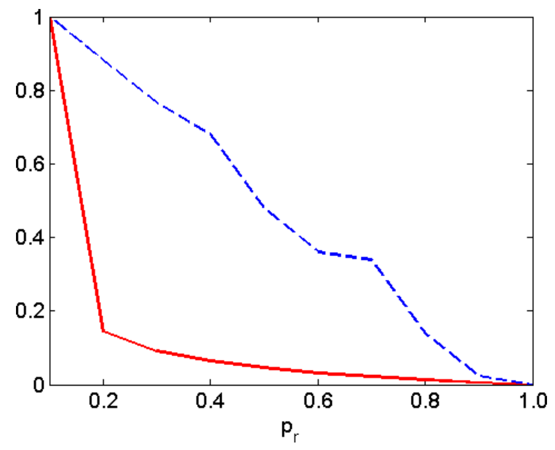


Figure 3.3: Characteristic Path Length (solid line) and Clustering Coefficient (dashed line) for different values of rewiring probabilities (p_r), starting with a regular ring network. Replicated results after experiments originally performed in (WATTS; STROGATZ, 1998).

interact through a SWN instead of the more traditional grid network, and then manipulating the network to improve convergence time(WATTS, 1999). Agent cooperation in artificial societies also emerges faster if interactions are performed through a SWN (OLFATI-SABER; FAX; MURRAY, 2007).

3.4 Scale-Free Networks

Another class of networks recently discovered is composed of Scale-Free Networks. This class was first proposed by (BARABÁSI; ALBERT, 1999) as a way to describe the growth of a network formed by hyperlinks between websites. This network is composed by websites and each time a new one is created, it creates hyperlinks to a number of existing websites. Typically, a researcher would use a random network to model such network: when a new node enters the network, it creates connections to nodes chosen uniformly at random. Hence, any currently existing node has the same probability of receiving the connections of the entering node.

However, Barabási observed that on the World Wide Web, some websites are extremely popular, attracting a very large number of connections, while most failed to attract more than a few links. Such highly connected nodes, named *hubs*, should not exist if the network was indeed growing randomly. The degree distribution of a random network follows a Poisson distribution, which presents a typical (average) degree. Lower or higher degrees are increasingly unlikely - a node with several standard deviations from the average, as hubs are, has essentially zero probability of existing. The degree distribution found on website networks, however, displayed a strikingly different behavior (Fig. 3.4): most nodes have a very low degree, while a few have extremely high degrees. This distribution follows a power-law distribution, in that the probability $P(k)$ that a node will have degree k is defined by: $P(k) \sim k^{-\gamma}$

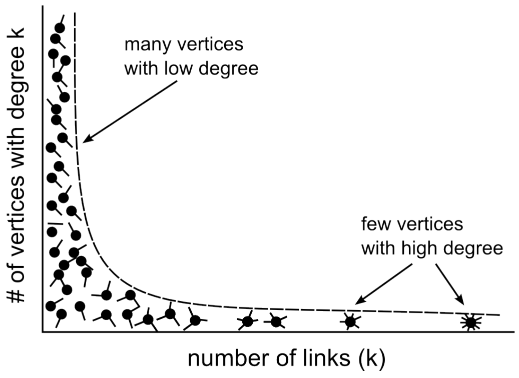


Figure 3.4: A stylized power-law distribution, reproduced from (BARABÁSI, 2003).

For the World Wide Web, an estimate for γ is of $\gamma = 2.1 \pm 0.1$. This means that a node having twice as many connections as another node is approximately twice more likely to attract a new link, rendering a version of the popular phrase "the rich gets richer". Such distribution is clearly not predicted by the random network model, or even by the Small-World network model. This distribution allows the existence of hubs due to its "fat tail" - a slow decaying probability to the right (much slower than a Poisson distribution).

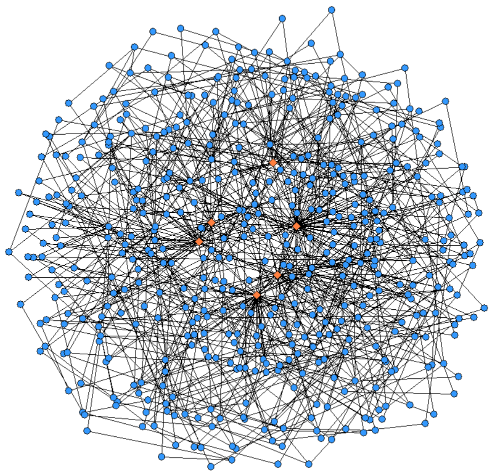
Barabási and Albert proposed a generative model for networks that display the power-law distribution that characterizes scale-free networks[2]. The algorithm proposed introduces the concept of *preferential attachment*, in which nodes connect preferentially to highly connected nodes. This generative model, known as Barabási-Albert model, executes the following basic steps:

1. Start with a small number (m_0) of pre-existing vertices;
2. At every time step, a new vertex is introduced with $m \leq m_0$ edges that link the new vertex to m existing different vertices;
3. The probability $P(v)$ of an existing vertex v receiving a link from the new vertex is proportional to its degree k_v, such that:

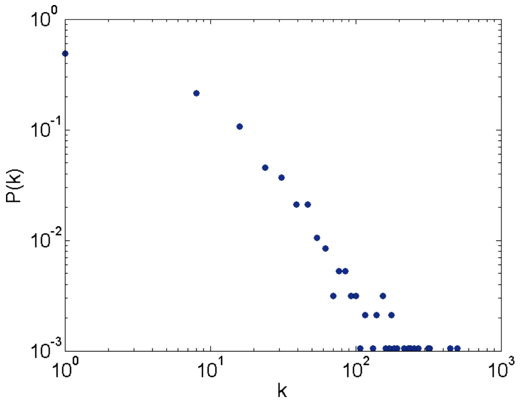
$$P(v) = \frac{k_v}{\sum_{u \in V} k_u}$$

After running the above steps a few times, the emergent network will display the power-law degree distribution with $\gamma = 2.9 \pm 0.1$. A Scale-Free network is often also a Small-World network; the reason for that is the presence of hubs which, by attracting many connections, end up serving as shortcut between many nodes. Therefore, characteristic path lengths are typically very short in these networks.

[2]The term *scale-free* is a reference to the fact that such networks lack "scale", in the sense that many nodes have low degrees and a few have high degrees, rendering the average between these non-informative (WILLINGER; ALDERSON; DOYLE, 2009).



(a) A Scale-Free Network.



(b) Degree distribution function

Figure 3.5: A Scale-Free Network with 500 nodes generated by the Barabási-Albert generational model. The resulting network is pictured in (a) and the corresponding degree distribution function is shown in (b).

This preferential attachment mechanism, and the resulting scale-free network, are often found in real networks. Both the network formed by hyperlinks between websites (BARABÁSI; ALBERT, 1999) and the underlying infrastructural network that compose the Internet (ALBERT; JEONG; BARABÁSI, 2000) are scale-free. The network of citations in scientific papers is also scale-free, having a majority of papers that are seldom cited and a few papers cited extremely often. The collaboration network of movie actors is also scale-free, with a few actors prominently being cast for movies, while a vast majority play a very few roles in their careers.

Framing a network as a Scale-Free network is also important to make predictions on its properties and to take action to control the dynamics on the network. For instance, the knowledge that the network formed by sexual partners is scale-free (LILIJEROS et al., 2001; SCHNEEBERGER et al., 2004) allows for targeting specific individuals (the hubs) for testing or treating for sexually transmitted diseases. Another example is the robustness of the Internet to attacks by hackers: if the infrastructural Internet is scale-free (ALBERT; JEONG; BARABÁSI, 2000) (a fact that is still disputed (WILLINGER; ALDERSON; DOYLE, 2009)), it means that it is very resilient to random attacks, but very vulnerable to targeted attacks; that is, if the hubs are specifically targeted, then the Internet can become essentially disconnected by disabling very few nodes, while random attacks would very rarely hit one of these hubs (because there are many more non-critical, low degree, nodes).

3.5 Social Systems and Computing

Consider a group of people trying to solve some difficult problem - for instance, how to predict the stock price of a company in a stock exchange market. Each person can, and usually does, have a hypothesis on how to predict the price and applies this hypothesis with some degree of success. But these people are not independent, as they write books about their hypotheses, read books from others or simply exchange ideas during lunch. Each of these interaction transfer some information from one person to another, which ultimately have an effect on the hypotheses each one use. Even without the participation of any other person, a single individual would be able to create such hypotheses, but how does the interaction patterns inside a group affect the quality of such hypotheses?

If we, as observers, consider the above mentioned group as a system that is trying to come up with a correct hypothesis to the problem at hand, then we should ask whether we should allow more or less communication between the participants, so that we could improve how effective the system will be. This is a decision that managers often have to make in business organizations: should employees be able to communicate more (say, by grouping them in open spaces, sharing desks etc.) or should communication be constrained (e.g. by putting each employee in a separate closed office), so that the organization productivity is improved?

This idea of seeing a group of people as a system trying to solve a problem is gaining momentum in the form of *crowdsource*. To crowdsource a problem is to delegate its solution to an undefined, large group of people (a crowd), in the form of an open call, in such a way that no single individual has the responsibility of solving the problem but any of these individuals may step up and propose a solution (BRABHAM, 2008). Given adequate incentives for individuals to want to solve problems, this creates a problem-solving system that can be quite different from conventional employee-based systems and very effective in generating solutions for certain problems (HOFFMANN, 2009).

Many successful examples of crowdsourcing exists. A recent example is Amazon's[3] Mechanical Turk[4], which is often referred to as an "*artificial* artificial intelligence". Amazon makes available a web-based system where anyone can post problems to be solved. The poster set a price for the problem and a detailed description of what is to be done and how a solution quality is to be evaluated (whether it is acceptable or not). This problem then becomes visible to all users of the service, so that any user may decide to solve it, receiving the stipulated value in the end. The poster does not know beforehand who may solve it or whether it will be solved at all.

For an example of such system in action, consider *Amazon Remembers* service for the Apple's[5] iPhone. This service allow customers to send a photo of an item to Amazon and it will then return a link to that item (or a similar item) in Amazon's online catalog. Instead of using complex algorithms to process the image, extract its components, identify the item and search for it on the database, Amazon chose to crowdsource the problem. When an image is sent by a customer, it is forwarded to the Mechanical Turk as a problem, stating something like "find an item in Amazon's website that is similar to the item in the photo" and setting a value for a solution (often a few U.S.

[3]http://www.amazon.com
[4]http://www.mturk.com
[5]Apple Computers Inc.

dollar cents). Any user may then simply browse Amazon's website to try and find the product and post the link back to the customer. With a high enough incentive (i.e. price) and enough users, the system replies very quickly and with a high quality to such simple requests.

A distinguishing characteristic of the Mechanical Turk, and most crowdsourcing systems, is that there is no collaboration between users to solve a problem. Once a problem is taken by a user, it is not available anymore for other users, so that each problem is assigned uniquely to a single user[6].

A different crowd-based problem-solving method is known as *wisdom of crowds* (SUROWIECKI, 2005), where a group of people is responsible for finding a solution to a problem. In such setting, each individual in the crowd still works alone (i.e. they are independent of each other) but their solutions are aggregated so as to compose one final solution. In a very simple example, a group undertake the task of counting the number of balls inside a jar. Each individual propose an independent solution and then all solutions are averaged to compose the final answer. If each individual answer is kept independent of each other and there is enough diversity, then this average is often a better estimate than any of the individual estimates. The reason for that is that individual biases and errors cancel each other out. Examples of systems that make explicit use of the *wisdom of crowds* concept include prediction markets and online auctions.

While this latter approach use effectively a group to solve a given problem, still during the problem-solving process there is no interaction between individuals. In fact, in order for the wisdom of crowds to work, it requires that there is no such interaction (otherwise, individual biases and errors will be shared and spread).

These problem-solving techniques have received a great share of attention due to their easy applicability and manageability. Models of such systems are also easy to create and analyze making them ideal testbeds to investigate (theoretically and experimentally) properties of problem-solving by groups. However, they do not include any *social* component, as each problem-solver acts independently of each other.

A more social approach can be found in the areas of *Social Computing* and *Social Information Processing*. These terms are often used in a general sense to include any system where humans, and not machines, are used to produce some outcome. Social Computing is more commonly used to describe computational systems that provide support for social interaction (e.g. forums, social network sites, chats), while Social Information Processing (SIP) is used to describe computer-mediated systems where social interactions are able to produce useful outcomes for some task.

An example of SIP system is its application to *image tagging*. The problem is to create useful *tags* (i.e. keywords) that can be used to describe the contents of an image. This is useful for online search engines, since tags allow users to search for images based on their content. Google uses a SIP system to perform this task by transforming

[6]Of course, the "user" may consist of a group of people itself, but such granularity is not important for the present discussion.

it into a game and using human players to tag images (AHN; DABBISH, 2004). The process is simple: by accessing a website, users are paired randomly and the same image is shown to both of them; these users must then type words or phrases that describe the image. Every time both users type the same tag, it is stored and used to describe the image afterwards; the players then receive a reward for reaching such agreement.

The idea behind this SIP is that if two unrelated people agree on describing a common image using the same tag, then it must be the case that this tag is a good descriptor for the image. While we do have a social interaction and the system does solve a well-described problem, interaction is somewhat limited (non-persistent, only between two random people) and the system's goal is effectively to *extract* knowledge from a crowd, rather than solve general problems.

Most SIP's have these characteristics. Social interaction are very restricted in some way and the task's goal is often to extract knowledge from users. This can be very different from general problem-solving, because when extracting knowledge the problem is already solved in some way - in the example, users already know tags to describe the images and the system is only validating the tags in order to avoid malicious/erroneous uses.

Another example of a system that can be considered a SIP is *wiki* systems, of which the best example is the Wikipedia[7]. In a wiki system, a document is shared among a group of writers and anyone can add or edit information contained in the document. After a number of editing iterations, the document may reflect the whole knowledge contained in the group, which can be more than what any single member knows. This is in contrast to the image tagging task, where the knowledge extracted was a intersection of the knowledge contained in the two users, while in a wiki the captured knowledge can be seen as the union of the group's knowledge. All social interaction happens in the document itself and is completely unstructured - every change to the document is automatically shared with everyone else.

None of the mentioned systems capture the full complexity of the example given in the beginning of this section - a group of independent social actors interacting to progressively solve a problem by exchanging information. For another example, consider the engineering problem of designing a product. An organization will deliver this problem to a group of employees, which will come up with a solution (e.g. a prototype or final product). It is not the case that each employee will come up with a product and then all these solutions will be combined to deliver the final product (as in wisdom of crowds). Neither it is the case that any single employee will take hold of the whole process and deliver one final product (as in crowdsourcing).

Rather, it is more likely that there will be an interactive process where employees will exchange ideas and knowledge in pairs or in group so as to reach an agreement on what the product should be and how it is to be built. There will be meetings, brainstorms, talks with consultants outside the company and several iterations of (hopefully)

[7]http://www.wikipedia.org

increasingly better designs, until a single final product is reached. This is also the same process that guides modern scientific research. A researcher will work typically within a group, going through the same basic processes as employees in a business organization, but will also attend conferences to exchange ideas with peers, publish papers in journals and proceedings to share findings with others and read several papers from peers.

While these interactive processes are closer to a SIP system due to the possibility of social interactions, they are also much less restricted - e.g. employees certainly do not interact only in pairs, a paper published by a researcher will be read by an unspecified and variable number of other researchers, a trader looking for a new system to beat the stock market will read books from several authors and so on. In the most general case, information must be allowed to flow in diverse ways: one-to-many, many-to-one and many-to-many.

In these cases, a social network, composed of otherwise independent actors, collaborate and interact to solve a problem. How these actors interact seems to be of major importance, since this network ultimately constraints how information flows between one actor to another, guiding the problem-solving process. To the best of our knowledge, there is no existing model for a system that makes primary use of communication between a set of networked agents to solve problems. To come up with one such model is a central part of this thesis, as detailed in the next chapter.

4 GOALS AND METHODOLOGY

In this chapter we define our general and specific goals and present the methodology used in the research carried out in this thesis.

4.1 Goals

In a broad sense, this thesis fits in a research agenda that focuses on understanding how social interactions in social networks can be used to solve problems. Our main objective is to better understand how social systems can be organized or built so that they can solve problems efficiently. That is, we are interested in how to extract computation from social networks. Two guiding research hypotheses are considered in this thesis:

Hypothesis 1: Given a set of independent agents acting to solve some arbitrary problem, the collective problem-solving capabilities can be improved by allowing these agents to exchange information on the problem.

Hypothesis 2: Given a set of interacting agents composing a problem-solving system, the network formed by the interaction patterns of these agents can affect the quality of the solution.

The first hypothesis aims at verifying whether allowing communication between independent problem-solvers can improve the collective outcome of a system composed of all such interacting actors. Hence, we are interested in comparing two types of related systems; one where a group of problem-solvers act individually and independently to solve some problem, and another where the same basic type of problem-solvers are allowed to exchange information on partial solutions.

The second general goal aims at understanding *how* network properties can affect distributed problem-solving performance. Assuming a system composed of interacting problem-solvers, we ask how the specific patterns of interaction and the resulting network can affect the performance of such system. Moreover, we ask how different common topologies compare to each other.

We consider strictly homogeneous actors, each running the same algorithm to process and distribute information. In addition, we are *not* interested in problems where the solution *requires* the cooperation of multiple problem solvers (e.g. pushing a heavy box that no single robot can push by itself), but rather in problems that can be solved by a actor, but where cooperation can lead to improved efficiency and/or efficacy. This latter case is better suited for performance analysis, as the raw number of actors involved is not important for how the solution is composed (i.e. they are not part of the solution).

Hence, we are also *not* interested in problems where the network itself is part of the final solution. That is, we do not wish to find a particular network configuration that is required to solve a problem (e.g. graph-colouring problems), but rather we view a network simply as a "facilitator" that helps reaching solutions that could be obtained without a network at all.

Under this general scope, several previous studies aimed at understanding the role of networks in a system composed of social actors. Most of these studies focused on relating specific network topologies to *properties* of a system (e.g. (MASUDA; AIHARA, 2007; KIRLEY, 2006; FORTUNATO, 2005; DELGADO; PUJOL; SANGÜESA, 2003; BIN; YU; SINGH, 2000)). For instance, in (FU; LIU; WANG, 2007) a set of learning artificial agents played a spatial version of the Prisoner Dilemma problem, each choosing whether to cooperate or defect based on interactions with neighbors; it was shown that the fraction of cooperating agents changed as a function of the network topology being used (the Small-World Network, for example, yields faster convergence towards cooperation when compared to a regular grid). In (FORTUNATO, 2005), it was shown that a transition towards dramatic opinion changes in a population happens earlier in Scale-Free Networks.

In these cases, a solution to the problem being solved is endogenous - while the network is not strictly part of the solution, the solution only makes sense in the context of the system: it is generated by and to the system itself. We, however, are interested in systems that generate solutions that are exogenous to the problem-solvers, in the sense that the solution is generated by the system, but the system is not necessarily the direct beneficiary of the solution. Hence, a solution can be extracted and used without any knowledge of the system and the processes that generated it, and the evaluation of this solution is effectively performed outside the system.

The following are specific goals of this thesis:

- To propose a model of social problem-solving where a network plays a strong part in the process of finding solutions;
- To identify relevant network properties and its effects in problem-solving performance;
- To compare the efficacy of common network structures in problem-solving scenarios;
- To verify the utility of social-inspired search algorithms as general search tools for automatic problem solving.

4.2 Methodology

Our approach is to propose a model and associated algorithms that are able to solve specific instances of well-stated problems and that make use of multiple agents acting in parallel towards solving the same problem; such agents are allowed to exchange information by means of a configurable network, so that it is possible to modify the network as we please in order either to fine-tune the algorithm to specific problems, or to represent cases where a network structure is predefined and fixed.

The proposed model uses recent ideas from Network Theory and is particularly inspired on concepts of Social Networks, in the way that vertices in a network consume and distribute information. It does not try to mimic social behavior in all its specificities, but rather tries to capture what we perceive as being the basic mechanisms for information to be processed in such environments. Therefore, the proposed model must use as much as possible the exchange of information as the main driving mechanism for solving problems. Models of *networked problem-solving* are nonexistent in the literature, as far as our knowledge goes, hence the need to propose one such model.

The algorithms studied in this thesis are composed of networks with hundreds of vertices and an even larger number of edges, which may be dynamic. The size, complexity and variety of these models make it difficult to study them by analytic means. Therefore, our approach is based on numerical simulations of the models on a computer, with posterior analysis of the outcomes.

We frame problem-solving tasks as *search tasks*, in which one desires to find one or more solutions that satisfy the problem statement. This means that our proposed model must be able to build solutions and evaluate them to verify whether or not a viable and acceptable solution was found. The social aspect of the model comes from its ability to build solutions by aggregating information from a social network. Moreover, we focus on *optimization* tasks, which provide a more objective measurement of the quality of solutions, as any solution can be evaluated and mapped to a real-valued number representing its quality.

The performance of a search algorithm can be measured by at least two methods (RARDIN; UZSOY, 2001). The first one measures the quality of solutions and the second how fast the algorithm finds a reasonable solution.

The former can be measured by setting a limit on resources (e.g. time, space, number of rounds, number of evaluations) after which the best found solution is returned and used to compose the evaluation of the algorithm. This is a reasonable methodology if the resource constraint is a natural part of the problem. Otherwise, setting the resource limit can be tricky, as the evaluation may change depending on how we set it. For instance, some algorithm may take a long time time to find very good solutions, while another find reasonable solutions very quickly, but is unable to reach very good solutions. If the limit is set high enough, the first algorithm is the winner, but with tight resource constraints, the second would be the choice.

The second method of evaluation consists in measuring how long the algorithm takes to find a reasonable or the best solution. The term “reasonable” can be applied in different ways, but often means reaching a solution that is "good enough" for the problem at hand. Of course, finding the best solution accounts for finding a solution that has a better evaluation than any other possible solution. The latter can only be used when the best solution's evaluation is known *a priori* (but note that the solution itself may not be known, only its evaluation) and the former requires the problem statement to define what accounts for a reasonable, acceptable, solution.

In practice, search algorithms are tested over either benchmark problems or real-world problems. In benchmark problems, many of the problem's properties are known (or specified) and, often, so is the best possible solution. For these, it is possible to relate an algorithm's performance to specific properties of the problem. Moreover, the complexity of the problem is often controllable in some way. Hence, evaluation is made easier since not only we know what the best solution is, but we can also have a better grasp on why the algorithm is performing as measured.

In real-world problems, the problem's structure and properties may not be known or well-understood and, more often than not, the best solution is unknown. When these problems are used, they are used to evaluate an algorithm in a very specific environment and its applicability to that type of problem, rather than unveiling insights about the algorithm itself. Nonetheless, some real-world problems are often used in the literature, making it possible to compare different algorithms applied to the same problem, even if an analysis on the reasons behind the performance may be somewhat more limited.

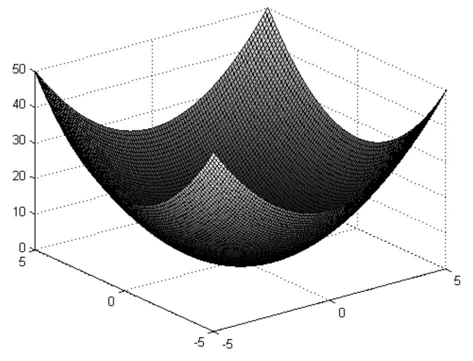
In this thesis we use both benchmark and real-world problems to evaluate and compare algorithms. We apply the algorithms to benchmark multi-parameter functions, with well-known properties, in order to better understand the algorithms themselves and their many parameters. We also use some real-world problems to compare the algorithms with other optimization approaches.

We chose five benchmark functions to conduct the evaluation of the algorithms. Two functions are unimodal (only one optima) and three are multimodal (multiple optima). Table 4.1 describes each function. Their dimension is a parameter that can be set and used to control how complex is the function, since it directly controls the size of the search space. We consider dimensions ranging from 2 up to 200 in our experiments. The number of dimensions is set differently for each case and is described in the appropriate sections.

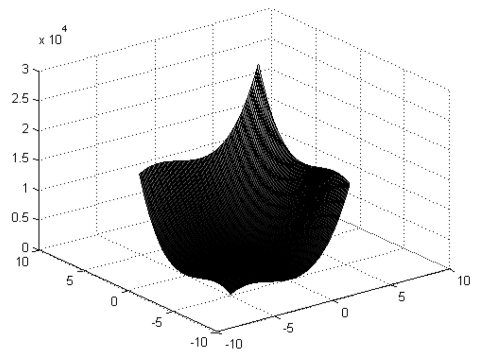
The unimodal functions are considered easy to solve by most optimization algorithms, while the multimodal functions chosen are considered quite hard, for different reasons. In what follows, we provide a brief description of each function.

The sphere function, depicted in its two-dimensional form in Fig. 4.1(a) is a simple unimodal function that can be easily solved by hill-climbing. The hyper-ellipsoid function (Fig. 4.1(b) is also an unimodal function, but each parameter has a different optimum value (while for the sphere function, all parameters have the same optimum value). Having different optimum values for each parameter may decrease performance of search algorithms that exchange parameters outside their locale in the function (e.g. crossovers in Genetic Algorithms).

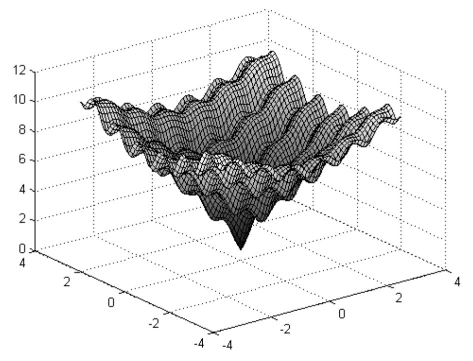
The Schwefel function (Fig. 4.1(e)) is multimodal and it has a second best mini-



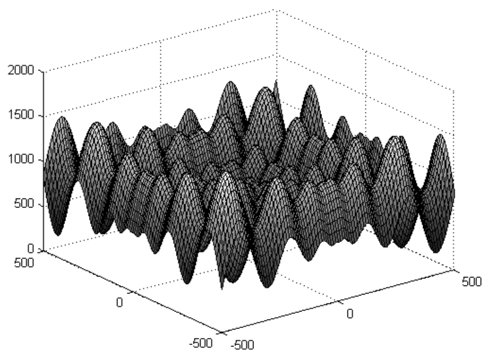
(a) Sphere Function



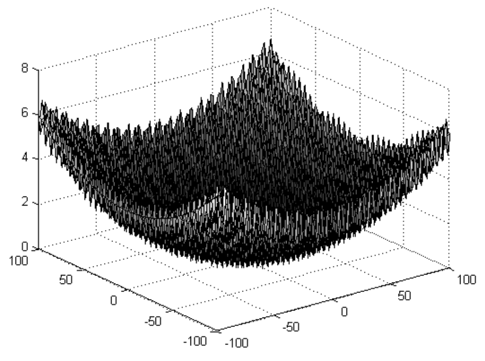
(b) Hyper-Ellipsoid Function



(c) Ackley Function



(d) Schwefel Function



(e) Griewank Function

Figure 4.1: Two-dimensional plot for benchmark functions used for performance evaluation of the algorithms.

Sphere	$f1(\vec{x}) = \sum_{i=1}^{\lvert\vec{x}\rvert} x_i^2$
Hyper-Ellipsoid	$f2(\vec{x}) = \sum_{i=1}^{\lvert\vec{x}\rvert} 5ix_i^2$
Ackley	$f3(\vec{x}) = -20e^{0.2\sqrt{\frac{1}{\lvert\vec{x}\rvert}\sum_{i=1}^{\lvert\vec{x}\rvert} x_i^2}} - e^{\frac{1}{\lvert\vec{x}\rvert}\sum_{i=1}^{\lvert\vec{x}\rvert} cos(2x_i)} + 20 + e$
Schwefel	$f4(\vec{x}) = 418.9829\lvert\vec{x}\rvert - \sum_{i=1}^{\lvert\vec{x}\rvert} -x_i sin(\sqrt{abs(x_i)})$
Griewank	$f5(\vec{x}) = \sum_{i=1}^{\lvert\vec{x}\rvert} \frac{x_i}{4000} - \prod_{i=1}^{\lvert\vec{x}\rvert} cos(\frac{cos(x_i)}{\sqrt{i}}) + 1$

Table 4.1: Benchmark functions used in this thesis.

mum that is located far from the global minimum, leading many search algorithms to become trapped. The Ackley function (Fig. 4.1(c)) has a deep valley, full of increasingly better local optima leading to the global optimum; it requires local optima to be constantly overcome. Finally, the Schwefel function (Fig. 4.1(d)), is the sphere function with added modulation, which creates a highly multimodal function, where the global optimum is surrounded by large hills.

Throughout most of the thesis, a greater focus is put into the multimodal functions, as they represent more realistically the sort of problems involved in general optimization problems. In particular, the Griewank function is the "hardest" of these functions, as it is both multimodal and non-separable, hence we shall use it more extensively and provide comparison with other functions where appropriate. As we shall see, since our proposed algorithms make no assumption on the underlying topography of the function being optimized, the results presented do not vary qualitatively across these functions.

As for real-world problems, we applied the algorithms to two test scenarios. The first one is the Traveling Salesman Problem, using data from real cities and from VLSI[1] circuits. The second one is concept learning, with data taken from real classification problems.

Given the stochastic nature of the studied models, a statistical analysis is performed over the results of each experiment. Therefore, we will report average results over multiple independent runs, along with information on standard deviation or standard deviation of the mean where the variation itself is not relevant. Unless otherwise noted, these statistics are applied over data extracted from 20 independent runs. Where comparisons are made, a standard *t-test* was applied to test for significance at 5% level of significance - i.e. whenever we say something is or is not significant, we are referring to this level of significance.

We also make extensive use of boxplots to graphically visualize distributions. A

[1]Very Large Scale Integration

boxplot (or box-and-whiskers diagram) is a graphical visualization of five statistical features of the data, namely: the sample minimum, the sample maximum, the lower quartile (Q1), the median (Q2) and the upper quartile (Q3). We also include the visualization of any outliers, which we consider to be any observation that lies more than 1.5 interquantile range(IQR) lower than Q1 or that lies more than 1.5 IQR above Q3, following (TAMHANE; DUNLOP, 1999).

The experiments were implemented using the programming language C++, using both the GNU G++ and Microsoft VC++ compilers running on multiple x86-compatible computers. The large number of experiments consuming random numbers required a careful decision over the pseudo-number generator to be used in the implementation. We used the Mersenne Twister MT19937 32-bits algorithm (MATSUMOTO; NISHIMURA, 1998), as it provides a very large period ($2^{19.937} - 1$), with relatively low time and space requirements.

5 MEMETIC NETWORKS: A MODEL OF NETWORKED SEARCH

As argued in Chapter 3, an adequate search algorithm that makes use of explicit networks is nonexistent in the literature. In this section we propose a general model that aims at performing search by using an underlying network to explicitly structure communication between multiple parallel searches. This model is inspired on the exchange of information in social networks and have the following main characteristics:

- It is population-based, executing multiple communicating searches in parallel;
- Communication between searches is structured by an explicit, controllable, network;
- New states are generated by aggregating information from neighbor states.

The inspiration for this model comes from Richard Dawkins' concept of *memes* (DAWKINS, 1976; DISTIN, 2004; BLACKMORE, 2000). Dawkins argues that cultural evolution takes place using similar mechanisms as genetic evolution. He coined the term *meme* as the cultural equivalent of the gene. A meme is anything that can be copied from one mind to another by any means. As Dawkins (DAWKINS, 1976) put it:

> Example of memes are tunes, ideas, catch-phrases, clothes fashions, ways of making pots or of building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperms or eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in a broad sense, can be called imitation.

Cultural evolution, Dawkins argues, proceed by a process of natural selection of memes. Individuals expose their memes to audiences, which copy (remember) those that are considered interesting or useful. These copies will then be able to propagate further, to other minds. In the process, some memes will be changed, effectively creating new memes, which will go through the selection process too. After many iterations, like in genetic evolution, only the fittest memes survive in the cultural pool. Of course, what is considered "fittest" is difficult to define precisely, but in general one may consider a fit meme one that provide some value, an incentive to be copied. Despite the common basic mechanisms, it is argued (BLACKMORE, 2000; DAWKINS, 1976) that memes seem to evolve at a much faster pace than genes.

Although meme evolution share much in common with gene evolution, it is by no means identical. Dawkins observed that memes seem to be more prone to continuous changes than genes. Genes are able to replicate with a high-fidelity, with mutations being rare. However, memes are more susceptible to changes and blending - i.e. memes are rarely passed on in their original form, but are changed by the "current" meme holder by adding his or her own ideas and blending with other memes. For example, when we read an article in a newspaper, we may comment about this article with a colleague, but we certainly never forward the article word-by-word to him or her; we emphasize those parts that we found more compelling, possibly adding our own opinion and other sources, maybe even citing incorrectly some date or name due to a misinterpretation.

However, it can be the case that such changes are not important at all, as they may be happening just on peripheral information surrounding the "real" meme. A similar process exists in genes, where parts of a DNA, called introns (RODRÍGUEZ-TRELLES; TARRO; AYALA, 2006), are non-coding - i.e. they do not code proteins and were often called junk DNA (although this is now very disputable). It has been argued that introns have the role of absorbing much of the mutation (WANG et al., 2007), so that the parts responsible for coding proteins have a smaller probability of being "hit", increasing the fidelity of copies. Likewise, much of the information contained in memes can be just accessories to a smaller part which is being transmitted with high-fidelity. Therefore, it is currently unknown whether this higher mutation plays a role in how fast meme evolution can happen.

We argue here that there is another significant difference between memes and genes, which may account for such faster evolution, and that we try to capture in our model. This difference is in how memes are propagated. While genes are transmitted to geographically close offspring, memes have no such limitation. With the invention of the press and with the current telecommunication infrastructure, memes are much less restrained by geography - i.e. they may propagate quickly to (almost) any part of the world. But it is not the case that memes are transmitted (broadcasted) to all individuals in the world, nor are transmitted randomly to any subset of individuals. Rather, memes diffuse through a social network, which impose restrictions on who may receive a meme generated at any node of this network. Such social network can be much more dynamic than any genetic diffusion system, as individuals can choose to create new connections (or have these connections created by random chance), effectively changing whom they may receive memes from or to whom they will pass on their memes. In genetic terms, it would be like an offspring that could receive genes not only from its parents, but from any other individual in the world.

Additionally, in a biological setting, there are two ways to create a new set of genes. The first one is by mutation, the sole driver of changes in asexual organisms. The second one is sex, which combines genes from exactly two individuals. In a cultural setting, however, a new meme can be created by changes in a single meme or by blending information from several memes. There seems to be no restriction on the maximum number of “parents” that may take part in such blending. A scientific paper is a good example of this property: the fact that a paper cite multiple sources is an evidence that the ideas being presented are an aggregation of these sources, in addition to some original thoughts and data from the author.

Moreover, the number of “parents” is not fixed at any number. Each meme may be affected by a different number of other memes. As we have argued, memes diffuse using a social network, where each vertex may represent a collection of memes (or an individual holding such memes). Hence, the number of influencing memes is ultimately defined by a node’s degree.

The following properties summarize what we believe are the most relevant differences concerning meme evolution when compared to gene evolution:

- Memes diffuse through a social network;
- Memes can influence a potentially large number of other memes;
- Memes can be influenced by a potentially large number of other memes;
- The number of memes influencing and being influenced by any other meme is not fixed and may vary from meme to meme and through time.

5.1 The Memetic Network Model

In this section we describe our model of networked search, named *Memetic Networks*. The name reflects a fundamental characteristic of this model: the inspiration on the diffusion of memes in a social network. The model’s goal is to allow for search using basic social principles involving the exchange of information (memes) between nodes in a network, while retaining the relevant properties of meme evolution discussed in the previous section.

The central idea is to have multiple parallel searches exchanging information on the state being visited through a network. Each search can be understood as a container for memes, while memes are propagated through the network to and from other containers. A meme in this model is any piece of information relative to the state held by a search, which include the state itself (i.e. a candidate solution to the problem) and meta-information about this state (e.g. its evaluation). Each search aggregates the memes received in some way, possibly adding local information to it, and makes available the resulting new memes to the network.

To comply with the desired properties, the model must be able to aggregate any number of memes relative to a candidate solution into a single candidate solution. The number of memes being received and the number of recipients of the new meme must be defined by network itself, without further restrictions. Moreover, we require some level of autonomy to each node, so that even a single node could perform search and improve a solution. This is necessary because we understand the network much more as a facilitator that allows for a better use of multiple parallel searches, rather than a necessary condition to perform search.

We also allow for some level of selfishness to the nodes, but do not allow them to be deceitful. Hence, a node will not make its own solution *worse* even if that may help the global search. The solution they make available to the network is always their best known solution.

Our model is a basic framework that guides how a network is formed and how it is used to perform search. The model uses of a graph $G = \langle V, E \rangle$, where V is a set of nodes representing candidate solutions to the search problem at hand and E is a set of edges $\{(u, v)|u, v \in V\}$ representing adjacency between vertices. These edges are not weighted, but may be oriented. Additionally, an evaluation function *eval* evaluates the utility of the candidate solution.

The set E defines the network topology and its properties, i.e. the dynamics *of* the network. It can be construed in two different ways: statically or dynamically. In the static case, E is predefined and remains fixed during the execution of the algorithm. On the second case, E may be modified in runtime by following rules to guide the creation of new connections and deletion of old ones. In the next chapters we will consider both cases.

The set V is composed of containers for memes, which encode candidate solutions for the search problem. Each node holds exactly one candidate solution. Once E is defined, V is updated by following update rules that make use of E to structure how information is to flow through the network (thus defining the dynamics *on* the network).

The evaluation function *eval* evaluates the state, indicating its value to the search problem at hand. For example, it may indicate whether the state is a goal state or map the state to a real number, indicating its utility to an optimization problem.

To better separate each stage of the model, we divide the general algorithm into three main steps: *Connection Step*, *Aggregation Step* and *Appropriation Step*. In what follows, we justify and explain each of these steps in turn.

5.1.1 Connection Step

The Connection Step is responsible for defining the structure of the network by setting its edges appropriately. Any changes made to E, the edges of the network, is thus part of the Connection Step. It may be executed only once, hence composing a fixed structure. In this case, network properties can be set a priori and remain unchanged during a search. Conversely, this step can be executed regularly, inducing a dynamic network. As we will see, dynamic networks can use information about the ongoing search to adjust connections, in order to try and improve performance.

Naturally, the latter is a more general case of the former, but we find useful to differentiate them nonetheless - Chapter 6 is concerned solely with the static case, while Chapter 7 deals with dynamic networks.

5.1.2 Aggregation Step

In the Aggregation Step, each node copy memes available to it through its connections (as specified in the Connection Step), blending them into a new candidate solution. This step is the social part of the search, as nodes may influence each other's search by exchanging information about their current states. It defines the precise way that memes are to be aggregated into a new solution, which, as we will see, is a problem-dependent issue. In order to do so, it must specify what a meme is in the context of the problem - hence, the framework provides a way to formalize the definition of a meme.

It is also the responsibility of this step to specify what kind of information is to flow through the connections. While communicating memes relative to each candidate solution is a natural choice, it is also useful to allow for the evaluation of each solution to be also propagated through the connections, so that nodes can decide whether to use the received memes based on their relative utility to the problem. Naturally, for some problems, each node could re-evaluate the received memes to measure their utility, but since evaluation may be a costly process it is more efficient if each node propagates its own utility through the network.

Any number of memes may potentially take part in the aggregation process. However, an aggregation method may decide not to use all information available, or select a subset to make use of.

5.1.3 Appropriation Step

In social sciences, an information is said to be appropriated when it is assimilated by the individual, making some concept or idea that was someone else's into his or her own (BLACKMORE, 2000). Following this definition, the Appropriation Step is responsible for adding to the recently-aggregated candidate solution any local information or process to it. It is in this step that external[1] information, not present in the received memes, can play a role and become integrated with the existing memes.

Such local information may take form of heuristics or as information available only to a single node. For an example of the latter, in a multi-robot foraging scenario the Appropriation Step could be responsible for adding information about a robot's sensors, which is not available to or from other robots. In a more abstract search, this step could apply for example other search algorithms (such as hill-climbing) to the solution before making it available to the network.

This step represents the independent search - i.e. the search that would be performed if there was no communication present. This step may be empty, in which case the Aggregation Step is the sole responsible for the search. On the other extreme, the Aggregation Step may not be present (or the network may not have any edge) and the model acts as $|V|$ completely independent searches.

[1]External in relation to the aggregation step.

5.2 Instantiations of the model

The described model is only a framework to guide the construction of networked search algorithms. An implementation of the model requires the specification of each step, which are to some extent problem-dependent. We call an instantiation of the model a *Memetic Network Algorithm* (MNA).

As we have argued, the Connection Step can be used to create a static network, with fixed topology, or a dynamic network, where connections can be created and removed at runtime. It is important to distinguish between two types of neighborhood implicit in the model. The first is the physical neighborhood, defined by the network itself, which represents access to information - i.e. given any node, it can receive information from any node in its physical neighborhood. On the other hand, the actual neighborhood is a subset of nodes in the physical neighborhood from which the node effectively access information from.

This is certainly true in a social space. For example, in a workplace, we work near several colleagues and they constitute our physical neighborhood. However, when executing a job, we may choose not interact with all of them, but only with a subset. This can happen for several reasons - in the workplace, we will interact with colleagues that can help get a job done and will avoid those that have nothing to contribute. This defines the actual neighborhood. From this example it is clear that the actual neighborhood may be dynamic even when the physical neighborhood is static; if the job changes, or suddenly some colleagues are suddenly able to help us, we may change with whom we are interacting.

The Connection Step is responsible for defining the physical neighborhood, while the Aggregation Step defines the actual neighborhood. In this section we detail possible Aggregation Steps for different scenarios, but we delay the discussion on the Connection Step for later chapters.

The Aggregation Step is responsible for using information from a node's neighborhood to build a new solution. As such, the exact implementation can be dependent on the codification used for the problem. As we have argued, the concept of memes is mostly based on the idea of imitation. This social/memetic approach leads to a very simple aggregation method that is based on copying successful solutions from nodes' neighbors, as follows:

Definition *(Aggregation by Copying the Best Neighbor)* Let A be the set of adjacent vertices to any vertex v; let $u = argmax_{x \in A} eval(x)$. If more than one vertex in A may satisfy this condition, u is chosen randomly among these. Then, make $v \leftarrow u$, otherwise v is left unaltered.

This method, for each vertex, copies the whole solution of the best neighbor to the vertex. Naturally, the way a solution is copied is dependent on the implementation details, but it is also clear that the general idea would work at least for a wide range of problems using very different codifications. While the physical neighborhood is defined by the network's topology, the actual neighborhood is restricted to a single node - the best node in the physical neighborhood. It is with this node that other nodes interact.

This Aggregation by Copy, as we will refer to the described method hereafter, is very simple and does indeed use some information from all neighbors. However, except for the best neighbor, only the evaluation of the solutions are being used, not information contained in the solutions themselves. Copying is the very essence of the concept of memes and this method treats a solution as a single, complete, meme. As we have argued, the issue of memes' granularity (i.e. what particular division of information is to be treated as a meme) is still an open question and the model proposed here allows for experimenting with different granularities.

We have also argued, following (DAWKINS, 1976), that memes can be recombined and blended to create new memes and that that is a central characteristic of meme evolution. In our model, to effectively blend the solutions available in multiple neighbors, one must know how a solution is represented so as to devise an aggregation method that fits the problem. Such method may not be immediately obvious, and certainly more than one method exists for any given representation. In what follows, we describe possible implementations for several problems, using different representations for solutions.

5.2.1 Real-Valued Parameter Optimization

In this scenario, an arbitrary multi-parameter function $f(\vec{x})$ is given for which we want to find $\vec{x}*$ so that $f(\vec{x}*)$ is the minimum possible value of the function. Hence, this constitutes a minimization problem[2]. A candidate solution for the problem is represented by a vector $\vec{x}$ of real values inside some specified range. Hence, it is a case of bounded, constraint-free, real-valued optimization (MACNISH; YAO, 2008).

A straightforward aggregation method that makes use of information contained in neighbors can be created by exploiting the chosen representation: compose a new vector by averaging over elements of the solution.

Definition *(Aggregation by Averaging over Neighbors)* For every vertex v, let $A(v) = \{u|(v,u) \in E \wedge eval(u) \succ eval(v)\}$. Let $\vec{x_i}$ represent the candidate solution contained in vertex i, and $x_{i,j}$ the $j-th$ component of this solution. Create a new aggregate candidate solution $\vec{x_0}$ such as $x_{0,j} = \frac{1}{A}\sum_{i \in A} x_{i,j}$.

Note that this aggregation method compute the average over only better-evaluated neighbors. This is again a case where the actual neighborhood is different from the physical neighborhood. The network topology works in the direction of limiting each node's choice, but otherwise set no constraints in how the defined neighborhood will be used. The reasoning behind this heuristic is that when solving a problem, it is not in our interest to interact with sources that are not doing very good at solving that problem; rather, we will pursue interaction with nodes that are actually doing better than we are.

[2]The maximization problem is symmetrical; any minimization problem can be described as a maximization problem and vice-versa without loss of generality.

Averaging over components of solutions is not something new. Genetic Algorithms applied to real-valued parameter optimization often make use of this approach to perform the crossover between pairs of individuals. In a Memetic Network, however, the average may be performed over an unspecified and potentially variable number of solutions. Averaging is a scalable method - unlike, for example, a single-point crossover in GA, it supports seamlessly and efficiently any number of components.

Averaging is also used in ensemble learning, where a group of independent classifiers are trained and their output is averaged to compose the final result, which is often more accurate than the output of any individual classifier. This idea became popularized by the term wisdom of crowds and the book of the same name. However, to obtain "wisdom of crowds" it is required that the individual problem-solvers are independent from each other, which is not completely the case here - nodes are constantly exchanging information with each other, making their solutions correlated.

We can calculate the computational efficiency of running such method. Let N_p be the number of physical neighbors, N_a the number of actual neighbors and C the number of components in a solution, running this algorithm takes time in $\theta(N_p + N_a C)$ for each node, since we have to find the best performing neighbors ($\theta(N_p)$) plus take the average over all components of every actual neighbor ($\theta(N_a C)$).

A minor variation to this aggregation method is to perform a weighted average, thus allowing a stronger influence by better solutions in the construction of new solutions. Such variation allows for a smooth transition between the Aggregation by Averaging and Aggregation By Copy - given enough weight, the best solution will dominate the average, hence reducing the method to the copy mechanism. A different approach altogether is detailed below.

Aggregation By Shuffle over Neighbors. For every vertex v, let $A(v) = \{u | (v, u) \in E \wedge eval(u) \succ eval(v)\}$. Let $\vec{x_i}$ represent the candidate solution contained in vertex i, and $x_{i,j}$ the j-th component of this solution. Create a new aggregate candidate solution $\vec{x_0}$ so that each component $x_{0,j} = x_{k,j}$, where for each value of j, $k \in A$ is randomly selected.

This aggregation method uses information from multiple sources to compose a new solution, but in opposition to the Aggregation by Averaging, it blends the solutions by randomly shuffling components of each solution, so that each component is fully copied from a single vertex - i.e. blending is coarser than in the previous case, as it take place in the solution as a whole, but not in individual parameter. In this case, a meme is considered to be a single component of a solution.

This method is more efficient than the previous one, since we do not have to access all components of all actual neighbors. It rather can be implemented so as to run in $\theta(N_p + C)$ (for each node) by constructing an array of the best performing physical neighbors ($\theta(N_p)$) and then for each component of the solution selecting randomly a component from the nodes in this array ($\theta(C)$).

There are also several choices for the appropriation step. As discussed previously, the appropriation step can perform any local changes to the solution in addition to the aggregation process. For instance, we could apply some local search heuristic to the aggregated solution, such as a hill-climbing or a gradient search. However, the simplest form of local modifications is arguably a simple random search, as follows.

Definition *(Appropriation by Randomization)* For each real-valued parameter, with probability p_n, replace the parameter with a random number inside the domain's range.

This implements (for small values of p_n) a random walk - i.e. small changes create a new solution in the neighborhood of the current solution. Of course, this aggregation method alone it is not an effective search method, as we are only randomly generating new solutions, without concerns on guiding the search in any way. Nonetheless, we shall use such appropriation in most of our experiments.

A complete algorithm using Aggregation by Shuffle and Appropriation by Randomization is depicted in Algorithm 1.

Algorithm 1 Memetic Network Algorithm using Aggregation by Shuffle

```
Initialize set $V$ with $N$ solutions with $k$ random components each
while termination condition not met do
  initialize network (Connection Step)
  $newNodes \leftarrow \emptyset$
  for $node \in V$ do
    evaluate solutions in nodes
    $C \leftarrow\ emptyset$
    for each $n \in neighbors(node)$ do
      if $eval(n) \succ eval(node)$ then
        $C \leftarrow C \cup n$
      end if
    end for
    if $C \neq \emptyset$ then
      for $j = 1$ to $k$ do
        $randomNode \leftarrow$ random node from $V$
        $newNode(j) \leftarrow randomNode(j)$
      end for
    else
      for $j \leftarrow 1$ to $k$ do
        if $random(0, 1) < p_n$ then
          $newNode(j) \leftarrow random(lowerbound, upperbound)$
        end if
      end for
    end if
    $newNodes \cup newNode$
  end for
  $V \leftarrow newNodes$
end while
```

5.2.2 Binary Function Optimization

Genetic Algorithms introduced the idea of manipulating information in a low, genetic level instead of working on coarser, phenotypical, granularities. Binary strings are often used for that purpose. However, it has been shown that the original reasoning in favor of such low-level representation does not hold and that the choice of representation does not affect how search is performed, at least for any two bijective representations (FOGEL, 2000). Therefore, other representations that are more natural to the problem have been preferred in the literature.

Binary representation, nonetheless, has the advantage of being very general and relatively standard codification. That is, if a search technique exists that works on binary representation then, if the original representation can be converted to a binary form, the technique can be used without changes [3]. This is particularly useful in search algorithms that perform some sort of recombination between solutions (e.g. crossover); such recombination operators must typically be custom-built for the representation used and using a standard representation may reduce the hassle of customizing the algorithm to the problem. Therefore, it may still be useful to provide search algorithms that operate over binary representations. If, for nothing else, at least because many problems are naturally represented by binary strings. Additionally, it allows for a more direct comparison with the canonical GA.

The scenario is the same as the one in the previous section: we are interested in optimizing a multi-parameter function by adjusting its parameters. However, now parameters are represented by a single contiguous binary string, instead of real numbers. More formally, a candidate solution is composed of a bit vector $\vec{B} = b_1, b_2, ..., b_n$ of length n; each of the C parameters is encoded using k contiguous bits, hence $n = Ck$.

Again, we wish to use a simple method to aggregate multiple sources of information. Unlike recombination in GA, the method must be scalable and support any number of sources. For instance, the proposed Aggregation by Shuffle over Neighbors, proposed previously, can be applied to binary representations in a straightforward way, simply by considering each bit as a parameter. Hence, bits are copied from random neighbors, forming a new solution. A different approach made possible due to the discrete nature of the problem is to aggregate a new solution by taking the most common value for each bit among solutions in the neighborhood.

Definition *(Aggregation by Majority Voting)* For every vertex v, let $A(v) = \{u|(v, u) \in E \wedge eval(u) \succ eval(v)\}$. Let $\vec{B_i}$ represent the candidate solution contained in vertex i, and $b_{i,j}$ the j-th bit of this solution. Create a new aggregate candidate solution $\vec{B_0}$ such as :

$$b_{0,j} = \begin{cases} 0, & \text{if} \frac{1}{A} \sum_{i \in A} b_{i,j} < 0.5 \\ 1, & \text{if} \frac{1}{A} \sum_{i \in A} b_{i,j} > 0.5 \\ b_{0,j}, & \text{otherwise} \end{cases}$$

[3] Of course, this is true for all representations, but conversions to binary strings tend to be effortless.

This method performs a majority voting and takes the most common value as the value to be used when composing a new solution. If there is a draw for any bit, that bit remains unchanged. Again, an alternative method is to use weighted voting, allowing more "votes" to better evaluated solutions. This method clearly runs in $\theta(N_p + CkN_a)$ and can be less efficient than those previously presented because k increases exponentially with the required precision and domain's range.

A random search can be implemented in the appropriation step in a similar fashion as the real-valued case. We simply flip each bit of the aggregated solution with some probability.

Definition *(Appropriation by Binary Randomization)* Flip each bit of a solution with probability p_n.

As with the real-valued method, for small values of p_n, this method implements a random walk, generating solutions near the aggregated solution.

5.2.3 Combinatorial Optimization

In a combinatorial optimization problem, we wish to find a permutation of elements that optimizes some evaluation function. This is the case of e.g. scheduling problems, vehicle routing problems and the Traveling Salesman Problem (TSP) (APPLEGATE et al., 2007). We focus on the latter in our discussions to exemplify this class of problems.

The TSP consists of finding the optimal Hamiltonian path in a graph. That is, given a graph, the goal is finding a path that include every vertex in the graph exactly once such that the cost of this path is minimized. This is a NP-Hard problem in its general form (APPLEGATE et al., 2007). We consider the case where the graph is fully connected, hence it is always possible to reach directly any vertex from any another vertex.

A candidate solution for the TSP can be represented as a vector $\vec{r}$ of integers, where each integer represent a city[4] in the graph and the vector represent the order that the cities must be visited (see Fig. 5.1). The evaluation of such candidate solution is straightforward. Therefore, the problem is finding the best of all $\frac{(|\vec{r}|-1)!}{2}$ possible permutations.

Since a candidate solution for the TSP is discrete in its nature, we could consider using the same Aggregation by Shuffling or Majority Voting proposed in the previous section. However, it should be clear that this could lead to solutions that are not well-formed. Figure 5.2 depicts a case where shuffling can lead to a solution where some cities appear more than once, while other are excluded.

Some repairing algorithm would have to be used to fix the cases where a malformed solution arises. Instead of fixing solutions, we propose an aggregation method that always returns well-formed hamiltonian paths. The idea is to compose a path by looking

[4]We use the term city to refer to a vertex in the TSP graph, so that there is no confusion with vertices of the Memetic Network.

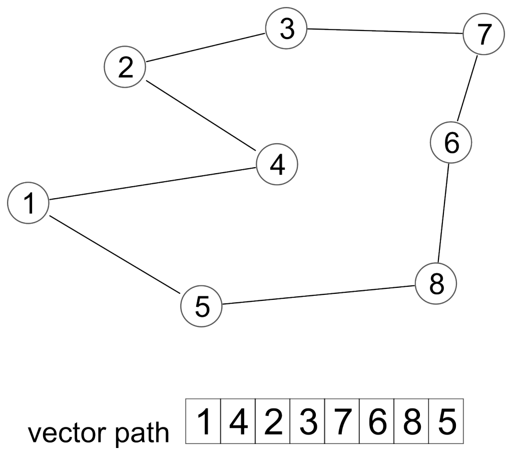


Figure 5.1: A hamiltonian path over 8 vertices and the corresponding representation using a vector.

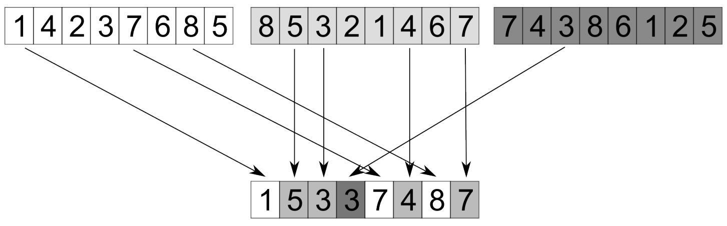


Figure 5.2: Three well-formed hamiltonian paths are aggregated using the Shuffle mechanism into a new candidate solution, but this new solution is not a hamiltonian path.

for common transitions between cities, instead of the absolute position of each vertex in the vector.

Definition *(Aggregation by Common Transitions)* For every vertex v, let $A(v) = \{u|(v,u) \in E \wedge eval(u) \succ eval(v)\}$. Start the new path with an arbitrary city c_0. The next city c_1 will be the one that follows c_0 most frequently in the solutions in $A(v)$. The process is repeated until a full path is formed. Ties are solved by random chance.

This procedure will always return well-formed paths that include the most common transitions in the solution's actual neighborhood. Unlike the Aggregation by Majority Voting, this method introduces heuristic knowledge about the problem into the aggregation process - a good path is formed by good partial routes, hence it may be worthy to find and keep these.

The appropriation step must also be adapted to retain well-formed solutions. Randomly swapping two cities is an adequate choice, as any well-formed solution will remain well-formed after the swap.

Definition *(Appropriation by Swap)* For each city in the solution, swap it with a random city with probability p_n.

Hence, the application of a MNA to the TSP shows that for certain scenarios, the aggregation and appropriation methods must be adapted to the problem.

5.2.4 Experiments

In order to have a better understanding on the role of each component of a Memetic Network Algorithm, we conducted some comparative experiments disabling some parts of it. In what follows, we used the binary shuffle method for binary-encoded function optimization described in Section 5.2.1, with 16 bits used to encode each dimension.

Figure 5.3 shows results for 1000 rounds of the algorithm applied to the Griewank function in three different cases: without the Aggregation Step (with $p_n = 0.01$), without the Appropriation Step (or with $p_n = 0.0$) and with both steps (full algorithm, with $p_n = 0.01$). For all cases, a static network with 100 nodes, a regular ring topology and neighborhood size of eight is used. The same qualitative results was observed for all other functions.

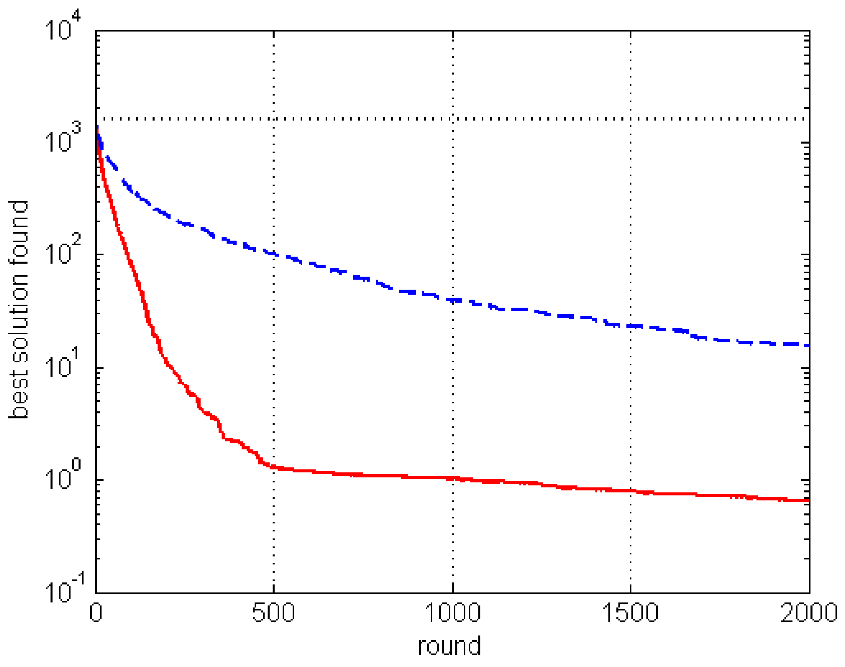


Figure 5.3: Evaluation of the best solution found for each round of the full MNA (solid line), the algorithm without the aggregation step (dashed line) and the algorithm without the appropriation step (dotted line).

It is possible to observe that the full algorithm performs much better than any step running individually. When only the Aggregation Step is absent from the algorithm, the algorithm is reduced to a random search, no communication is present and new solutions are randomly and independently generated from the current set of solutions. We observe that the probability of finding good solutions, or even of improving the initial solutions, is small in this case.

On the other hand, without the Appropriation Step, the algorithm has limited ability to explore the search space, becoming restrained by the initial set of solutions generated randomly at the beginning of the algorithm. When building new solutions by shuffling the neighbor's bits, initially new solutions can be formed, and such solutions have a better-than-random chance of being an improvement over the existing solutions; however, not all bits have the same probability of appearing in a new solution, since this is dependent on the initial set of solutions. This may lead to a state where no further search is performed. The simplest example happens when all nodes encode the same solution - shuffling over any number of nodes create no new solutions in this case. Once the information available is depleted, the algorithm becomes stuck, because it can no longer create new information (i.e. explore other areas of the search space).

The Appropriation Step must guarantee that the whole search space is potentially available for the algorithm to explore. This is the case for all appropriation proposals so far: any possible solution in the search space has a non-zero probability of being generated at this step. Hence, new information is always available to be aggregated, making the algorithm less susceptible to premature convergence. This step guarantees convergence to the global optimum in the limit, since at each step the global optimum can be generated.

This is akin to Evolutionary Algorithms, where mutation is only used as a background operator to keep the whole search space available to the search being performed. Since it is a background operator, mutation rates are kept small. The same holds for our algorithm, as shown in the next section.

5.2.4.1 Effects of Noise

Apart from the network, the only parameter of the algorithm is p_n, the probability of each bit being flipped. Small values of p_n means that new solutions are created, during appropriation, near the current solutions, hence exploring a small neighborhood in the search space. Large values, on the other hand, allows larger "jumps" in the search space - in the limit, for $p_n = 1.0$, a completely random solution is generated every time.

We tested several values for p_n and evaluated the impact in search performance. The results shown in Figure 5.4 are relative to the best solution found after 2000 rounds using a regular ring network and the Aggregation by Shuffling method applied to real-valued solutions for the Griewank function. The best result is obtained for a very small value of p_n. For large values, any information obtained from the aggregation step is destroyed, by replacing the outcome with random solutions.

The same qualitative behavior is observed for all tested functions, but the precise optimum value of p_n changes with the particular function, its particular number of dimensions and the aggregation method used. Nonetheless, this value is always very low, ranging approximately from 0.01 to 0.05. While we try and fine-tune this parameter in later chapters, for most of the arguments in this thesis we are not interested in obtaining the *best* performance from the algorithm, but rather provide comparative results. Hence, we use for most experiments the value of $p_n = 0.01$, which provides a good average performance on the functions being considered.

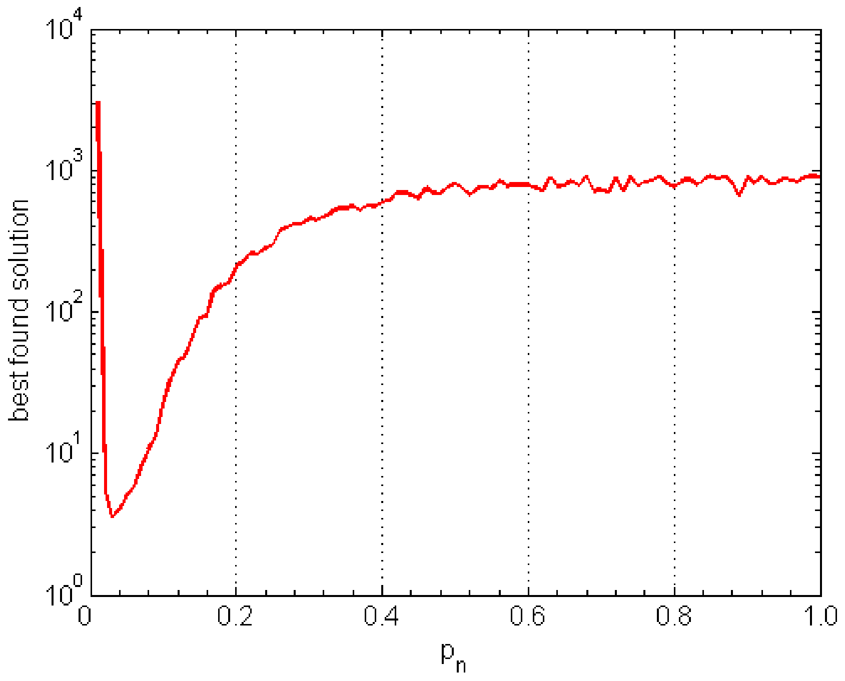


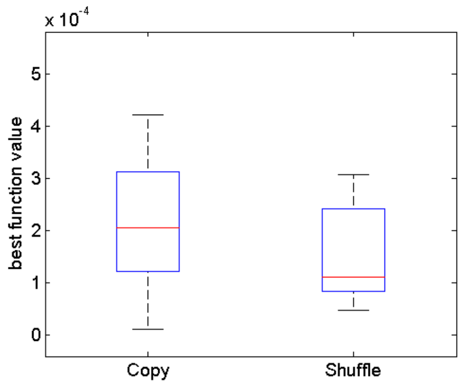
Figure 5.4: Best solution after 5000 rounds for different values of p_n, using the Ackley function and a ring network. Adding too much or not enough noise make the algorithm unable to find good solutions.

5.2.4.2 The advantage of aggregating

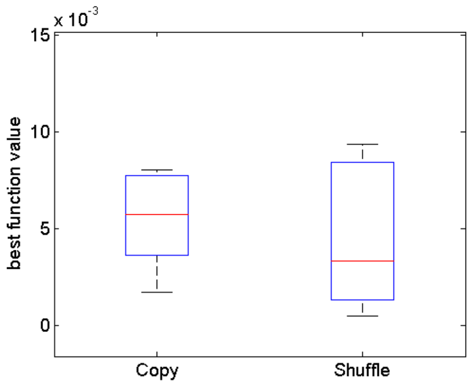
Let us consider two different MNA. The first, uses the Aggregation by Copy, which simply copies the best solution in the neighborhood. The second, uses the Aggregation by Shuffle, which takes information from the better evaluated neighbors' solutions to compose a new solution. We experimented with both algorithms on different optimization functions using networks with identical properties. Even though the results presented here are useful to provide a general idea on the behavior of each method, we will see in the next chapter that each aggregation method performs their best using strikingly different parameters. The following parameters were used: regular ring network, neighborhood size of eight, 200 nodes, $p_n = 0.01$.a

Figure 5.5 shows the quality of the best function value after 10000 rounds using each method for different 10-dimensional functions. We can see that in this case, both methods present very similar performance. Indeed, no statistical significant differences can be found between the results provided by the two methods.

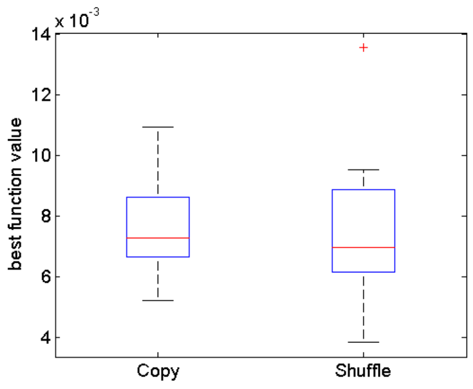
However, for a higher dimensionality, aggregating from multiple sources provides a considerable advantage. Figure 5.6 shows box-plots for the same functions, but now with 100 dimensions. In all types of functions tested, Aggregation by Shuffle provided better results when compared to Aggregation by Copy. Not only the found solutions were better on average, but also deviations were smaller for the shuffling method (i.e. the algorithm provides improved solutions consistently). Such difference in performance is persistent throughout any single round, as shown in Fig. 5.7.



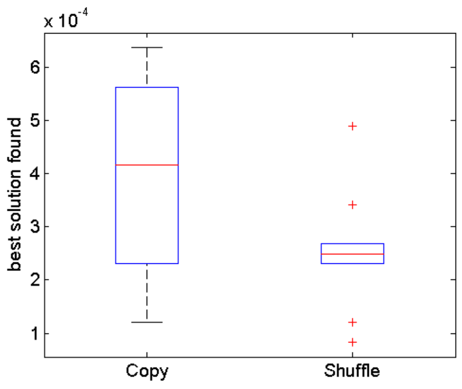
(a) Sphere function



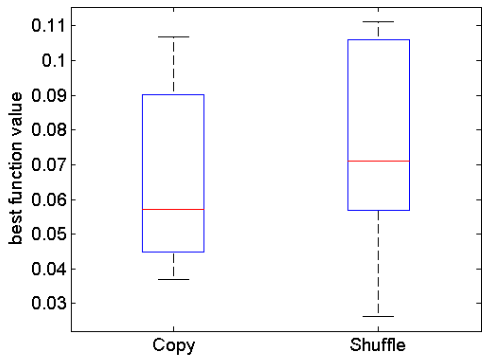
(b) Ellipsoid function



(c) Ackley function

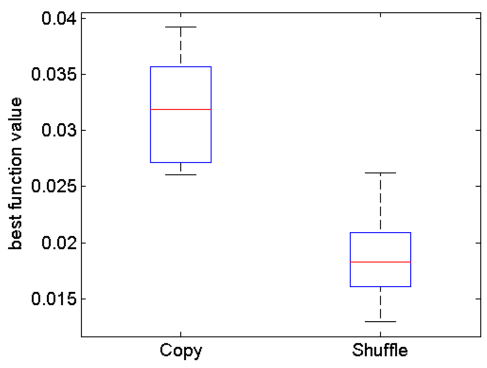


(d) Schwefel function

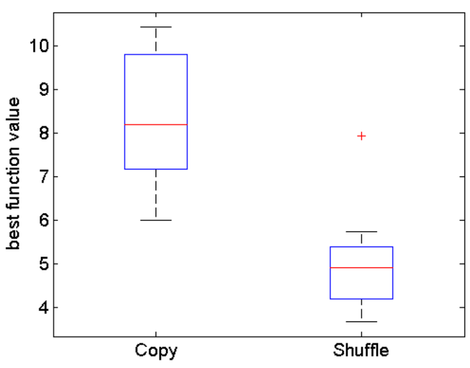


(e) Griewank function

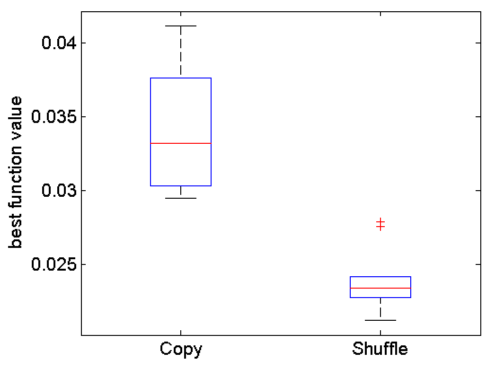
Figure 5.5: Comparative box-plot of the best function evaluation after 10000 runs and 20 independent runs using Aggregation by Copy and Aggregation by Shuffling for different 10-dimensional functions.



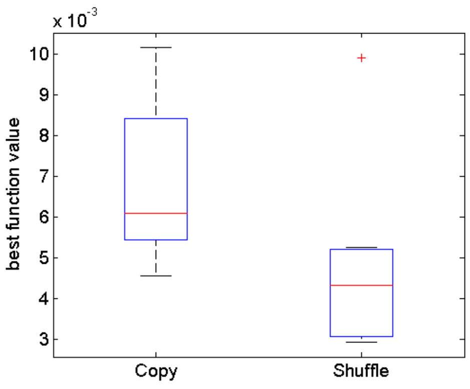
(a) Sphere function



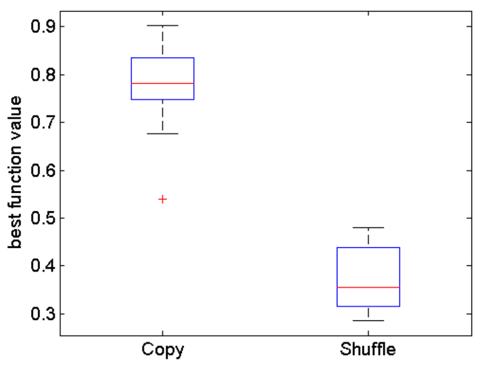
(b) Ellipsoid function



(c) Ackley function



(d) Schwefel function



(e) Griewank function

Figure 5.6: Comparative box-plot of the best function evaluation after 10000 runs and 20 independent runs using Aggregation by Copy and Aggregation by Shuffling for different 100-dimensional functions.

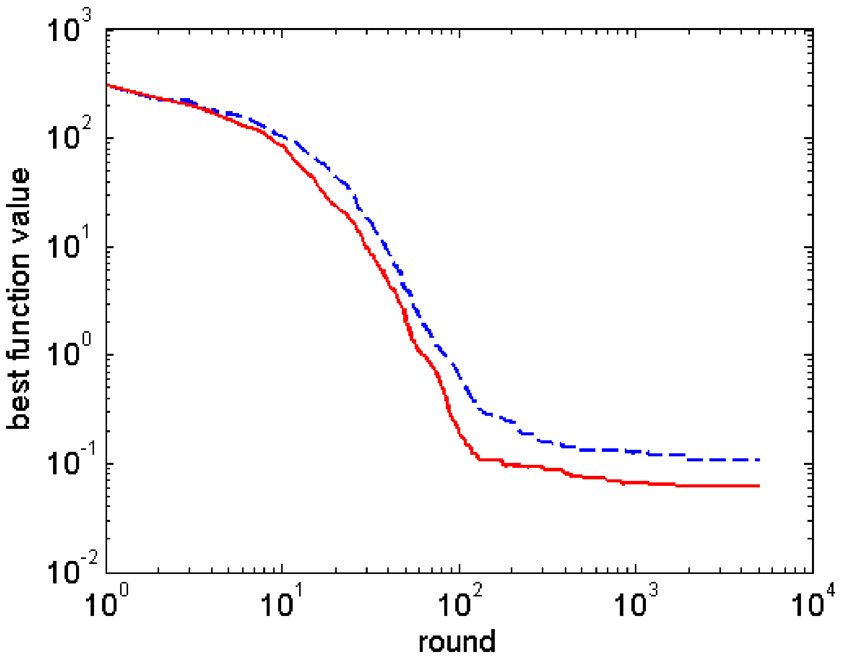


Figure 5.7: Convergence history when optimizing the Griewank function using Aggregation by Copy (dashed line) and Aggregation by Shuffling (solid line). Averages over 20 independent runs are presented. Shuffling provides better solutions throughout the whole run.

5.3 Discussion

To the best of our knowledge, there is no general search model that captures the features that characterize meme evolution. There are, however, a few works that are related to ours and aim at studying the same basic problem tackled here.

In (LAZER; FRIEDMAN, 2005), a study was conducted on a group of interacting people trying to find a solution to a simple problem under different conditions. In the experiments, they allowed the participants to interact under an strict environment, so that several network topologies could be forced upon then. They then commented on the results, showing that different network topologies resulted in differences in how fast the group was able to converge to a solution. The problem in question was how fast did the solution spread through the group, an issue close related to how technological advances propagates through the society, rather than how good was the solution found (because the problem was very simple).

A similar study was conducted by (MASON; JONES; GOLDSTONE, 2008), where, again, a group of people was asked to solve a simple problem (number guessing) and the particular network structure forced upon the group was related to how fast did the group find the optimal solution. Only four (fixed) network structures were tested and similar results to that presented in (LAZER; FRIEDMAN, 2005) were found.

The model and results here presented distinguishes themselves from these previous work in three main features. First, we consider search tasks that are considerably more complex than previous approaches, which mostly consider exceedingly simple search tasks. In contrast, we consider optimization in multiple dimensions and scenarios, which may benefit from more sophisticated mechanisms to build new solutions, as we have discussed. Second, we provide extensive comparison with more traditional search techniques, effectively evaluating in which domains a network search can perform well or badly. Third, we conduct a more systematic approach towards experimenting with networks, examining multiple network types and their full range of relevant parameters - conversely, previous work deal with a very restricted number of networks and almost no experiments are made on variations of the same type of network.

Those familiar with evolutionary algorithms, and genetic algorithms in particular, may have noticed that the proposed model have similarities to these algorithms. Indeed, there are studies on spatially structured GAs, where the population is distributed in such way that parent selection occurs only over a limited neighborhood. Moreover, other studies investigate the utility of combining multiple parents during a crossover. Therefore, the Memetic Networks model could possibly be classified as a particular type of multi-parent spatially structured evolutionary algorithm. However, we observe the following differences:

- *In multi-parent evolutionary algorithms*, the number of parents is fixed during a run of the algorithm. Hence, every crossover will use the same number of parents. In a MNA, the social network allows for each "individual" (node) to use a different number of "parents" to compose its new solution. If the network is dynamic, these numbers may change during a single run of the algorithm.

- With a few noteworthy exceptions, studies *in spatially structured evolutionary algorithms* are motivated by the need to create distributed versions of the algorithms, running on clusters and computer networks, which impose a natural structure over the problem - the result is that most works are concerned with grids or toruses as the underlying structure, limiting the network's properties that may affect performance.

- As far as our knowledge goes, there are no studies that merge both lines of research and investigate multi-parent recombination in spatial structures.

The main reason, however, for this work *not to* be classified under the broad area of evolutionary algorithms is that it departs from the basic analogies typically used in that area. The genetic analogy breaks down as there are no known natural occurrence of recombination of more than two chromosomes. Memes provide a much better analogy and justification for the recombination of multiple sources of information. The social network analogy is also more suitable to model such recombination than the simpler pair mating, allowing more flexibility in not only controlling, but also analyzing patterns of information diffusion using tools from Network Theory.

Maybe one of the main differences between a GA and a MNA is the statistical approach towards building new solutions. While a GA builds its solutions from two parents, a MNA using the appropriate Aggregation Step will build a solution using several “parents”, which is amenable to statistical analysis - e.g. in the Aggregation by Common Transitions we build a new solution by taking the transitions that occur most often in the connected vertices. This is a central social aspect of the MNA, where new solutions are based in other solutions that are performing well - the higher the frequency of a solution (or partial solution), the higher the probability of it being adopted by others.

The term *Memetic Algorithm* was introduced in (MOSCATO, 1989) to denote a type of Genetic Algorithm that uses some local search heuristic along with the traditional genetic operators. It is however mostly unrelated to our approach both in its goals and its embedded concepts of memes.

While we used the term *social network* to denote the framework’s underlying inspiration, our attempt is to capture the basic features of constrained information exchange and distributed processing, without concerns at effectively modeling actual social behavior. Therefore, the framework is composed of what we believe are the essential features to perform a networked search, but we leave out complexities from real social networks. The implementation of each step into a working model may be done so as to display characteristics of real-world systems, such as cognitive traits, local heuristics or preferential connections between nodes. We however make no attempt at emulating any existing system in its full complexity.

6 SEARCH IN STATIC NETWORKS

We begin our studies on networks by asking how well-known network properties of common network classes affect search performance. Experimenting with static networks allows for a finer control over the network's parameters, hence making it easier to relate properties to performance. Nonetheless, several real-world scenarios present a fixed, or slow-changing, network. In a business organization, a hierarchy is typically fixed and employees consult with approximately the same colleagues when solving problems. Friends consult with roughly the same groups of friends (BARABÁSI, 2003).

In these cases, the network topology, and its structural properties, can be considered fixed. Hence, the physical neighborhood of vertices remains unchanged, even if the actual neighborhood is allowed to change (e.g. even though our group of friends are somewhat stable, for a specific problem we may consult only with a subset of these friends).

In order to create static networks using the Memetic Network model presented in the previous chapter, we define the Connection Step and execute it only once, at the beginning of a search. When executing multiple independent runs, we allow each run to create a new network.

6.1 Search in Small-World Networks

Let us start by using Small-World networks to structure the information flow for our search tasks. Several studies - see e.g. (BIN; YU; SINGH, 2000; KLEINFELD, 2002; KUPERMAN; ABRAMSON, 2001) - suggests that real social networks have indeed many properties of small-world networks, making this class particularly interesting to our studies. Our interest resides in instantiating a MNA to perform search using a small-world network and then implement experiments to understand how properties of these networks can impact the search.

The Connection Step implements the generative model for small-world networks shown in Chapter 3. We start with a regular ring network, with each vertex connecting to the K immediate neighbors, and then add random shortcuts - for each edge in the network, we rewire it to connect to a random vertex with probability β. It is immediately clear that K (the neighborhood size) and β (the rewiring probability) are central parameters for the model and we focus on analyzing them.

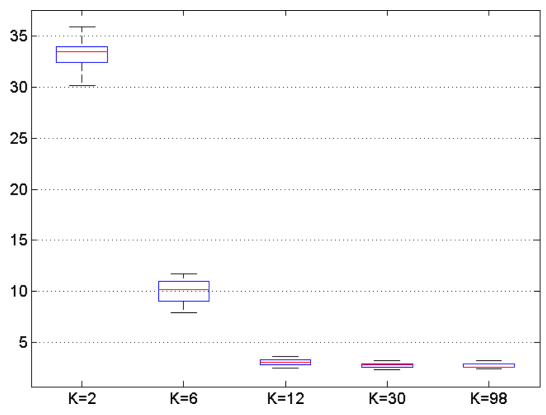
6.1.1 The Effects of Neighborhood Size

The neighborhood size K defines the average vertex degree in a network. For the regular ring network, K defines the exact network degree, since all vertices have the same degree. In our model, K defines the maximum number of solutions that may be used to compose a new solution, i.e. the physical neighborhood. Also, K is always even, since the neighborhood is always increased by adding two new vertices to it (one neighbor on each "side" of vertices). Hence, the smallest value that induce a connected network is two and we may vary K in increments of two. The largest value for K is $N - 1$ (or the closest even value smaller than $N - 1$), in which case the network is fully connected. The higher the value of K, the more sources are available for the aggregation step. Hence, for each node, K defines the amount of information available to build new solutions.

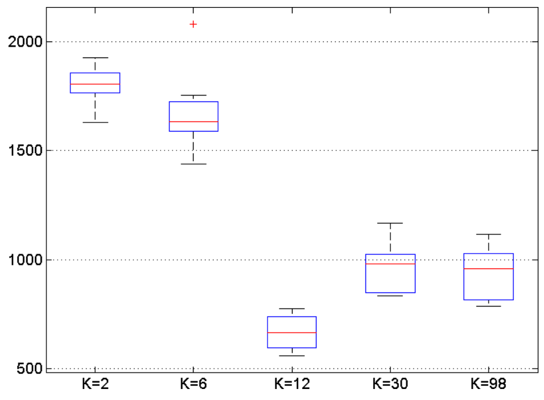
To verify how neighborhood size affects search performance, we consider regular ring networks with different values for K. In the following experiments, we use both Aggregation by Copy and Aggregation by Shuffle applied to the optimization problems described in Section 4.2. We report on experiments using $N = 100$ and $p_n = 0.01$, running each algorithm for 5000 rounds and performing 20 independent trials. We consider the following neighborhood sizes: $K = 2$, $K = 6$, $K = 12$, $K = 30$ and $K = 98$. These were chosen so as to represent a variety of scenarios, respectively a minimally connected network, a small number of neighbors, a moderate number of neighbors, a large number of neighbors and a fully connected network.

Figure 6.1 shows the results using Aggregation by Shuffle in two different moments of the algorithm applied to the Griewank function. Figure 6.1(a) shows the best function value after a long run, defined as $T = 2000$ rounds of the algorithm. The first thing we can observe is that K has a strong influence on how well the algorithm performs. The worse analyzed case ($K = 2$) provides solutions that are on average more than an order of magnitude worse than the best case ($K = 98$); more precisely, the best case provides solutions that are on average about 16 times better than those provided by the worst case. Moreover, after 2000 rounds, larger values of K always lead to solutions that are at least as good as those obtained with smaller values of this parameter; the best solutions are obtained for $K = 30$ and $K = 98$. It is not statistically significant the performance difference for these two values; the difference between $K = 30$ and $K = 12$ is somewhat small, but significant. Therefore, allowing a more dense network is beneficial to the search being performed after such large number of rounds.

However, Figure 6.1(b) depicts a different behavior for a smaller number of rounds ($T = 200$). For such shorter run, the best performance is actually obtained for an intermediate value of K (in this case, $K = 12$). Either setting K above or below such value leads to a decrease in performance. As in the previous case, the worse performance still happens for sparsely connected networks and the difference between the best and worse cases is still quite steep, but less so.



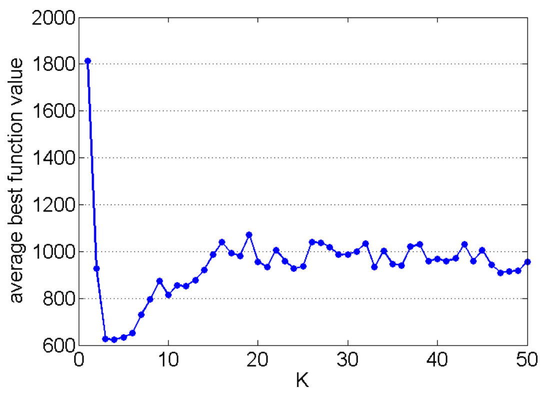
(a) T=2000



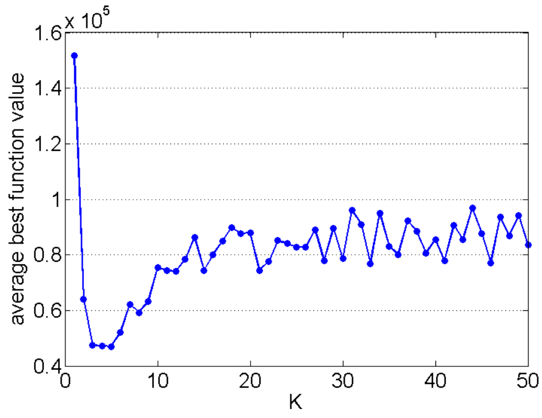
(b) T=200

Figure 6.1: Best function values when optimizing the Griewank 200-dimensional function using Aggregation by Shuffle and real-value encoding, with different neighborhood sizes and at different moments of the algorithm.

In order to better visualize this latter behavior, Figure 6.2 shows a complete plot for all possible values of K. A smooth transition between each performance level is present, with a clear optimum value for K being visible. In the same figure we also show that the same general behavior is present for the Sphere function. Applying the algorithm to any of the tested functions provide similar results, and the optimum value of K varies only slightly over these functions.



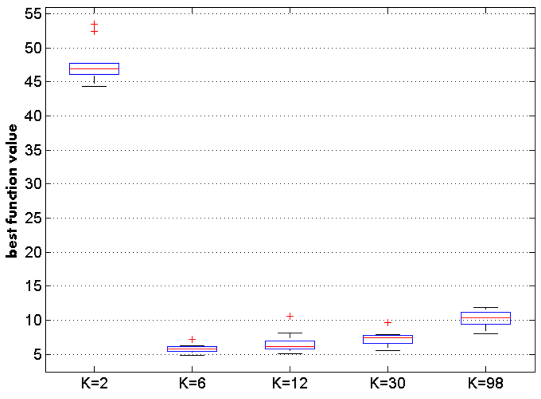
(a) Griewank function



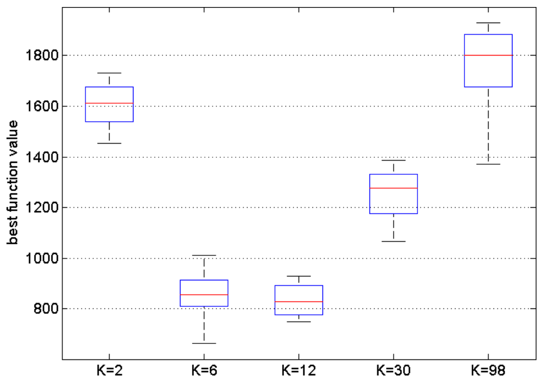
(b) Sphere function

Figure 6.2: Average best function value after 200 rounds for the full range of values of K using a 200-dimensional Griewank function (a) and a 200-dimensional Sphere function (b), Aggregation by Shuffle and real-value encoding. Averages are over 5 independent runs.

Using binary encoding instead of real-values leads to a similar behavior and the presence of an intermediate optimum value for K is present even in longer runs. Figure 6.3 shows the results using this encoding method. We can see that the differences in performance for different values of K become more extreme. While the worse situation in a long run is still to have a very sparse network, in shorter runs it is actually much worse to have a fully connected network. In both cases, the best case is to have an intermediate (but relatively small) number of neighbors.



(a) T=2000



(b) T=200

Figure 6.3: Best function values when optimizing the Griewank 200-dimensional function using Aggregation by Shuffle and binary encoding, with different neighborhood sizes and at different moments of the algorithm.

These results also make more evident that, for longer runs, the effect of K becomes less pronounced, since the difference in performance is much less accentuated for long runs. This is expected, since the algorithm guarantees convergence to the global optimum in the limit (i.e. for very large runs); hence, it must be the case that eventually the algorithm will reach the global optimum regardless of the value of K.

The relation between K and performance is clearly non-linear. This hints that K may be modifying some underlying network property that is responsible for such behavior. We can use the observed optimum in early rounds when using the Shuffling mechanism to try and assess the underlying property that influence the optimum. By changing K, we are not only changing the vertices' degree, but also several other properties of the network, such as its diameter, density, characteristic path length and clustering coefficient. Naturally, all these properties are dependent on K, but also on N, the number of vertices in the network.

Hence, by repeating the experiments with different network sizes, we can observe whether and how the optimum value of K moves. If it does move with changes in N, this is evidence that some network property is responsible for the observed behavior, rather than the vertices' degree (which is independent of N for regular networks).

Table 6.1 shows the location of the optimum value of K for several values of N, along with relevant global network properties, calculated for those specific values of N and K. We can see that the optimum does change with N; K_{opt} is shifted towards larger values as we increase the size of the network. We can also see that none of the considered global properties is kept constant. Nonetheless, it seems that K changes so as to maintain the following relation roughly constant:

$$\alpha = \frac{N}{K^2}$$

The values for α are shown in the last column of Table 6.1. We currently do not know why α seems to play a role in how efficient the search can be, since for the regular ring network no known global property scales inversely with K^2 (or, equivalently, proportionally to $\sqrt{N}$).

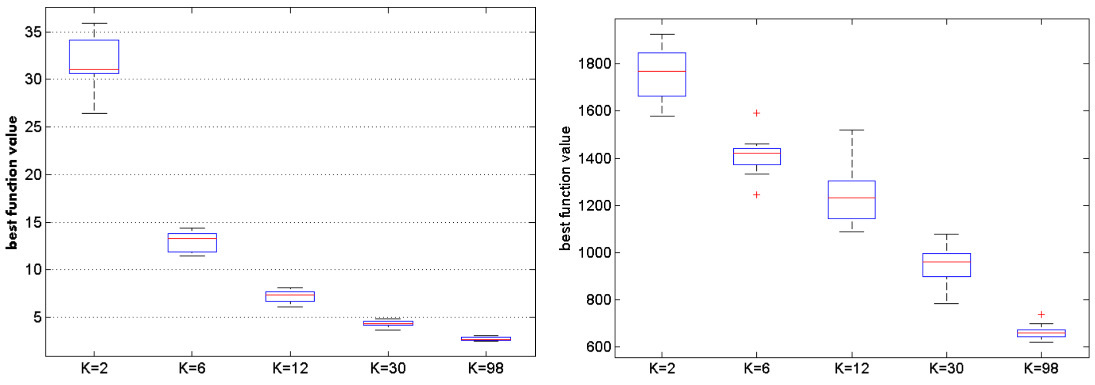
N	K_{opt}	**Diameter**	**Density**	**Path Length**	α
50	6	8.33	0.061	4.59	1.39
100	8	12.50	0.040	6.69	1.56
150	10	15.00	0.034	7.95	1.50
200	12	16.67	0.030	8.79	1.39
300	14	21.43	0.023	11.18	1.53
400	16	25.00	0.020	12.97	1.56
500	18	27.78	0.018	14.36	1.54
1000	26	38.46	0.013	19.71	1.48

Table 6.1: Optimum value of K for different network sizes when optimizing the Griewank function using the Aggregation by Shuffle method over binary encoding. Calculated values for several network properties that depend on N and K are also shown.

Switching to Aggregation by Copy, on the other hand, leads to a qualitatively different behavior in earlier rounds. Figure 6.4 shows the results of using this aggregation mechanism. For long runs we see the very same monotonic improvement in performance with increases in K, but now the same behavior is observed even for short runs. Hence, when using this aggregation method, we can do no better than using a fully connected network. Note that, in this case, the algorithm is reduced to a form of stochastic hill-climbing: the single best solution is replicated in all N nodes and each explore a neighbor of this solution, repeating the process at every round of the algorithm.

These results show that when using information from multiple sources, networks where nodes have a large number of neighbors are able to perform better than a moderately sparse network, but requires a large number of algorithmic rounds to do so. If only a moderate number of rounds is to be performed, having too much neighbors leads to a poor performance. If a copy mechanism is being used, however, having a highly connected network is the best strategy to obtain improved performance, independently of the number of rounds available. In both cases and at any round of the algorithm, having a very small number of neighbors is highly detrimental to performance.

In addition, we can see that for long runs, the best cases for both Aggregation by Copy and Aggregation by Shuffle (fully connected network) provide comparable solutions. However, for all other values of K, Aggregation by Shuffle provides a better performance. For shorter runs, when using the best value of K in each aggregation method, shuffling again provides an advantage but for other values of K the copy mech-



(a) T=2000 (b) T=200

Figure 6.4: Best function values when optimizing the Griewank 200-dimensional function using Aggregation by Copy and real-value encoding, with different neighborhood sizes and at different moments of the algorithm.

anism often performs better. Hence, unless the network is set close to the optimal value of K, in short runs the copy mechanism will on average outperform shuffling.

6.1.2 Effects of rewiring probability

We now turn our attention to another central aspect of the Small-World generative model, the probability of randomly rewiring connections of a regular network. As we have discussed previously, such random rewiring allows for shortcuts to be created between otherwise distant parts of a network, drastically reducing its characteristic path length. The amount of rewiring controls a transition between a fully structured network and one that possess properties of a random network.

Our methodology is the same as the one used in the previous section. With a MNA starting from a regular ring network, we vary the probability of rewiring (β), recording the best found solution for each case. We consider only networks that are connected. In order to do so, we discard any disconnected network generated by the rewiring process[1].

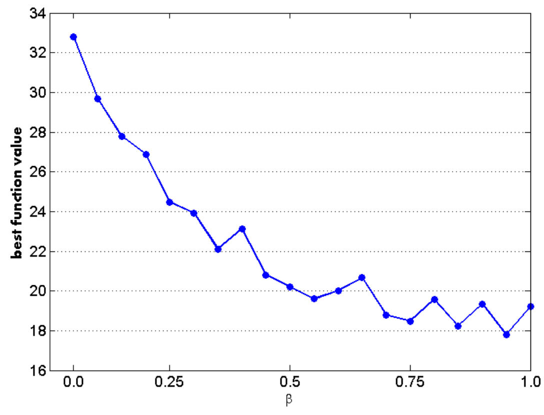
We must, additionally, specify the network density (i.e. K). In the previous section we observed that K can have a strong impact on how well the algorithm can perform. Therefore, we must experiment with random rewires in different performance regimes associated to different values of K. We experimented with the same values of K used in the previous section: $K = 2$, $K = 6$, $K = 12$ and $K = 30$. The fully connected network was naturally not considered, since random rewires do not change the network in any way. In the following experiments, the initial network is again a regular ring network with 100 nodes and $p_n = 0.01$.

We varied β, the probability of rewiring each edge of the network, from 0.0 to 1.0 with 0.05 increments. For $\beta = 0.0$ the network is kept completely regular, i.e. no

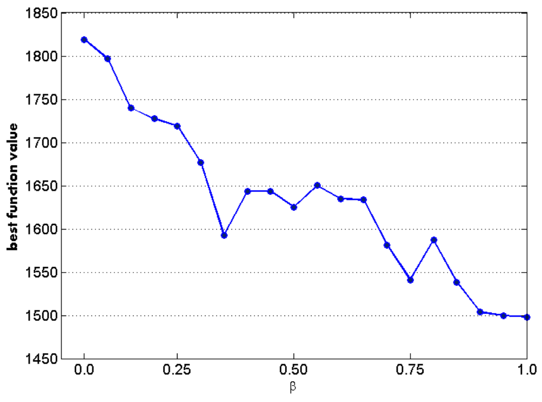
[1]Disconnected networks are nonetheless interesting, as they allow for sub-graphs to search independently from one another, in a similar fashion as Island models in Evolutionary Algorithms (FOGEL, 2000).

rewiring is performed. For $\beta = 1.0$ all edges are randomly rewired, leading to a close approximation to a random network.

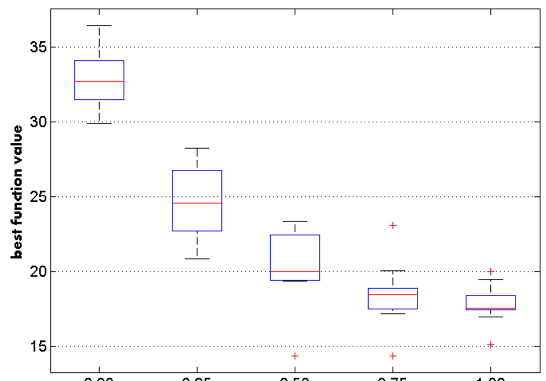
Figure 6.5 shows results for $K = 2$. We can observe that both in the long and short runs the fully structured network performs poorly and better solutions are found as β is increased. A random network was found to provide the best performance in any round. The performance differences between the two extreme cases is again quite accentuated - a random network is able to find solutions that are on average about 83% better than a structured network - evidencing that this parameter too has a strong influence on how effective a search can be. This behavior is essentially unaltered for different functions or when using Aggregation by Copy or binary encoding.



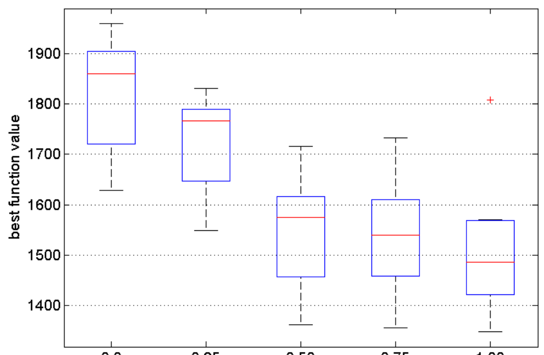
(a) T=2000



(b) T=200



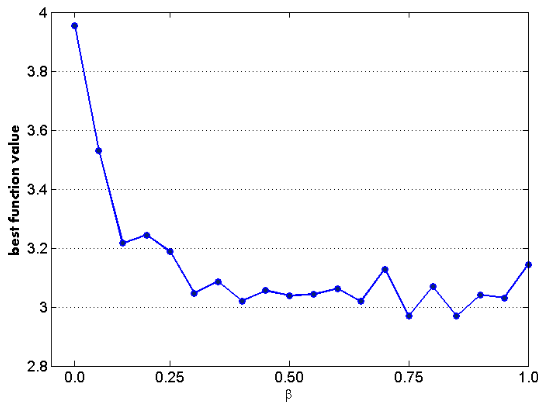
(c) Boxplot for T=2000



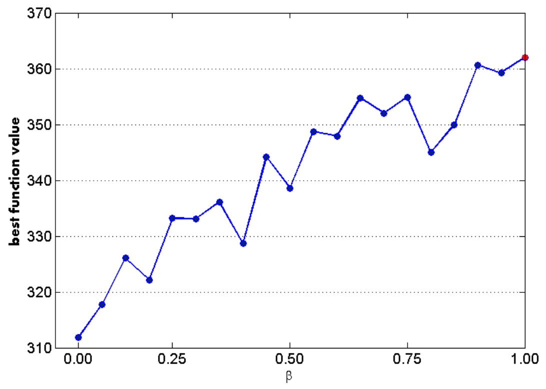
(d) Boxplot for T=200

Figure 6.5: Average best function values using Aggregation by Shuffle applied to the Griewank 200-dimensional function and different values of β and at different moments of the algorithm. Boxplots shown for selected values of β. Network density is set at $K = 2$.

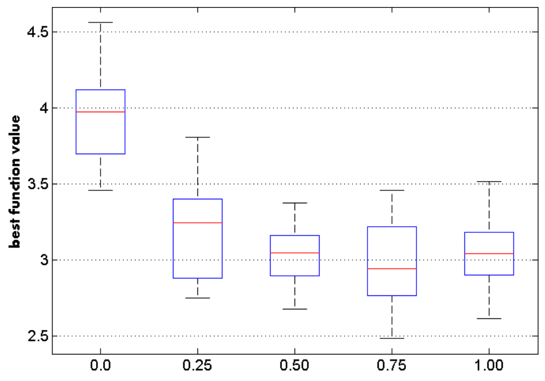
For $K = 6$, shown in Figure 6.6(a), in the long run we observe a similar behavior, but now effectively for $\beta > 0.35$ no statistically significant differences in performance is detected. Also, the average difference between solutions found for $\beta = 0.0$ and $\beta = 1.0$ is of about 30%, much smaller than the one measured for $K = 1$. This is evidence that the influence of β decreases with increases in K. This is to be expected,



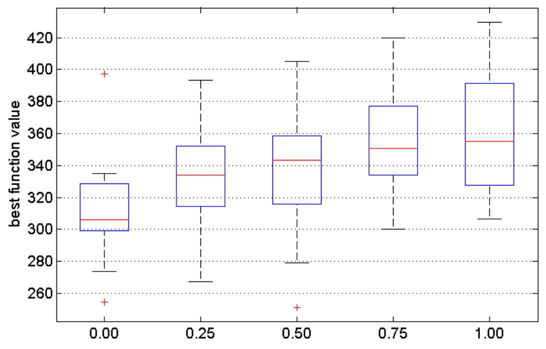
(a) T=2000



(b) T=200



(c) Boxplot for T=2000



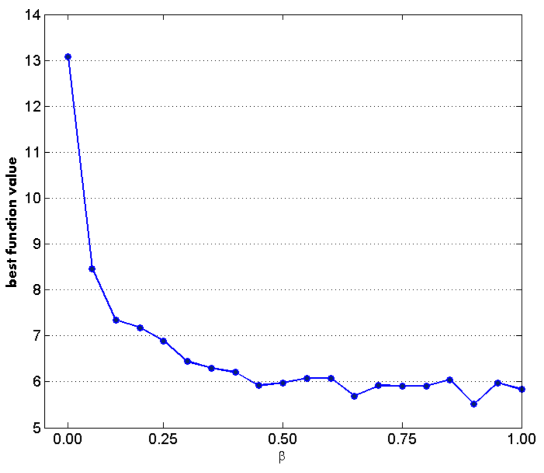
(d) Boxplot for T=200

Figure 6.6: Average best function values using Aggregation by Shuffle applied to the Griewank 200-dimensional function and different values of β and at different moments of the algorithm. Boxplots shown for selected values of β. Network density is set at $K = 6$.

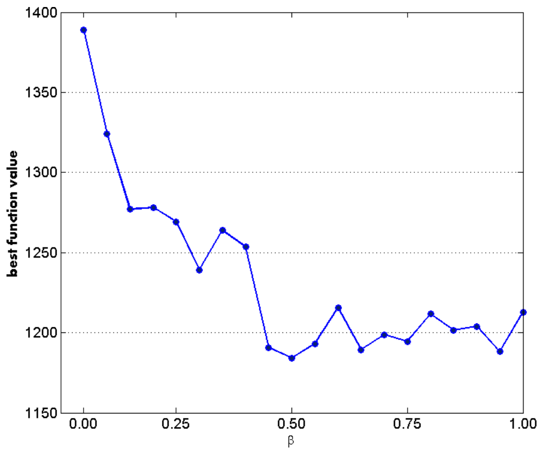
since in the limit, for a fully connected network, β cannot have any influence (random rewirings cause no changes in the network).

Interestingly, for $K = 6$ the observed behavior is effectively reversed when considering shorter runs. Figure 6.6(b) shows that increasing β actually *decreases* the measured performance. The differences in performance are somewhat small (at most about 15%), but significant. This is only observed when aggregating from multiple sources (the same happens when using Aggregation by Average or binary encoding), but not when using Aggregation by Copy, which provides a monotonic increase of performance with increases in β in all rounds of the algorithm (see Figure 6.7); also, performance differences are much more accentuated when using the copy mechanism in the long run, showing that such aggregation method is considerably more affected by β.

Why would the general behavior be effectively reversed for shorter runs? The answer relies on how better solutions are distributed on the search space. In initial rounds, it is easy to find better states, since we start with random solutions which are poorly



(a)



(b)

Figure 6.7: Average best function values using Aggregation by Copy applied to the Griewank 200-dimensional function and different values of β and at different moments of the algorithm.

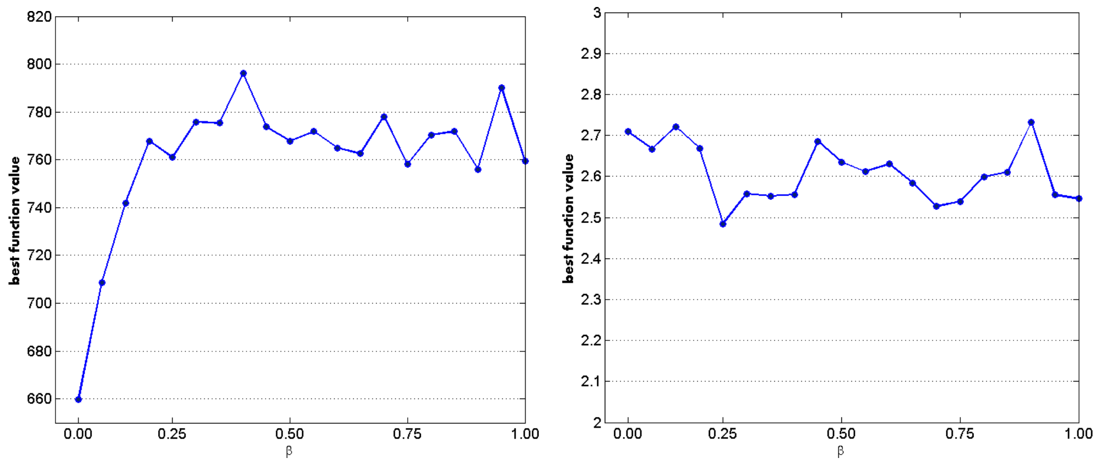
evaluated. Hence, on average, it pays to explore as much as possible the search space since the probability of finding increasingly better solutions is high. However, as the algorithm progress, finding better solutions becomes harder and exploring anything but the neighborhood of good known solutions is, more often than not, fruitless.

As we increase K, we are reducing the distance between nodes; this allows for good solutions found anywhere in the network to spread faster to all nodes. Therefore, by increasing K we are increasing exploitation. On the other hand, smaller neighborhoods mean that any information will take time to spread, allowing for a greater independence of the nodes, which effectively allows them to explore relatively independent areas of the search space.

As we have previously argued, the shuffle mechanism favors exploration, since the worse nodes will combine information from a large number of sources, which leads to large changes in the solution being aggregated. This is in contrast with the copy mechanism, which does not generate new recombinations by aggregation; rather, it relies on the appropriation step only to produce new solutions. Hence, when using this latter mechanism, even in a short number of rounds

Making $K = 12$ still leads to a situation where short and long runs differ in how they behave as β is changed. Figure 6.8(a) shows that in earlier rounds increasing β decreases performance, but for $\beta \geq 0.25$ the differences are not significant. Finally, for $K = 30$, β has no influence whatsoever on performance, as shown in Figure 6.8(b).

As in the last section, we have a case where improving communication between nodes in the network leads to an improvement in performance. For long runs, as we increase β, the network's Characteristic Path Length (L) decreases quickly, reducing the average number of intermediate nodes between any two nodes, hence allowing for information to spread faster. However, the same is not always true for shorter runs. As



(a) K=12, T=200 (b) K=30, T=2000

Figure 6.8: Average best function values using Aggregation by Shuffle applied to the Griewank 200-dimensional function and different values of β and $K = 12$ (a) and $K = 30$ (b).

we have shown, for shorter runs the behavior can be precisely the opposite and reducing L leads to a decrease in search performance. As with neighborhood size, this peculiar behavior is only present when aggregating information from multiple sources - when using the copy mechanism, shorter path lengths always lead to improved performance.

We have applied the same experiments to all other proposed benchmark functions and found that the same behavior is present at all of them, independently of the function's modality or separability.

6.1.3 Discussion

The results in this section show that different SWNs generated using diverse parameters for the Small-World generative model can have strikingly different performance. We observed that when there are still a high probability of finding better solutions, as in early rounds of a search, it is beneficial to allow a wider exploration. This is obtained by reducing path lengths in the network, so that information is spread slowly through vertices, by either reducing the number of neighbors for each vertex or by restricting the number of shortcuts allowed. However, as less better solutions are available, exploiting the few good solutions is the best choice. Hence, in long runs, allowing as much communication between nodes always lead to better performance, since we decrease the number of steps for a good solution to reach all vertices.

In any case, having an excessively sparse network with very high characteristic path length always performed poorly, which constitute further evidence on the benefits of allowing some communication between independent searches.

6.2 Search in Scale-Free Networks

In this section, we provide results of experiments on using MNAs over Scale-Free Networks (SFN). Like in the previous section, where we focused on Small-World Networks, we are interested in generating networks that possess SFN properties and observe how parameters of such networks affect search performance.

We use the SFN generative model described in Chapter 3. Starting with a small number of connected vertices, we add N new vertices one by one; each new vertex creates M edges, probabilistically connecting to existing vertices with a bias towards those already well-connected. Hence, the network density is defined by M and the preferential attachment mechanism allows for a power-law distribution of degrees to emerge.

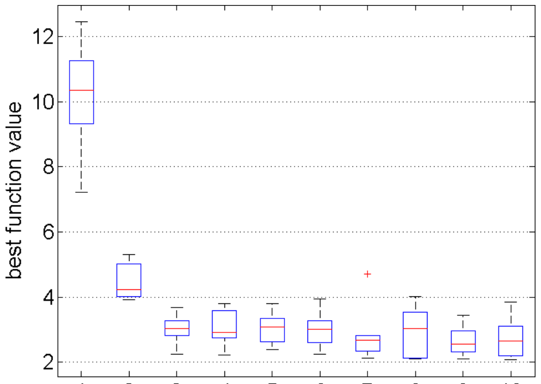
Our first parameter of interest on this network model is naturally the network density. However, SFN's main characteristic is the existence of *hubs*, which are highly connected vertices. Hence, we are interested in verifying the utility of such hubs during a search. In order to so, we shall allow *egalitarian networks*, where the size of hubs are limited, by setting a parameter H ($H > M$) that specify the maximum number of edges that any vertex may be part of at any time.

6.2.1 Effects of density

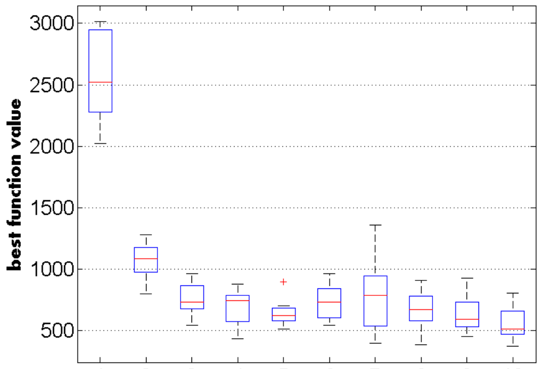
The parameter M controls the number of edges created by new vertices entering the network. It therefore directly controls the network density. Unlike the neighborhood size (K) in regular networks, it does not directly control the vertices' degree, since we have an uneven distribution of the newly formed edges. This parameter does, however, set the *minimum* degree of a node - every vertex must be part of M edges, even though later on they may attract more edges to themselves. Therefore, the maximum degree in the SFN considered here is $N - 1$ and the minimum, M.

We tested varying M inside a reasonable range and observe how this parameter affects the search performance. Note that for $M = N$, the network becomes fully connected due to the way the generative model works (it starts with M fully connected nodes), which is not a case of interest here. Rather, we keep M relatively low so that the power-law distribution is maintained. We thus used $1 \leq M \leq 10$ using a network with 100 nodes.

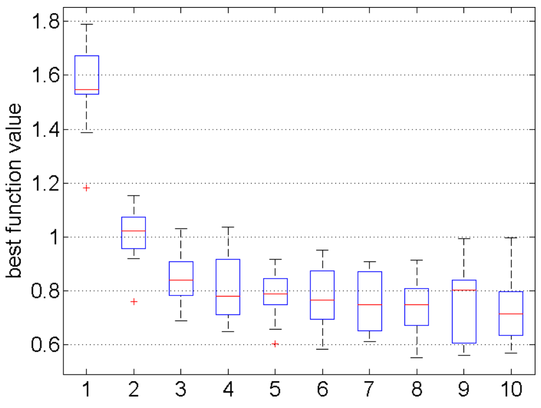
Figure 6.9(f) shows results over different functions and 2000 iterations of the algorithm. We show here only results using Aggregation by Shuffle; when using the copy mechanism or binary encoding, the basic qualitative behavior is unchanged. A behavior comparable to the effect of K in Small-World networks is visible, as low density leads to poor performance and increases in M quickly leads to better solutions until a saturation point is reached, after which no further gains are attained by increasing M. Just like in Small-Worlds, in Scale-Free networks, denser networks provide better performance, but the gains are diminishing - a moderately dense network provides the same performance as a much denser network. Indeed, what we observe is that for most part of the tested range, M does not influence performance at all, with the exception of



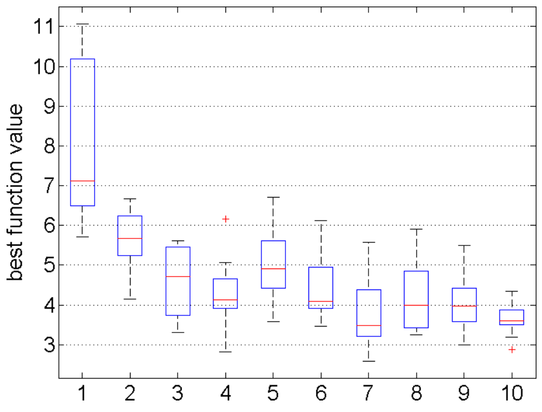
(a) Sphere function



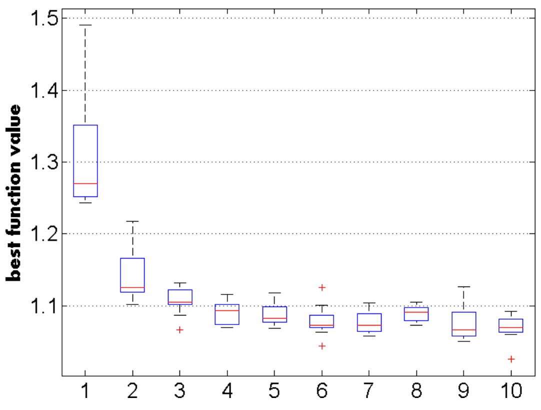
(b) Hyper-Ellipsoid function



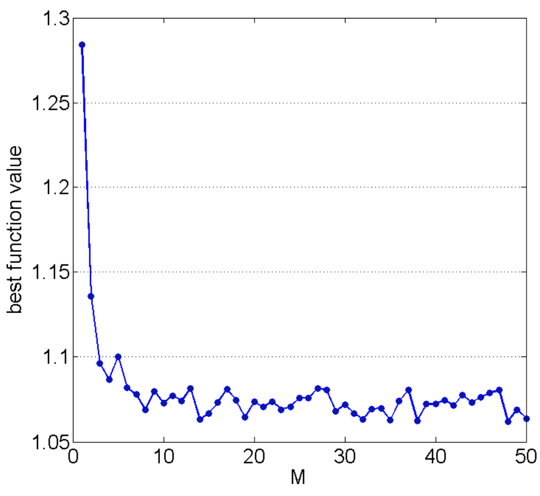
(c) Ackley function



(d) Schwefel function



(e) Griewank function



(f) Average for Griewank function

Figure 6.9: Boxplots for the best function value after 2000 rounds using different functions and different values of M. Aggregation by Shuffle is used on a network with 100 nodes and $p_n = 0.01$. In (g) the average over a more extensive range of values for M is shown.

very sparse networks: for $M \geq 4$ differences are not significant.

We can also observe that unimodal functions are more influenced by M than multimodal functions. For the Sphere and Hyper-Ellipsoid functions, the average best solution ($M = 10$) is a little over 4 times better than the worse solution ($M = 1$), while for the other functions the average best solution evaluation does not exceed 2 times the evaluation of the worse. For shorter runs and different functions we found no qualitative differences in behavior.

However, the observed influence of M on performance is very small when compared to the influence of K in Small-World networks. Both parameters have a direct impact on the network density and nodes' degree and are, thus, comparable. Variations in K lead to performance differences that were up to an order of magnitude, but variations in M in Scale-Free networks did not have such impact. The reason for that seems to be that in SFNs the overall characteristic path length is already very small, even for small values of M. While in regular ring networks networks a small K was associated with high diameter and path lengths, the structure of Scale-Free networks allows for shortcuts to occur naturally. Indeed, networks formed by using the generative model employed here have also the small-world property[2]. Moreover, the model requires that only a minimum degree is set for nodes; while the majority of nodes will indeed have a degree that is close to this minimum, some nodes will be part of many more connections.

Nonetheless, these results are in accord with the results for Small-World networks: networks with sparsely connected nodes perform poorly, but performance is quickly increased by even small increases in density. Continuing to increase density leads to a case of diminishing returns in both network types.

6.2.2 Effects of hubs

In the previous section, we allowed hubs to grow without limits - it was possible for a node to be connected to all other nodes in the network. Unlimited sizes for hubs are possible in a number of scenarios (e.g. the network composed of websites and their hyperlinks - it is possible for all sites to link to one single site), but often there will be a limit over which a node cannot receive further connections. SFN whose hubs' sizes are limited are called *egalitarian*, because the connections that would otherwise be given to a hub will go to other nodes. For example, the network composed of air routes between airports is egalitarian: an airport can only handle a certain maximum number of flights - once this maximum capacity is reached, flights have to be routed to other airports.

We define a parameter H that specifies the maximum number of edges any node can have at any time. Certainly, H must be greater than M and at most $N - 1$. This parameter controls how egalitarian is the network; the lower is H, the more egalitarian it becomes. On the other side of the spectrum, a network is said *aristocratic* if hubs can grow without limit (or at least very large). We tested networks with different values of

[2]This is not always the case, though. Some generative models lead to networks that possess the scale-free property but have low clustering and long path lengths(BARABÁSI, 2003).

H with $M = 2$.

Note that the smallest possible value for H is $H = 2M$, so as to guarantee that new entering nodes will have available nodes to connect to (Figure 6.10 shows an example). Figure 6.11 shows the results using real-value encoding, Aggregation by Shuffle and different values of M. Only for small values of H does M influence the resulting search performance. For $H \leq 3$, no significant differences can be measured between any two values of M. For $H = 1$ and $H = 2$ we can observe that small values of M leads to a very bad performance, which improves quickly as we increase M. Further increasing M beyond a relatively small value do not continue to improve performance.

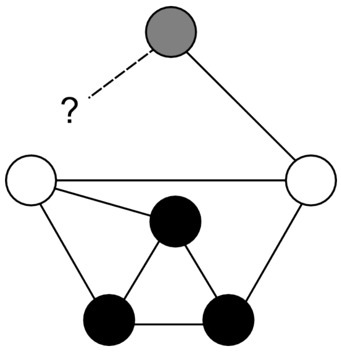


Figure 6.10: Graphical representation of the initial steps of the generational model for SFN using $M = 2$ and $H = 3$ (below the required $H = 2M$). The black nodes are the initial nodes. The fourth and fifth node (in white) are able to enter the network, as there are still connections available in the initial nodes. The sixth node (in gray), however, is able to create the first of its two connections, but the second cannot be made since all nodes are at their maximum degree.

The largest difference in any of these two cases happens when switching from $H = 2M$ to $H = 2M + 1$. Other differences are much smaller or non-existent. By analyzing the emergent network for $M = 2$ and $H = 4$ and comparing to a network with $H = 5$ we can understand better why performance is so different. Figure 6.12 shows two typical networks generated with these parameters. We can see that the way the generative model works lead, for $H = 4$ to a network that is very regular, close to a grid. By increasing H by a single unit, a very different network emerges, without regularities and many shortcuts between different parts of the network. In fact, the network for $H = 4$ is not scale-free at all: all nodes have approximately the same degree (4, in this case). For $H = 5$, however, there is enough connections available so that a few nodes can attract a larger number of edges, leading to a few (small) hubs; this results in a case where most nodes have a 2 connections and a few are part of 5 edges.

Qualitatively, networks with $H \geq 5$ are indistinguishable. The main difference between these networks is precisely the maximum size of the hubs, but all of them present a power-law distribution to some extent. Hence, the scale-free property seems to be beneficial to the search process, but the precise value of H plays a somewhat limited role, as very small improvements in performance are measurable for $H >=$

$2M + 1$.

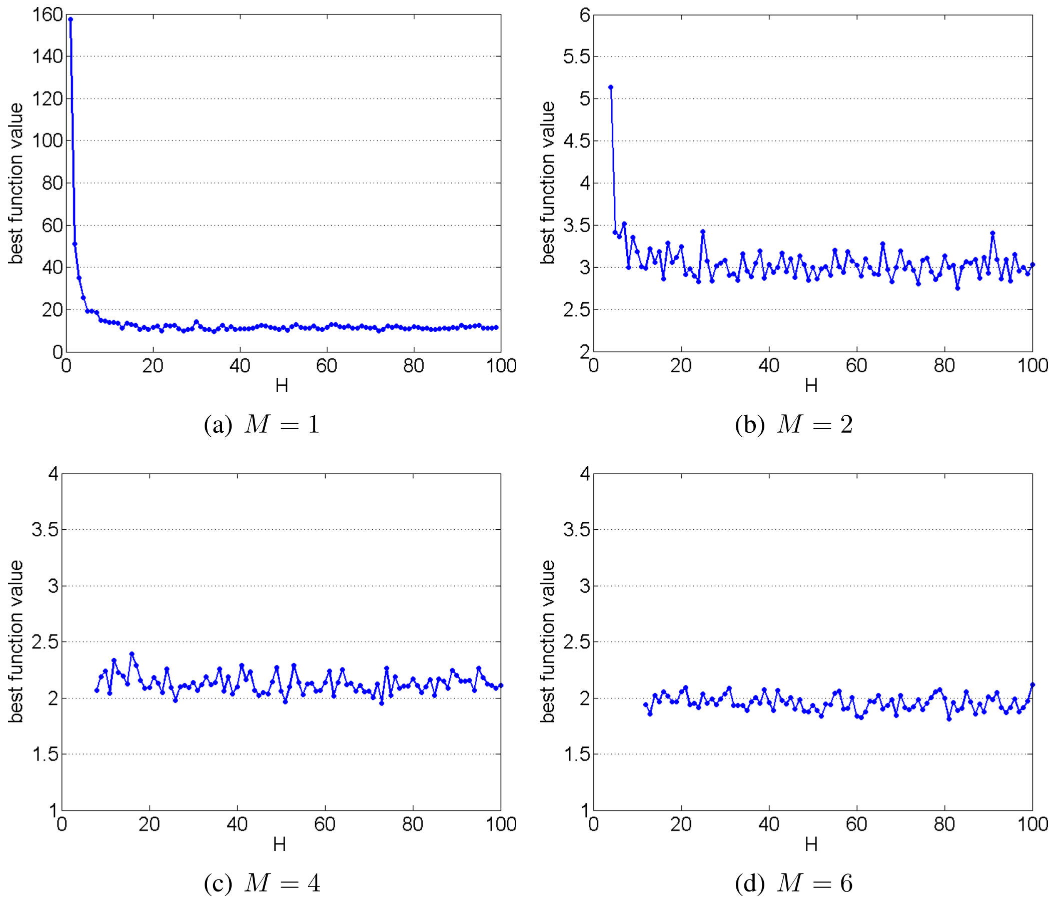
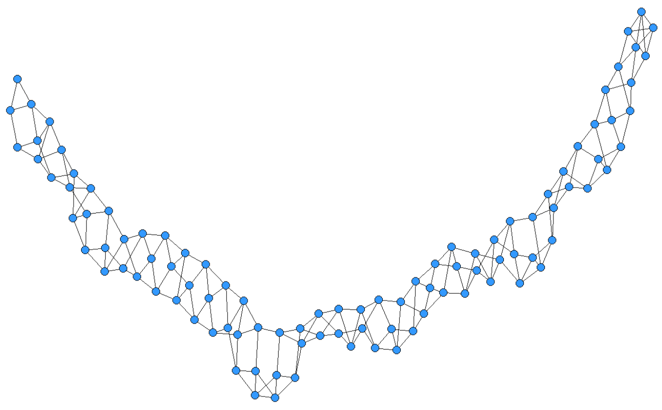


Figure 6.11: Best function value for a 100-dimensional Griewank function and different values of the maximum degree for the network's vertices (H).

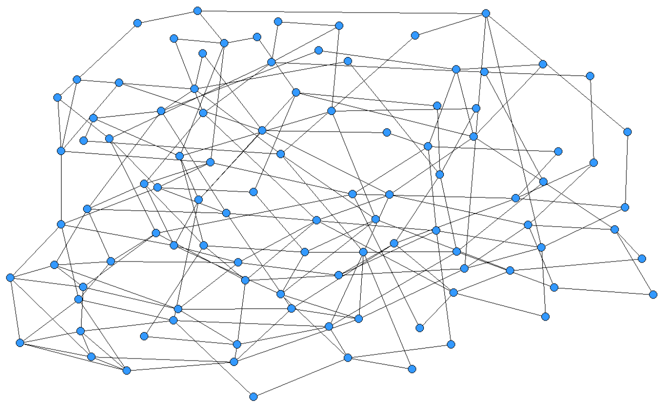
6.2.3 The role of location in SFN

One could ask if the position of a node in the network matters for its success. Several studies argue that being well positioned (i.e. well connected) in any social network is a major advantage for matters of e.g. collecting social capital, electing trust or leveraging influence over others. In the context posed in this thesis, we could ask whether the position of a node in the network influence its search performance. Since we do not take into account geographical positions, the role of a node in a network is uniquely defined by its connections. The most relevant property of a node in SFN is arguably its degree. Hence, the problem can be posed as a relation between a node's individual search performance and its degree.

In order to tackle this issue, we must abandon, momentarily, our general approach of considering the whole network as composing a single entity responsible for performing search and focus on the performance of individual nodes. This is straightforward, since each node has its own associated solution.



(a) $H = 4$



(b) $H = 5$

Figure 6.12: Emergent networks using the Scale-Free generative model with a forced limit on the maximum degree (H) and $M = 2$. Plots generated by maximizing the distance between nodes with maximum edge length (MCGRATH; KRACKHARDT; BLYTHE, 2003).

We proceed as follows. We generated a (aristocratic) SFN network with $N = 500$, $M = 2$ and kept it fixed. Over this network, we ran 20 independent trials of 1000 rounds each, registering for each trial and each round the evaluation of each node, along with its degree. We then averaged the evaluations over the trials and *also over the rounds*, in order to get the average evaluation of each node throughout a full run.

Figure 6.13 shows the nodes' evaluations when optimizing the 100-dimensional Griewank function against their degrees. It is possible to see that the worst performance is associated to a very low degree, while the best performance is associated to a high degree. The correlation coefficient between these two parameters is of -0.497, which is a medium correlation (TAMHANE; DUNLOP, 1999). When considering only the average performance at the 1000th round (without averaging over all rounds), the correlation is still moderate, at -0.332. When using Aggregation by Copy, the correlation is only slightly stronger, at -0.510 and is essentially unchanged when optimization is applied to other functions.

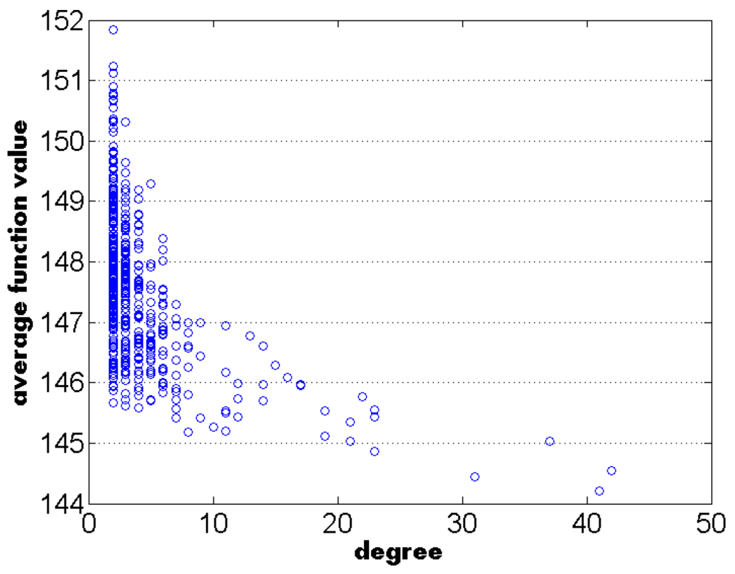


Figure 6.13: Average function value for each node in a SFN consisting of 500 nodes plotted against each nodes' degree. Each point is the average evaluation of 1000 rounds and 20 independent trials. The correlation coefficient is -0.497.

This result shows that even though all nodes have the same internal capability (i.e. they run the same algorithm), their position alone in the network can predict how well they will perform in an individual basis. Just like most other results so far, we have a positive correlation between a node's performance and its connectedness.

6.3 Discussion

We have observed in this chapter that both density and hubs' sizes can affect the search being performed over a SFN, but differences in performance were much smaller when compared to the differences observed for SWNs, an evidence that SFNs are somewhat more robust to changes in its parameters (even though this robustness comes at the cost of losses in performance, as we will show in Chapter 8). In similar fashion to SWNs, increasing the network density leads to a direct increase in performance. Unlike SWNs, however, we found no cases where sparser networks provided better performance. Simpler, unimodal, functions were found to be more susceptible to changes in the network's density, but otherwise the same qualitative behavior was present. Changes in the maximum node degree were only significant for very sparse network; for even moderately dense networks this parameter did not affect performance at all.

Therefore, when using the SFN model, we can do no better than trying to design dense networks in order to improve search performance. However, the gains are only considerable when improving from an extreme case of very sparse network. By allowing vertices to create as few as $M = 4$ connections, any further increases are essentially irrelevant to the search being conducted. For that density, the choice between an aristocratic and egalitarian network is also irrelevant. For sparser networks an aristocratic network always provided better solutions; the significant correlation between a vertex degree and its performance is evidence that the improvement in performance for aristocratic networks is a direct product of the existence of hubs.

7 SEARCH IN DYNAMIC NETWORKS

In this chapter, we turn our attention to dynamic networks, those networks whose edges may change during a search. This is in contrast to the networks we have dealt so far, where a well-known network topology was imposed upon the algorithm. Now, we allow for an MNA to execute the Connection Step at each round of the algorithm, making it possible for vertices to create and destroy edges at any time.

The Connection Step is now not concerned with an overall topology and is rather egocentric - each node can decide by itself to create connections as it pleases. Nonetheless, each edge must reflect in some way the benefit of creating such connection between nodes. That is, an edge is created if there is a perceived benefit of doing so for a node. Hence, we define a general rule for connections to be created, which take into account only the relative benefit of doing so, without concerns on the impact on network topology.

The above approach requires that we deal with asymmetric, directed, edges. Therefore, unlike in previous chapters, we have that if (a, b) and (b, a) are edges and E is the set defining all edges, than it is *not* the case that $(a, b) \in E \rightarrow (b, a) \in E$. This requirement becomes clear after defining the Connection Step:

Definition *(Dynamic Connection Step)* Let E be the set of edges and V the set of vertices in the network. Then, $E = \{(u, v) | u, v \in E \wedge eval(v) \succ eval(u)\}$.

The idea of this connection step is that a node can only benefit from other nodes that are better evaluated than itself. This simplistic method is reasonable, since we are to expect that sources that are better off than ourselves have information that may benefit us - few people would seek financial advise from an economist that went bankrupt, for example.

By allowing unrestricted connections to be made, we can observe the emergence of two types of hubs. The first hub, which we will call *suppliers*, are well-evaluated nodes that receive many connections - i.e. they have a high in-degree[1]. If a single best node exist in the network, for instance, it will receive connections from all other nodes. These nodes have a high influence over the network, acting as major distributing hubs.

[1]Note that in our notation an edge (u, v) implies that node u have a directed edge towards v, but information flows in the other way, from v to u.

They will also have a low out-degree since, being well-evaluated, there will be fewer nodes better evaluated.

The second type of hub is composed of *sinks*, which are nodes with a high out-degree. These are badly evaluated nodes, which create connections to many other nodes. If a single worse node exist in the network, it will have connections to all other nodes. These nodes do not influence the network as much (since they typically will have a low in-degree), but can receive information from , and thus are influenced by, many other nodes.

This is similar to our previous discussion on Scale-Free networks. When using an aggregation method that use information from several sources, the worse a solution is, the more it will be modified. When using the copy mechanism, however, the worse a node is the more options it will have to choose a source to copy information from. Good solutions, on the other hand, are only slightly modified: the best solution is only modified by the Appropriation Step, while other slightly worse solutions will aggregate information from a few other nodes, hence keeping mostly intact the information contained in them.

7.1 The Effects of Limiting In-degree

As previously defined, suppliers are nodes with a large number of incoming connections and, thus, providers of information to a large number of other nodes. Up until now, our dynamic model allows nodes to receive an unlimited number of connections. Hence, any time a node try to connect to another node, this connection is granted. Since nodes only connect to better evaluated nodes, this means that good information is available to all other nodes in the network, without restrictions.

We are interested in studying how limiting the number of connections a node can receive affect the search performance. In order to do so, we establish a parameter $maxIn$ that defines the maximum number of incoming edges a node can support, i.e. its maximum *in-degree*. Whenever this number of edges is reached, no further connections are allowed. Hence, we force a restriction on the influence of well-evaluated nodes and the maximum size of suppliers. Note that unlike our experiments with Scale-Free networks, this does not necessarily imply an egalitarian network, since a connection that is denied is not forwarded to another node, it simply ceases to exist.

In order to implement this new parameter, we have to specify which nodes are granted connections. This is specified in the Connection Step and it proceeds as follows. If $n < maxIn$ vertices are trying to connect to a vertex v, then all requesters are granted the connection. If $n \geq maxIn$, then $maxIn$ vertices among these n are chosen uniformly at random and granted the connection. We can then rewrite the Connection Step as follows:

Definition *(Dynamic Connection Step (with maximum in-degree))* Let E be the set of edges and V the set of vertices in the network. Then, $E = \{(u, v)|u, v \in E \wedge eval(v) \succ eval(u) \wedge indegree(v) < maxIn\}$, where $indegree(v)$ returns the number of incoming connections of vertex v. Each edge (u, v) is added to E in a random order.

The imposed in-degree restriction effectively limits how large suppliers can become, restraining their effects on the network by limiting their reach. Since suppliers are the best evaluated nodes in a network, we are setting limits on the flow of good information. We are interested in observing the influence of $maxIn$ in search performance, which can be done by varying this parameter and observing the resulting performance.

Figure 7.1 shows the best function values for different maximum in-degree values and different moments of the algorithm using the above described Connection Step. From the boxplots shown in Fig. 7.1(a), we can observe that, after 2000 rounds, intermediate values of *maxIn* provide similar performance; the differences for $maxIn = 10$, $maxIn = 50$ and $maxIn = 90$ are not statistically significant. The two extreme cases, $maxIn = 1$ and $maxIn = 100$, however, severely deteriorates performance.

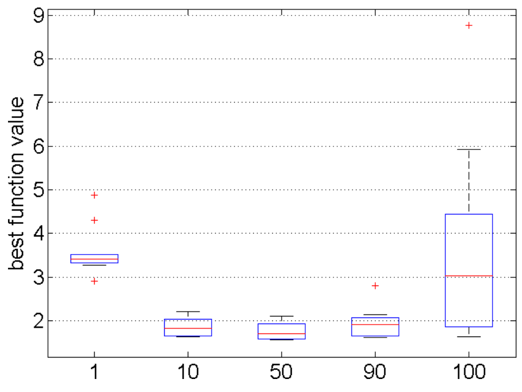
In the lower end of the spectrum ($maxIn = 1$), the MNA algorithm is unable to reach very good solutions at all, but the upper bound on the worse solutions is tight and close to the median. On the other end of the spectrum ($maxIn = 100$), the algorithm is able to reach reasonable solutions eventually (the lower percentile and median are much better than for $maxIn = 1$); however, the upper percentile is much worse than the former case. Hence, for unlimited in-degrees, the quality of solutions vary in a wide range, resulting in a poor performance on average. The intermediate cases are better in all measures: they provide a lower deviation, much better median and better best solutions than any of the two extreme cases. Figure 7.2(a) plots the average best function value for a wider range of $maxIn$, showing that performance deteriorates very quickly as we approach the extreme values of this parameter.

For earlier rounds (Fig. 7.1(b)), the main difference is that a low value for $maxIn$ $(= 10)$ provides the best performance. The extreme cases still perform poorly, but now high but non-extreme values also provide poor solutions. Figure 7.2(b) shows that there is a sharp optimum value for $maxIn$; increasing or decreasing this parameter quickly degrades performance.

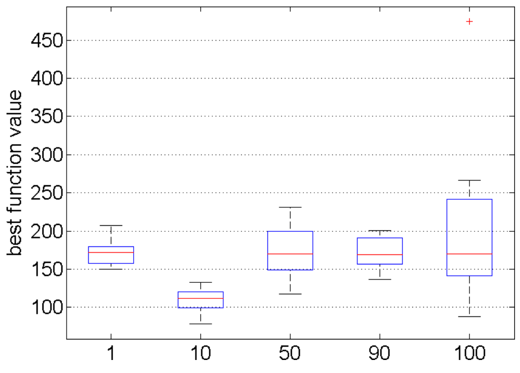
Comparing a full run for the best and worse values of $maxIn$ provides insights on the reasons behind the observed differences in performance. Figure 7.3 shows 1000 rounds of the algorithm using both $maxIn = 10$ and $maxIn = 100$. What we see is that in very early rounds using $maxIn = 100$ actually performs better than the more restricted case, but in the long run $maxIn = 10$ outperforms the former case by a large margin.

An explanation for this behavior that is consistent with the networked nature of the algorithm is that for $maxIn = 100$, we maximize how much good information is spread through the network; at any moment that a node finds a solution that is better than any other solution, all nodes have immediate access to this information. In other words, we are encouraging *exploitation* of good solutions. Hence, we expect that in early rounds convergence is faster. However, as the algorithm progress, local optima are reached and must be overcome. Therefore, in later rounds, it pays to reduce exploitation and improve exploration, which is enabled by restricting access to the best solutions so far.

These results are evidence that during a search, it is not beneficial to spread good information as much as possible. The best solutions were obtained when we restricted the influence of good nodes in the network, by limiting the number of nodes these well-evaluated solutions could reach directly. However, restricting excessively the influence of such nodes is also detrimental to search performance.

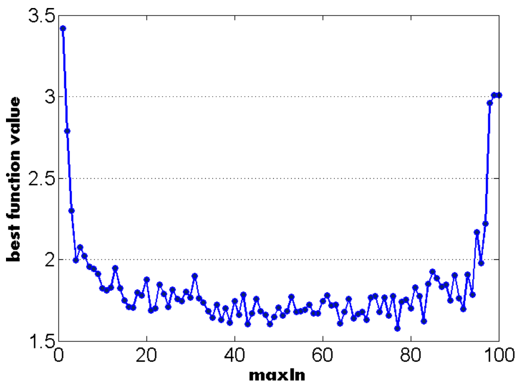


(a) $T = 2000$

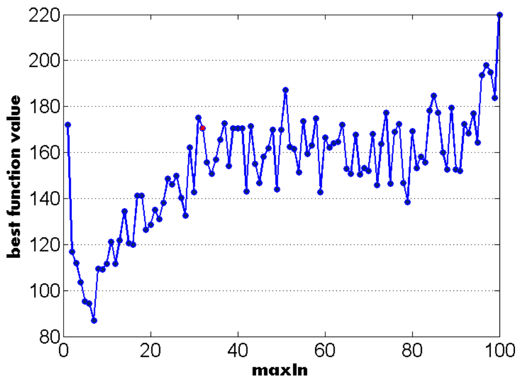


(b) $T = 200$

Figure 7.1: Best function values for different maximum in-degree values. Results obtained when optimizing a 100-dimensional Griewank function using a dynamic network with 100 nodes, $p_n = 0.01$ and Aggregation by Shuffle. Boxplots are shown for 20 independent runs.



(a) $T = 2000$



(b) $T = 200$

Figure 7.2: Best function values for different maximum in-degree values. Results obtained when optimizing a 100-dimensional Griewank function using a dynamic network with 100 nodes, $p_n = 0.01$ and Aggregation by Shuffle. Each point is the average over only 5 independent runs and is shown here to give a rough idea of the behavior over a wider range of values.

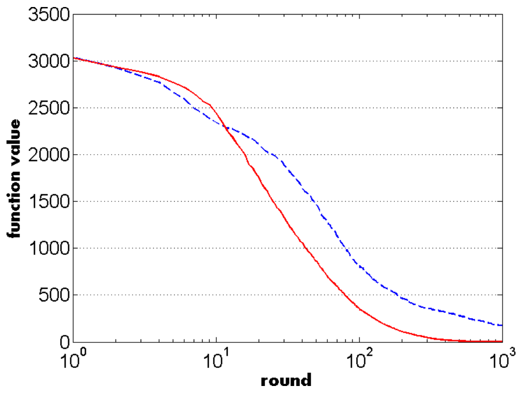


Figure 7.3: Best solutions found for a 100-dimensional Griewank function for 1000 rounds of the algorithm using $maxIn = 100$ (dashed line) and $maxIn = 10$ (solid line). Aggregation by Shuffle is used in a network with 100 nodes and $p_n = 0.01$.

7.2 Effects of limiting out-degree

Limiting the nodes' in-degree restricts the number of incoming connection and, thus, the number of nodes that any single node can send information to. A different restriction is also possible, namely the maximum number of nodes that any single node can be receive information *from*. In our model, this is accomplished by setting a limit in the number of outgoing connections that can be formed by a node - i.e. a maximum out-degree.

We modify our basic dynamic algorithm to include a new parameter $maxOut$ which specifies the maximum out-degree possible for any node. Without this parameter, nodes can connect to all nodes that are better evaluated. In a similar fashion to $maxIn$, if a node tries to connect to $n > maxOut$ nodes, then $maxOut$ nodes are chosen uniformly at random from among the n nodes. All connections that were not selected are simply discarded. We can then rewrite the Connection Step as follows:

Definition *(Dynamic Connection Step (with maximum out-degree))* Let E be the set of edges and V the set of vertices in the network. Then, $E = \{(u,v)|u,v \in E \wedge eval(v) \succ eval(u) \wedge outdegree(u) < maxOut\}$, where $outdegree(u)$ returns the number of outgoing connections of vertex u. Each edge (u,v) is added to E in a random order.

Note that the above Connection Step applies a limitation on the *source* of connections, while the one presented in the previous section was applied to the *destination* nodes. Setting such limit restrict the number of sources of information that any node may use to compose a new solution. For example, consider a node u in a network where n other nodes are better evaluated than u; if $maxOut = 1$, then one single node among the n will be randomly selected to become the provider of information to u. Likewise, if $maxOut = N$ then connections are allowed without restrictions.

We must also factor $maxIn$ in since, as we showed in the previous section, this parameter defines two levels of performance - intermediate values of $maxIn$ leads the algorithm towards good solutions, while either very low or very high values generate poor solutions. Hence, we also allow the number of incoming connections to be restricted along with the number of outgoing connections. The following rule describes this more general case:

Definition *(Dynamic Connection Step (with maximum out-degree and in-degree))* Let E be the set of edges and V the set of vertices in the network. Then, $E = \{(u, v)|u, v \in E \wedge eval(v) \succ eval(u) \wedge outdegree(u) < maxOut \wedge indegree(v) < maxIn\}$. Each edge (u, v) is added to E in a random order.

We experimented varying $maxOut$ in order to observe how this parameter affects search performance. Figure 7.4 shows average best function values for different maximum out-degree and different maximum in-degree values, using the above described Connection Step. We can see that the behavior for the two "inefficient" states ($maxIn = 1$ and $maxIn = N$) are quite different. For $maxIn = 1$, $maxOut$ has a substantial influence in the algorithm's performance; however, such influence is only significant when severely restricting the number of outgoing connections - for $maxOut > 3$ no statistically significant differences could be measured. The same is true for other values of $maxIn$ that are well below the maximum, and the same behavior is observed for $maxIn = 10$ and $maxIn = 50$. For $maxIn = 100$, however, the system produces very poor solutions for all values of $maxOut$. In this case, there is a small benefit in increasing $maxOut$ from 1 to 2, but no significant improvements are observable by further relaxing the restriction on out-degrees.

In any case, the worst performance was obtained for $maxOut = 1$. For this value, vertices connect to only one single vertex and the shuffling mechanism is rendered useless, since this single vertex will provide all information to the connecting node. Hence, the mechanism is reduced to the copy mechanism, which at least partially explain the reduced performance (we have shown in Chapter 5 that copying can be considerably worse than shuffling). Unlike $maxIn$, we did not observe that allowing out-degrees to grow without limit is detrimental to performance.

7.3 Discussion

Unlike search in static networks, the dynamic MNA is allowed to reconfigure the network on the fly. As we will show in the next chapter, this allows for improved performance in several scenarios. In this chapter we showed that when the network is dynamic, it is possible to devise two distinct parameters, namely the maximum number of incoming connections vertices can receive (in-degree) and the maximum number of outgoing connections a vertex can create (out-degree).

The experiments shown here lead to the conclusion that there is a wide range of maximum in-degree values that yield approximately the same search performance. However, very extreme values of this parameter caused a huge decrease in performance. While for severely restricted maximum in-degree the algorithm was consistently unable to reach good solutions, very bad solutions were also somewhat rare. On the other

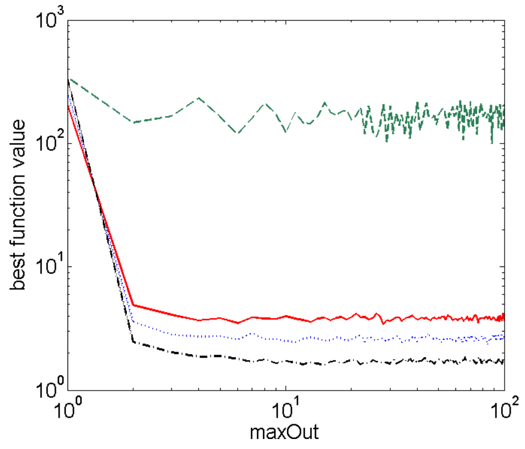


Figure 7.4: Average best function value when optimizing the 200-dimensional Griewank function with different values of $maxOut$. Solid line represents the case with $maxIn = 1$. Dotted line depicts the case with $maxIn = 10$. Dash-and-dotted line depicts the case with $maxIn = 50$. Dashed line is for $maxIn = N$.

side of the spectrum, when the vertices are allowed to capture an unlimited number of incoming connections, the quality of the solutions varies considerably - the algorithm's outcome can be anything between very good solutions and very bad solutions. Intermediate values of maximum in-degree allowed for better solutions to be reached consistently.

As for the maximum out-degree, its influence in the quality of the provided solutions was found to be very small. Only in the extreme case of severely limiting this parameter did the algorithm performed differently and provided very low-quality solutions. For all other values of maximum out-degree, no significant differences were found in the quality of the resulting solutions.

In our memetic framework, these results are evidence that it is in our best interest to allow for a vertex to receive information from as many sources as possible, but some care must be put into restricting the influence of individual sources in the network, by restricting the number of vertices any single vertex is allowed to send information to.

8 COMPARISONS AND APPLICATIONS OF MEMETIC NETWORKS

In this chapter, we compare the approaches proposed so far with three traditional local search algorithms. Moreover, we consider their application to two real-world scenarios that require extensive and efficient search algorithms to provide good solutions. The first one is the Traveling Salesman Problem (TSP), the problem of finding the shortest hamiltonian cycle in a graph. The second one is concept learning, where we are interested in finding a hypothesis that is able to represent a given domain by generalizing over known examples taken from the domain.

We are interested in verifying direct applicability of the proposed algorithms and compare them with each other and with traditional search algorithms. Three search algorithms were chosen to be used for comparison: Hill-Climbing, Local Beam Search and Genetic Algorithms. These algorithms were chosen due to their local search characteristics (i.e. they do not require gradients or information about the search space) and their wide acceptance in the academic community as general-purpose search algorithms.

8.1 The search algorithms

We consider three types of MNAs. The first two use static networks, with both small-world and scale-free networks. The third one uses a dynamic network. Each type is set to use the best observed parameters, as shown in Chapters 6 and 7 and made explicit in what follows. The Aggregation and Appropriation Steps are set according to each scenario. All networks are set with $N = 100$ vertices.

The MNA using the small-world generative model (SWN-MNA) is set with a neighborhood size of six ($K = 6$) and rewiring probability of zero ($\beta = 0.0$). The MNA using the scale-free generative model (SFN-MNA) is set with each node creating $H = 4$ connections an aristocratic network (i.e. hubs may grow without limits). The MNA using a dynamic network (DYN-MNA) is set to have maximum in-degree of ten ($maxIn = 8$) and unlimited maximum out-degree. It must be noted that while the SWN-MNA uses the small-world generative model, the network parameter that performed best in our experiments generated a network that is not small-world at all, since $\beta = 0.0$. Hence, while we do refer to this algorithm by the name of the generative process being used, the network being used here is actually a regular ring network.

The Hill-Climbing (HC) search algorithm used here is a first-choice stochastic hill-climbing as described in (RUSSELL; NORVIG, 2002): starting from a random state in the search space, a random neighbor state is generated and evaluated; if this state is better than the current state, then it becomes the current state, otherwise it is discarded and a new neighbor is considered. Since this method is not population-based, in order to provide a fair comparison in terms of the number of function evaluations being performed, when comparing to a MNA with a network containing N vertices, we execute N such searches in parallel, picking the best solution provided by any of these searches. The neighbors are generated by applying small random perturbations to the current state and we use the same basic algorithm used for MNAs in the Appropriation Step.

The Local Beam Search (LBS) method implemented is also the version described in (RUSSELL; NORVIG, 2002): it initializes N states at random and at each step of the algorithm, a neighbor of each state is generated and evaluated, totaling $2N$ states. Then, the best N states are kept, the remaining are discarded and the process is repeated. Unlike multiple independent Hill-Climbings, there is useful information being indirectly transferred to each search, since computational resources are directed towards promising states. In the former HC case, if a search is relatively unfruitful, it is allowed to continue anyway; in a LBS, these bad searches are discarded in favor of better ones. Once again, the neighbors are generated by applying small random perturbations to the states.

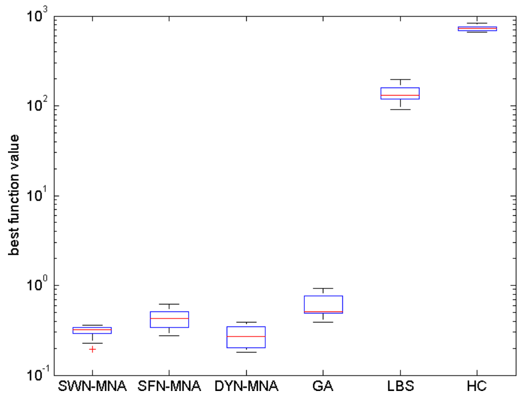
Finally, the Genetic Algorithm (GA) is the standard basic algorithm described in (MICHALEWICZ, 1996), using N individuals and roulette wheel selection. The algorithm for mutation is the same used in the Appropriation Step for MNAs. The crossover operation, however, is problem-dependent and specified in the next sections.

8.2 Function optimization

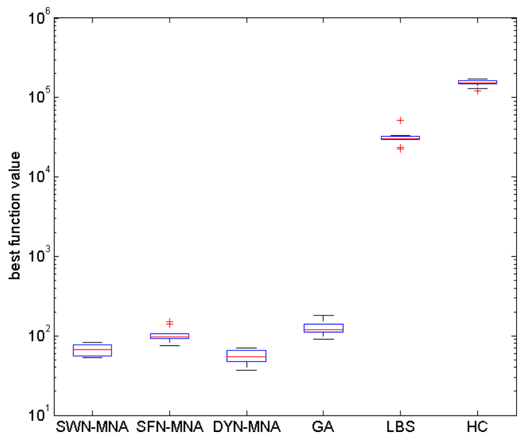
We begin by experimenting with the algorithms in function optimization tasks, using the functions proposed in Section 4.2. For all functions, 100 real-valued dimensions are used. Neighbor solutions, for all algorithms, are generated by applying a small random perturbation to the solution as follows: each variable, with probability $p_n = 0.01$ is replaced with a new value chosen uniformly at random from the function's domain. The MNAs all make use of the Aggregation by Shuffle.

Figure 8.1 shows the results after 5000 rounds of each algorithm. It is quite clear that HC perform very poorly in all instances, setting an upper-bound for the algorithms' performances. The LBS performs better than HC, but still worse than GA and the MNAs. Both HC and LBS perform at least one order of magnitude worse than the other algorithms, for all tested functions.

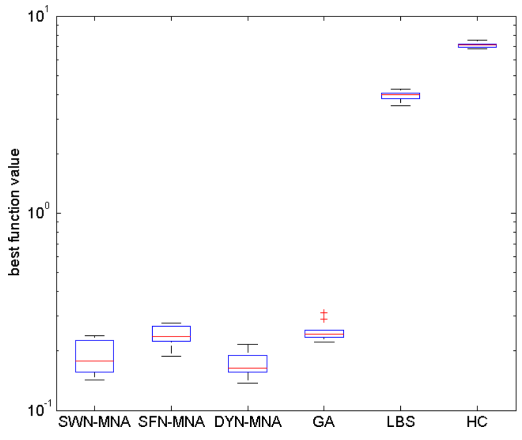
All MNAs perform better than the GA, in all tested functions. The difference is slightly more accentuated in simpler (unimodal) functions and, while small, it is sig-



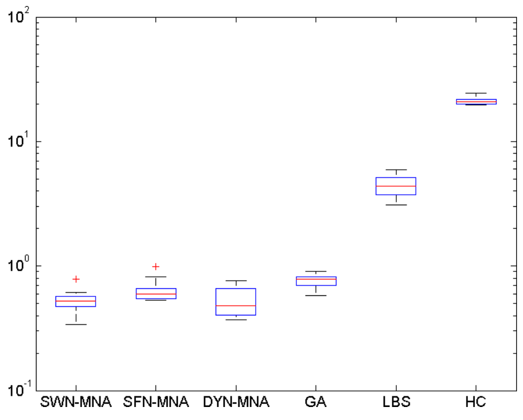
(a) Sphere function



(b) Hyper-Ellipsoid Function



(c) Ackley function



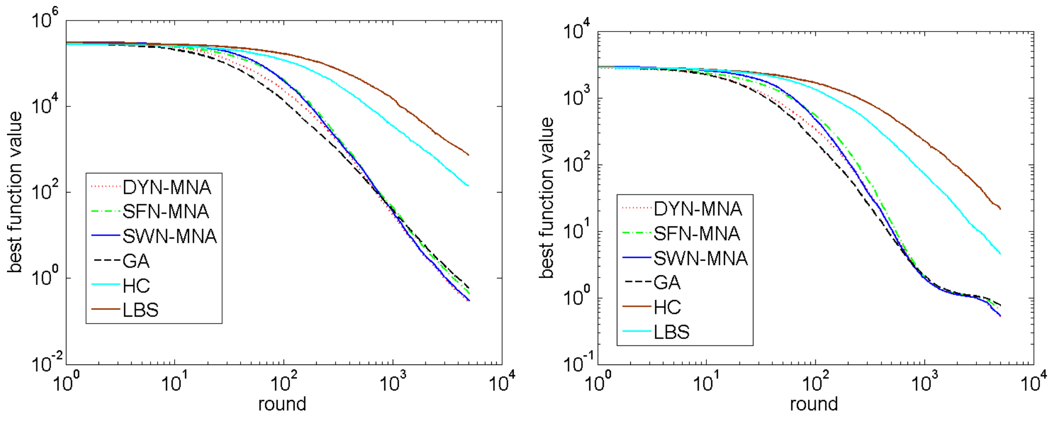
(d) Griewank Function

Figure 8.1: Best function values after 5000 rounds when optimizing different functions using different algorithms and 20 independent runs.

nificant. Of course, the GA used is somewhat conservative and many variations exist to improve on the traditional algorithm. Nonetheless, effort was put into tuning the algorithm's parameters so that it performed adequately on those functions. Hence, these results do not imply that MNAs are better than GAs, but rather that simple MNAs *can* be better than a equally simple GA. However, the huge differences when compared to LBS and HC are strong evidences that indeed MNAs are much better than these two methods. Among the MNAs, both SWN-MNA and DYN-MNA have comparable performance, but using a dynamic network yields a slightly better median, although this is not highly significant.

The SFN, however, performed 10%-20% worse than the other two networks in all cases. The performance using a SFN is actually closer to the performance obtained with the GA. Interestingly, the emergent mating network in GAs was actually shown to be scale-free (ONER; GARIBAY; WU, 2006). However, the connection between this two results is somewhat weak, as in a GA the mating network is not hard-wired and all individuals potentially have access to all other individuals. Rather, the DYN-MNA possess a closer relationship to the GA, in terms of the resulting interaction network, since the dynamic network also induce hubs and nodes have irrestricted access to all other nodes. Nonetheless, the DYN-MNA outperformed the GA in all test cases, an evidence that while the interaction network is adequate, the difference in performance comes from how information is used - in particular, due to the fact that MNAs are able to aggregate information from multiple sources.

Figure 8.2 shows convergence results for all tested algorithms. It is possible to observe that the GA is a quick starter, providing the best results in initial rounds, but failing to find good solutions in the long run. Even though DYN-MNA and SWN-MNA perform equally well in the long run, the DYN-MNA provides improved solutions earlier, approximating the GA in initial rounds (but lagging behind in intermediate rounds). Altogether, DYN-MNA provides a transition between the initial fast convergence of a GA and the long term performance of the SWN-MNA, performing at least as good as the latter in final rounds.



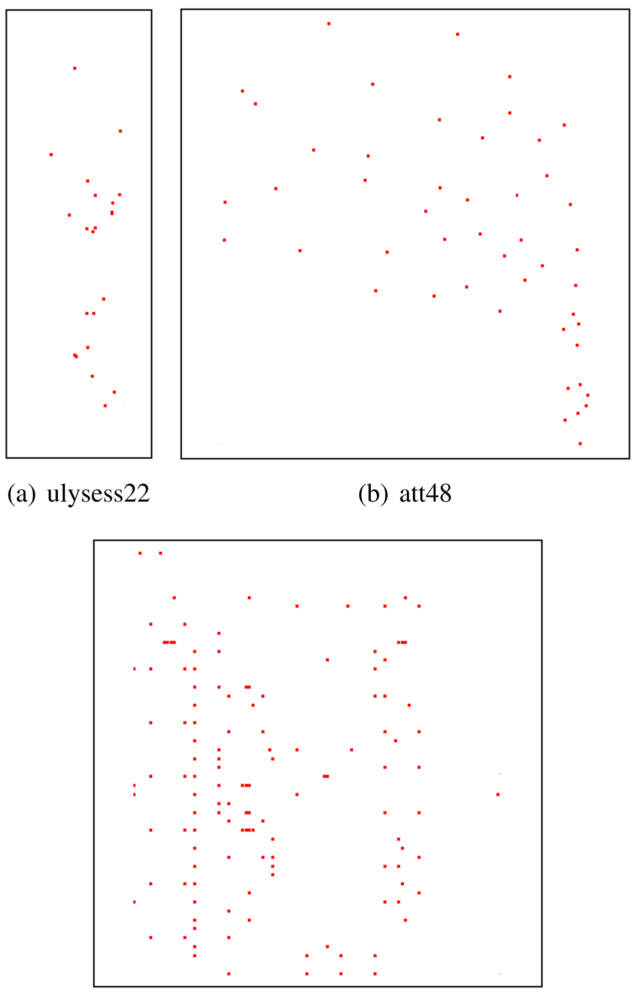
(a) Sphere function (b) Griewank Function

Figure 8.2: Average best function values when optimizing different functions using different algorithms. Averages are over 20 independent runs.

8.3 Traveling Salesman Problem

We apply the search algorithms to symmetrical TSP problems, taken from the TSPLIB [1]. We consider three datasets with increasing number of points. The smallest dataset, *ulysses22* is composed of 22 points representing locations mentioned in Homer's *Odyssey* poems. The second largest is the *att48* dataset, composed of 48 points representing the geographical location of 48 state capitals of the USA. Finally, the largest dataset considered is the *xqf131*, composed of 131 points representing a VLSI board. All three problems represent points in a euclidean space and are depicted in Figure 8.3.

[1]http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/


(a) ulysess22 (b) att48

(c) xqf131

Figure 8.3: Plots of the three TSP dataset used.

8.3.1 Settings

For all algorithms, a solution is a hamiltonian cycle and is represented by a vector of integers representing the order that points must be visited. Each point is represented by an integer from 1 to R, where R is the number of points in the problem. Hence, each integer must appear exactly once in the vector and the problem is a combinatorial one.

For the MNA, we use the MNA described in Section 5.2.3, which deals with combinatorial optimization. We also use the simpler Aggregation by Copy, in addition to the Aggregation by Common Transitions proposed in that section. The Appropriation Step proposed swaps each integer in the solution with a random position with probability p_n; following Section 5.2.4, we keep this parameter low and fix it at $p_n = 0.01$. The network is initialized with $N = 100$ vertices, containing random initial solutions. Therefore, N function evaluations are performed per algorithmic step.

The HC algorithm starts with a random solution and at each step, a neighboring solution is generated by using the same method presented above: each integer is randomly swapped with another position with probability $p_n = 0.01$. This new solution is evaluated and becomes the current solution if, and only if, it is better than the current solution. We allow $N = 100$ of these searches to be performed in parallel (but synchronously) and independently from one another, so that N functions evaluations are also performed and a direct step-by-step comparison to the MNA is possible. The best solution among these N is chosen to represent the best found solution in each round of the algorithm.

The LBS algorithm also start with N random solutions and at each step generates new neighbor solutions. These neighbors are generated by the same procedure of the two previous algorithms and each integer is randomly swapped with other position with probability $p_n = 0.01$. All (new and current) solutions are evaluated and the best N are kept to compose the new set for the next round of the algorithm.

Finally, the GA must use a specific-purpose crossover operator so that offsprings are generated without creating inviable solutions. Several crossover operators have been proposed in the literature. We implement the successful *order crossover* (OX) (DAVIS, 1985), which choose a subsequence of a path from one parent and tries to preserve the relative order of cities from the other parent to build a complete, valid, solution. The OX operator is a two-point crossover and as such selects randomly two points in the parents' solutions; to build an offspring, the part between these cut points from parent A is copied to the offspring and the remaining cities are copied from parent B in the same order that they appear, omitting the existing cities. The second offspring is generated identically, but inverting parents A and B. Mutation is also performed by the random-swap method of the previously presented algorithms, keeping $p_n = 0.01$.

8.3.2 Results

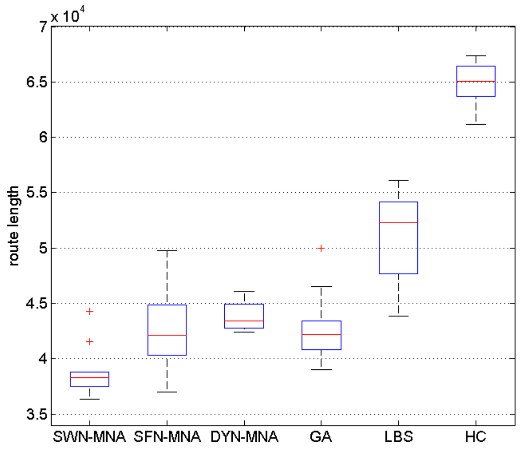
Figure 8.4 shows the results for each dataset and all search algorithms. In all datasets the LBS and HC performed much worse than the other four algorithms. We therefore focus our analysis on these four. For the smallest dataset, *ulysses22*, it is possible to observe that using a Small-World Network provided the best results - the median route length was considerably shorter and the standard deviation was also smaller than in other algorithms. The GA performed better than both SFN-MNA and DYN-MNA, while the SFN-MNA while providing a larger median, also provided a few overall better solutions than GA and SWN-MNA, at the cost of a wide variance.

For the intermediate case, *att48*, the gap between different MNAs and the GA is much smaller and not significant. All four algorithms provide comparable medians. The SFN-MNA, however, presents a larger variance among these four.

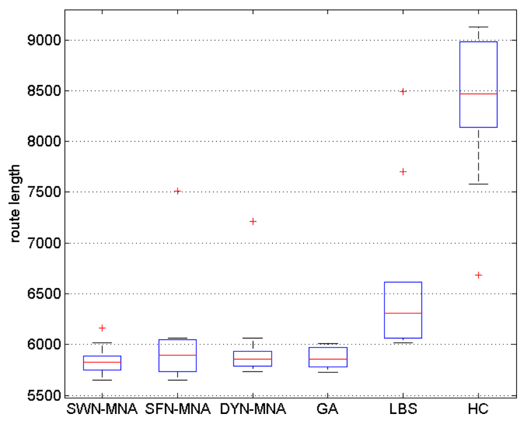
For the largest case, *xqf131*, the SFN-MNA provides the best results, with shorter median route lengths and smaller variance. Both SWN-MNA and DYN-MNA provide comparable results, with the DYN-MNA providing a better median. The GA now performs much worse, approaching the performance of the LBS.

Two conclusions can be possibly inferred from these results. First, they show that the aggregation method used, which aggregates common transitions, is quite effective to compose new routes for the TSP, an evidence that exploiting statistical features from multiple sources can be useful for this scenario. Moreover, the precise topology of the network to be used seem to be dependent on the problem at hand. The second conclusion is that it seems that Scale-Free Networks become more suitable as the TSP instance grows, while Small-World Networks are better for smaller instances.

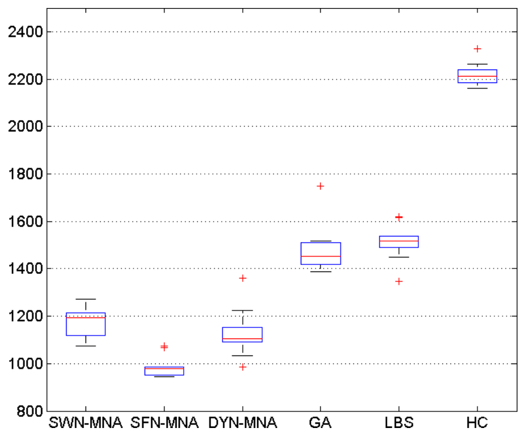
Of course, more data is still required to generalize such conclusion to other instances. Nonetheless, the aggregation method used compose new solutions by aggregating common transitions; therefore, it performs better if there is a large number of sources to extract such regularities. The more vertices using a transition, the more likely it is that that transition is useful. Since in a SFN there are hubs that concentrate edges, these hubs do have access to many sources. A better performance in SFNs is then expected.



(a) ulysses22



(b) att48



(c) xqf131

Figure 8.4: Boxplots showing the best routes found for different TSP instances after 1000 rounds of the MNAs using Small-World Networks (SWN-MNA), Scale-Free Networks, (SFN-MNA), Dynamic Networks (DYN-MNA), as well as results for a Genetic Algorithm (GA), Local Beam Search (LBS) and Hill-Climbing (HC).

8.4 Concept Learning

In this second application, we consider a Machine Learning task. Given a set of labeled examples taken from a domain, the task asks to learn the underlying concept from these examples and classify unseen ones. An example is a pair <input,class> that is drawn from a domain with a fixed distribution. The *input* is a set of attributes used for classification and the *class* is the class to which the object with the specified set of attributes belongs to.

We consider the popular Iris dataset (FISHER, 1936), where the goal is to distinguish between three types of flowers: *Iris Setosa*, *Iris Versicolour* and *Iris Virginica*. Four attributes are available, all in centimeters: sepal length, sepal width, petal length and petal width. The specific dataset used provides 150 labeled examples.

We divide the example set in two sets. The *training set* is a subset from the available examples that is used for training; training is the process of acquiring a concept from examples. The *validation set* is a subset from the available examples, disjoint from the training set, which is used to validate the learned concept, so that the task is not reduced to memorizing the training set. We randomly divided the Iris dataset in two disjoint sets, in order to compose a training set and a validation set.

The problem is then to create a mapping from an input to a class, so that the largest number of examples in the validation set is correctly classified. There are several choices for a structure to create such a mapping (e.g. production rules, decision trees, bayesian networks) and we use an Artificial Neural Network, more specifically a Feed-forward Multi-layered Perceptron (FF-MLP) (HAYKIN, 1998).

A FF-MLP is composed of interconnected artificial neurons (perceptrons). A typical perceptron is composed of a numerical input vector $\vec{x}$ and one output y. The inputs are coupled to a *weight vector*, $\vec{w}$, one weight per input. At each time step, the perceptron calculates output such as

$$y(\vec{x}) = f(\vec{x} \cdot \vec{w})$$

where $f()$ is some predefined function that limits the output to a range of values, such as a step function or a hyperbolic tangent. A FF-MLP is composed of layers of perceptrons, so that the output of a layer is the input of the next one. A typical FF-MLP has an *input layer*, which simply receives the inputs $\vec{x}$ from the environment and propagates them to one or more *hidden layers*. Each hidden layer is composed of predefined number of perceptrons and, usually, each perceptron receives as inputs the outputs of every unit in the previous layer (and only from this layer). Then, all hidden units compute an output and send them to the next layer and so on, until the *output layer* is reached, whereas the output of this last layer is the “answer” of the network to the input vector. A much more detailed description of FF-MLPs and its many variants can be found in (HAYKIN, 2001).

Provided the FF-MLP possess at least one hidden layer, by setting the weights appropriately it is possible to create arbitrary complex mappings. In fact, it has been shown that a FF-MLP can be an universal function approximator. Given a network structure, the problem of learning using a neural network is that of finding the appropriate weights that produce the desired mapping. The most popular algorithm to perform learning in FF-MLPs is the *Backpropagation* algorithm. This algorithm performs a gradient search in the weight space that minimizes the error of the network. There are many ways to define such error and the most common one, used here and called mean squared error (MSE), is as follows:

$$E(\vec{w}) = \frac{1}{N}\sum_{d \in D}(t_d - y_d)^2$$

where D is the training set, t_d is the desired output for the training example d and y_d is the output generated by the network given the weights $\vec{w}$ and training example d. The Backpropagation algorithm performs an iterative search by adjusting the weights so that at each step $E(\vec{w})$ is reduced:

$$w_i = w_i - \eta\frac{\delta E}{\delta w_i}$$

where η is a positive constante called the *learning rate*, and determines the step size of the gradient search. Setting η too low slows convergence down, but setting η too high may cause the algorithm to overshoot the optimum, making convergence impossible. Also, since it is a gradient search, it may converge to local optima which are not the global optimum. Nonetheless, this algorithm is widely used and the central reason for the success of neural networks in general.

We use a FF-MLP to act as our learning structure and apply it to learning the widely used Iris dataset. The neural network used is shown in Fig. 8.5; Four input units, one for each attribute available, one hidden layer with five units and one single output unit are used. A hyperbolic tangent function is used to compute the output of all units - hence, the output is limited between -1.0 and $+1.0$. Given the Iris dataset is composed of three distinct classes, we set the following rule for classifying the output generated by the neural network: -1 represents the *Iris Setosa*; 0 represents the *Iris Versicolour* and +1 represents the *Iris Virginica*.

8.4.1 Experiments

In order to learn the weights for the proposed FF-MLP, we used two approaches. The first one used the Backpropagation algorithm, as described in (HAYKIN, 2001). The second one uses a MNA to search for the best combination of weights. Both approaches initializes all weights with values picked uniformly at random from the range $-1, +1$. We tested all MNA variants described so far, namely SWN-MNA, SFN-MNA and DYN-MNA, and compared the results with the Backpropagation.

For the MNAs, we again set $N = 100$. Hence, effectively there are 100 neural networks being evaluated in parallel, one in each node of the memetic network. In order to perform a fair comparison in terms of number of evaluations, we performed 100 runs of the Backpropagation algorithm, picking the best performing run to represent

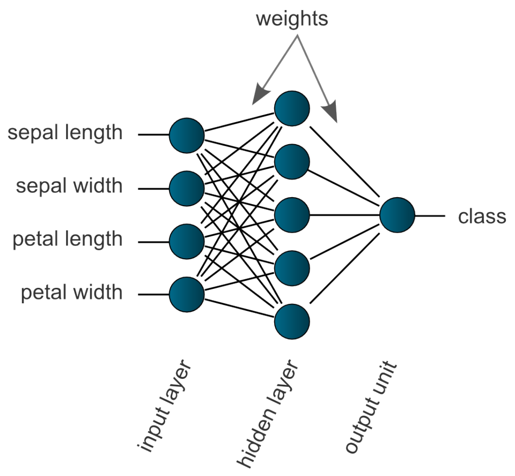


Figure 8.5: Feedforward Multi-layer Perceptron used to learn the Iris dataset.

the result. We then repeated each batch of 100 runs 20 times. The results presented here are the results of these 20 trials.

The training examples are presented to the neural networks in batch mode - i.e. all examples are presented, the error is calculated and only then weights are updated[2]. The presentation of the entire training set to the neural network is called an *epoch*. We conducted a maximum of 500 epochs for each neural network, in each run and each trial. Therefore, each algorithm performed in total one million epochs (20 trials $\times$ 100 runs/nodes $\times$ 500 epochs).

For the MNAs, the same parameters used in the previous sections were applied. The learning rate for the Backpropagation algorithm was set *ad hoc* at $\eta = 0.1$, which was the best performing value for tested values between 0.05 and 1.0 (in 0.05 increments).

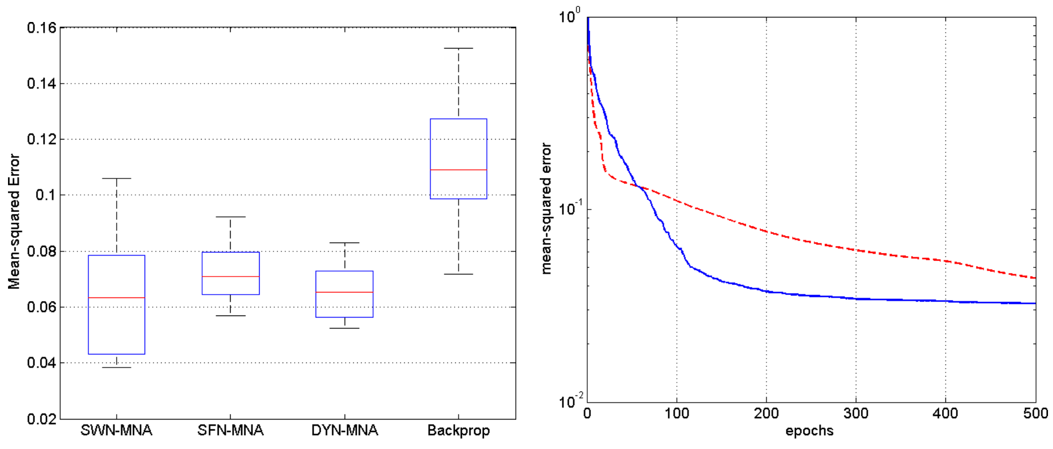
Figure 8.6 shows the results. From the boxplots, we can see that all three MNAs reach lower values of MSE when compared to the Backpropagation. This shows that MNAs are interesting alternatives to perform learning in neural networks. At least in part, the advantage comes from the fact that MNAs are less likely to get stuck in local optima, since the shuffling mechanism allows large changes to occur. Therefore, the algorithms are able to find weight configurations that are more suitable for the problem in hand.

Among the MNAs, the SWN-MNA provides the best overall results, but at the cost of a wider variability in performance: this variant provided both the best and the worse runs, but the median was better than the other two. The Backpropagation also displayed a large variance, an evidence of its susceptibility to the initial state.

Figure 8.6(b) brings a epoch-by-epoch comparison between the SWN-MNA and the Backpropagation algorithms. As expected, the Backpropagation performs much better in initial rounds, since it is able to exploit the gradient very effectively. However, in later rounds it is surpassed by the SWN-MNA, which is then able to reach better

[2]This is in constrast with on-line learning, where the weights are updated after each presented example.

solutions.



(a) (b)

Figure 8.6: Boxplots (a) and averages (b) for different search techniques applied to the concept learning task. In (a) mean-squared errors are for the 500th epochs. In (b), the dashed line represent the Backpropagation algorithm and the solid line represent the SWN-MNA.

8.5 Discussion

This chapter compared the proposed algorithms with other traditional algorithms in a number of search scenarios. In every scenario considered, at least one of the proposed algorithms fared better than the traditional ones. The specific MNA that provided the best result varied with the problem, which is to be expected given the No Free Lunch Theorem (WOLPERT; MACREADY, 1997). This shows that network properties must be matched to the problem at hand. As we showed in previous chapters, this can be accomplished by setting how much the algorithm exploits known solutions or explores unknown areas of the search space, which can be translated to specific network properties.

Nonetheless, the MNA did much better than the HC and LBS algorithms in all problems. The HC algorithm is a local-only search algorithm and thus it is expected that it would perform poorly in the multimodal problems proposed. The LBS, while being more able to further explore the search space, provides a very strong exploitation mechanism, which quickly limits the space being searched. The MNAs, on the other hand, can be tuned so that there is a better balance between exploitationd and exploration, hence providing better average solutions on these problems. The GA is also able to provide such balance, and did provide results comparable to those provided by the MNAs, but showed to be less scalable for the TSP problems (i.e. for larger instances, the solutions were worse).

9 CONCLUSIONS AND FUTURE WORK

This thesis presented a model of social search applied to problem-solving tasks. The proposed model is based on the concept of *memes* and the exchange of information through a network as the main search mechanism, hence the name Memetic Network. In this approach, candidate solutions to a problem (states) are encoded into the vertices of a network and new solutions are generated by the aggregation of elements of existing solutions in the network; such interaction between vertices is constrained by the network's topology.

The main purpose of the Memetic Network model is to provide a framework over which we explored one of the major goals of this thesis, which is the hypothesis that network properties can affect the problem-solving capabilities of systems composed of multiple interacting actors. This goal was motivated by the increasing number of systems that aim at solving problems and that require social interactions in some way, either by artificial actors, as in Multi-agent Systems, or by human beings, as in crowdsourcing. Setting the right social network to improve the solution quality or convergence rate of such systems is then of a major concern.

By building several instances of Memetic Networks, named Memetic Network Algorithms (MNAs), we conducted experiments aiming at verifying how network properties can affect the algorithm's performance. We showed that the presence of a network is often beneficial and a system that does not allow communication (i.e. the system perform several parallel independent searches) converges much slower than one that allows such interactions, often resulting in poor solutions. Hence, regarding our Hypothesis 1, we conclude that allowing communication *can* be beneficial, and that this is true for a wide range of network configurations and induced dynamics; nonetheless, for networks with some properties, the system performance can be considerably poor.

It also became clear that several network properties play a substantial role in how well a system can perform., thus confirming our Hypothesis 2. In most cases, we found that allowing excessive communication is harmful to the algorithm's performance, hence modifying the network so as to restrict interactions can improve the outcome's quality. For example, in the popular Small-World network model, we found that the best performance was often obtained when neighborhood sizes were restricted. When analyzing dynamic networks, we found that there in asymmetry in how information is spread: while solutions should be built by using information from as many sources as possible, there are benefits in restricting the reach of good information in the network. We have argued that most of these results can be mapped to the duality of exploitation of good known solutions and exploration of unseen areas of the search space.

While the Memetic Network model is most useful to provide insights on how complex interacting social actors are influenced by the underlying social network, we also provided evidences that instances of the model applied to optimization problems are actually useful as general-purpose optimization algorithms. We compared MNAs with other traditional search algorithms, namely Hill-Climbing, Local Beam Search and Genetic Algorithms, in benchmarks functions and the real-world instances of the Traveling Salesman Problem. The MNAs performed better than the Genetic Algorithm in most functions; there was no specific MNA (Small-World, Scale-Free or Dynamic) that fare well in all scenarios, an evidence that the network must fit the problem at hand.

We also applied the MNAs in a Machine Learning scenario, where the task was to learn the weights of a neural network so that it was able to correctly classify as many examples as possible. Again, the MNA did better than the Backpropagation algorithm in the long run, finding weights that allowed the neural network to classify the examples with smaller error; in the short run, however, the Backpropagation algorithm did better, providing a faster convergence.

In summary, the main contributions of this thesis are:

- A new model of social search based on meme exchange over a social network was proposed. To the best of our knowledge, no previous attempts were made to capture into a model social characteristics aiming at problem-solving in such general way. This model is useful as a starting point towards understanding the role of social relations in social systems target at problem-solving;

- Instantiations of the proposed model were provided. Algorithms implementing several variations of each step of the Memetic Network model were proposed, along with results of experiments validating the implementations and exploring their different parameters. Such instantiations can be directly applied to a number of problem-solving scenarios, such as numerical and combinatorial optimization;

- Results on the role of network properties in search performance were obtained from extensive experiments. Both Small-World and Scale-free networks have been considered and their main properties were related to the algorithm's performance in optimization tasks. Given the ubiquity of these network models in social systems, these results are stepping stones towards a more complete theory of problem-solving in social systems;

- Results on the effectiveness of using the model as a general-purpose search algorithm were provided. Instances of the model performed better than traditional general-purpose algorithms in several scenarios. These results, along with

> Dawkins's hypothesis that memetic evolution proceeds at a faster pace than genetic evolution, are evidence that further studies on social search are worthy pursuing as a way to build automatic problem-solvers or social information processing systems.

Several paths for further research can be devised. While we considered some of the most popular and widespread network models (e.g. random, structured, small-world, scale-free, aristocratic, egalitarian), still there are several network types and properties that may play a role in Memetic Networks' performance. For example, geographically-constrained networks (HOFFMANN, 2009) can present interesting properties and may be necessary to model systems where such constraints are a natural part of the scenario (such as in foraging with robots).

Our study also only focused on computational simulations of the model. Another interesting line of research is to apply the concepts and methodologies used here in systems composed of real people, in a similar fashion to the experiments described in (LAZER; FRIEDMAN, 2005). In contrast to this latter study, it is interesting to ask how humans fare against traditional algorithms and the simulated Memetic Networks in solving difficult problems; if it is the case that humans can perform better under controlled environments, then one can try and understand why they perform better and improve the existing algorithms. If, however, the algorithms perform better than humans, then the reverse is true and one can apply policies to improve social problem-solving, which could have an impact in the intellectual productivity of companies and organizations. We hypothesize that, in any case, the difference should be in how humans aggregate information from several sources and how they dynamically adapt the network to improve the filtering of useful information.

Finally, the results presented here are only part of an initial movement towards a better understanding of social systems and how they can be used to solve real-world problems, either by using artificial or human actors. An interdisciplinary effort must be pursued in order to provide such better understanding. The field of Artificial Intelligence is a natural candidate to lead such endeavour, as it already aggregates diverse fields such as psychology, sociology and computer science, which, we believe, are the central disciplines needed in order to meet this challenge.

A REDES MEMÉTICAS

Sistemas sociais, compostos de muitos atores interagentes envolvidos em alguma forma de relação social, têm sido alvo de interesse nas áreas de Ciência da Computação e Inteligência Artificial. Um dos motivos para o crescente interesse é a necessidade de algoritmos especializados para gerenciar, analisar e extrair conhecimento de conjuntos de dados massivos gerados por interações sociais (e.g. rede de citações de artigos científicos, troca de e-mails, hyperlinks em websites). Neste contexto, da mesm forma que a necessidade de analisar cadeias de DNA e outras estruturas biológicas deram origem a área de Bioinformática, a necessidade de analisar estrutras sociais complexas está dando origem a área de Ciência Social Computacional. Esta área utiliza computadores e algoritmos para analisar e entender sistemas sociais, incluindo simulação de sistemas sociais reais, algoritmos para extrair padrões de interações sociais, bancos de dados especializados e assim por diante.

Por outro lado, há cada vez mais interesse em fazer exatamente o oposto de utilizar computação para analisar sistemas sociais: utilizar tais sistemas para produzir computação. Esta abordagem pode ser dividida em dois tipos básicos. No primeiro tipo, estruturas sociais são utilizadas para compor sistemas artificiais que almejam a solução de problemas. Este é o caso de Sistemas Multi-Agentes (SMA), compostos de diversos agentes independentes e semi-autônomos que colaboram (ou competem) para solucionar uma dada tarefa; resultados de como estruturas sociais emergem no mundo real podem então ser aplicados em tais SMA de forma a melhorar seu desempenho ou reduzir seu custo de operação.

O segundo tipo de abordagem almejando a geração de computação a partir de sistemas sociais é a utilização de sistemas reais, compostos de pessoas, e fazer uso destes sistemas de tal forma que pessoas, e não máquinas, seja responsáveis por solucionar tarefas e produzir computação. Este é o caso de sistemas de crowdsource, Wisdom of Crowds, Computação Social e Processamento Social de Informações; todos estes utilizam a interação entre pessoas para solucionar problemas de maneira quase-algorítmica (i.e. com comportamento e resultados bem definidos). A principal idéia por trás desta abordagem é que há um grande potencial em solucionar problemas complexos exigindo pouco de um grande número de pessoas. Esta última abordagem, foco desta tese, pode ainda ser dividida em diferentes perspectivas em relação a como um problema é distribuído para o sistema social e como os resultados são extraídos. Em um extremo do espectro, crowdsourcing preocupa-se com criar uma chamada aberta para um grupo por contribuidores interessados em solucionar um problema e, uma vez que um voluntário toma posse da tarefa, este a solucionará sem ajuda dos demais. No outro lado do espectro, Processamento Social de Informações (PSI) utiliza a interação de muitos atores para colaborativamente e iterativamente construir uma solução.

Todos estes sistemas tem em comum a necessidade de algum tipo de interação entre atores. Ao combinar o conjunto de todo atores e suas interações sociais, podemos compor uma rede social. É claro que tal rede tem um papel central em qualquer sistema social, uma vez que ela representa e define como atores interagem uns com os outros, através da especificação de propriedades de rede específicas (como a sua topologia).

Uma pergunta extremamente relevante é, então, como propriedades de redes se relacionam com a capacidade de um sistema social em solucionar problemas. Nesta tese, relatamos um conjunto de modelos, algoritmos, experimentos e seus resultados direcionados a responder precisamente esta questão. Fazemos isso através de uma metodologia que encaixa-se entre os dois tipos de abordagens apresentados acima: propomos um modelo que imita diversos aspectos de interações sociais e a troca de informações em redes sociais que não se restringe simular tais sistemas, mas sim produzir computação na forma de algoritmos capazes de solucionar problemas, de forma a garantir um meio de definir e medir desempenho.

Assim, como argumentaremos, este modelo, denominado Redes Meméticas, é próximo em sua natureza de Algoritmos Genéticos (AG), que imitam processos naturais (seleção natural e evolução), mas que tem como objetivo a solução de problemas em geral através de meios automáticos (sem utilizar genes ou DNA reais). Da mesma forma, fazemos uso de conceitos de redes sociais para gerar algoritmos que são capazes de lidar com tarefas específicas de forma automática, sem que atores sociais reais estejam necessariamente envolvidos. Ainda assim, defendemos que o modelo e seus algoritmos associados, são também úteis para prover uma maior compreensão de sistemas sociais reais e que seu maior valor está exatamente nesta utilidade.

De forma a alcançar nossos objetivos, fazemos uso do conceito de memes - informação que se propaga por cópia em redes sociais - proposto por Dawkins (DAWKINS, 1976). Também fazemos uso de diversos conceitos e ferramentas de Teoria das Redes (ou “Ciência das Redes”), tanto na forma de topologias e propriedades específicas de rede para serem aplicadas aos nossos algoritmos, como também no uso de ferramentas de análise para extrair resultados. O ato de solucionar problemas nesta tese é definido como um problema de busca, onde a tarefa é reduzida a encontrar um estado desejável em um espaço de estados potencialmente muito grande. Definir o problema dessa forma permite que se meça de forma objetiva o desempenho de sistemas. Os algoritmos propostos são, portanto, algoritmos de busca que, como mostraremos podem ser bastante competitivos com técnicas de busca tradicionais (tal como Subida de Encosta e Algoritmos Genéticos).

A.1 Objetivos e Metodologia

De forma geral, estamos interessados em compreender como interações sociais em redes sociais podem ser utilizadas para solucionar problemas. Nosso objetivo principal é estudar como sistemas sociais podem ser organizados ou construídos de forma a serem capazes de solucionar problemas de forma eficiente. Isto é, estamos interessados em extrair computação de redes sociais. Duas hipóteses guiam este trabalho:

Hipótese 1 : Dado um conjunto de agentes independentes agindo para solucionar um problema comum, a capacidade de solucionar o problema de forma coletiva pode ser melhorada ao permitir que estes agentes troquem informações sobre o problema.

Hipótese 2 : Dado um conjunto de agentes que interagem entre si para solucionar um problema, compondo um sistema, a rede formada pelos padrões de interações pode afetar a qualidade da solução encontrada.

Os objetivos específicos desta tese são como segue:

1. Propor um modelo de solução social de problemas onde a rede social tenha uma parte central no processo de encontrar soluções;

2. Identificar propriedades de rede relevantes e seus efeitos na capacidade de solução de problemas de sistemas sociais;

3. Comparar a eficácia de topologias de rede comuns em cenários de solução de problemas;

4. Verificar a utilidade de algoritmos de busca inspirados em redes sociais como ferramentas genéricas para solucionar problemas.

Para atingir nossos objetivos, propomos um modelo, com algoritmos associados, capazes de solucionar problemas utilizando múltiplos agentes agindo em paralelo, capazes de trocar informações sobre soluções individuais através de uma rede configurável subjacente. Assim, é possível modificar tal rede para verificar sua influência no desempenho do sistema como um todo.

Instanciamos tal modelo para que possa ser aplicado em problemas diversos, realizando simulações numéricas sobre problemas de benchmark. Configuramos tais problemas como tarefas de busca, onde desejamos encontrar um estado desejado entre múltiplos estados possíveis, representando soluções. Mais ainda, restringimos nosso interesse em problemas de otimização. Assim, nosso modelo deve ser capaz de construir soluções e avaliá-las. Os resultados de tais simulações são analisados estatisticamente, de forma a permitir o relacionamento entre propriedades de redes e desempenho do sistema. A Tabela A.1 mostra as funções utilizadas como benchmark - nelas, desejamos encontrar os valores de x que minimizam as funções.

Sphere	$f1(\vec{x}) = \sum_{i=1}^{\lvert\vec{x}\rvert} x_i^2$
Hyper-Ellipsoid	$f2(\vec{x}) = \sum_{i=1}^{\lvert\vec{x}\rvert} 5ix_i^2$
Ackley	$f3(\vec{x}) = -20e^{0.2\sqrt{\frac{1}{\lvert\vec{x}\rvert}\sum_{i=1}^{\lvert\vec{x}\rvert} x_i^2}} - e^{\frac{1}{\lvert\vec{x}\rvert}\sum_{i=1}^{\lvert\vec{x}\rvert} cos(2x_i)} + 20 + e$
Schwefel	$f4(\vec{x}) = 418.9829\lvert\vec{x}\rvert - \sum_{i=1}^{\lvert\vec{x}\rvert} -x_i sin(\sqrt{abs(x_i)})$
Griewank	$f5(\vec{x}) = \sum_{i=1}^{\lvert\vec{x}\rvert} \frac{x_i}{4000} - \prod_{i=1}^{\lvert\vec{x}\rvert} cos(\frac{cos(x_i)}{\sqrt{i}}) + 1$

Table A.1: Funções de *benchmark* utilizadas nesta tese.

Em seguida, comparamos os algoritmos gerados com outros algoritmos tradicionais - Algoritmos Genéticos, Backpropagation, Busca em Feixe Local e Subida de Encosta - e com diferentes versões do algoritmo utilizando redes distintas, em problemas mais complexos. Aplicamos tais algoritmos em problemas de TSP e de Aprendizado de Conceitos.

A.2 Redes Meméticas: um modelo de busca em rede

Propomos um modelo de busca que utiliza uma rede subjacente para explicitamente estruturar a troca de informações entre múltiplas buscas paralelas. Este modelo é inspirado na troca de informações em redes sociais e tem as seguintes características principais:

1. É baseado em população, executando múltiplas buscas em paralelo;
2. Comunicação entre as buscas é estruturada por uma rede explícita e configurável;
3. Novos estados são gerados agregando informações de vértices adjacentes na rede;

A inspiração deste modelo vem do conceito de memes, proposto por Richard Dawkins. Dawkins argumenta que evolução cultural ocorre utilizando mecanismos similares a evolução genética. O termo meme foi criado como o equivalente cultural do gene. Um meme é qualquer coisa que pode ser copiado de uma mente para outra, através de qualquer meio - por exemplo, idéias, bordões, músicas, conceitos. Um meme “salta” de cérebro para cérebro na sociedade, transformando-se, unindo-se com outros memes, dando origem a novos memes.

A evolução cultural, Dawkins argumenta, procede por meio de uma seleção natural e memes. Indivíduos expõem seus memes a audiências, que copiam (lembram) aqueles que consideram úteis ou interessantes. Estas cópias tem a chance de se propagar mais, para outras mentes. No processo, alguns memes são modificados, seja por ruído, seja por inclusão de novas informações locais (que são outros memes) a seu “hospedeiro”, e estes passarão também pelo processo de seleção. Após algumas iterações, apenas aqueles memes mais bem “adaptados” permanecem no conjunto de memes.

Apesar das semelhanças, memes possuem diferenças cruciais em relação a genes. Dawkins observou que memes são mais suscetíveis a mudanças e uniões - memes são raramente passados na sua forma exata original, sofrendo modificações sutis no processo de comunicação, através da agregação de outros memes. Por exemplo, quando lemos uma matéria no jornal, podemos comentar o conteúdo com um colega, mas nunca reproduziremos palavra por palavra a matéria. Pelo contrário, omitiremos partes, adicionaremos nossa própria opinião ou idéias lidas em outras fontes.

Argumentamos que há outra diferença central entre memes e genes, que pode ser um dos fatores capazes de explicar a aparente maior velocidade de evolução de memes, e que tentamos capturar em nosso modelo. A diferença é em como um meme se propaga. Enquanto genes são transmitidos com fortes restrições geográficas, memes não tem tal limitação. Com a invenção da imprensa e com as tecnologias atuais de comunicação, memes sofrem muito menos restrições geográficas - i.e. memes podem se propagar para qualquer lugar do mundo de forma rápida. Mas não é o caso que memes são transmitidos para todos indivíduos do mundo, nem para um subconjunto aleatório de indivíduos. Memes propagam-se através de uma rede social, que impõem restrições em como o meme pode se difundir. Tais redes sociais podem ser muito mais dinâmicas que qualquer sistema de difusão genético.

Adicionalmente, em um arranjo biológico, há duas maneiras de criar um novo conjunto de genes. A primeira é através da mutação, único motor de mudanças em organismos assexuados. A segunda maneira é através do sexo, que combina genes de exatamente dois indivíduos. Em um ambiente cultural, por outro lado, novos memes podem ser criados a partir de mudanças em um único meme, ou através da agregação de múltiplos memes. Não há restrições no número máximo de "pais" envolvidos na criação de um meme. Mais ainda, o número de "pais" não é fixo. Cada meme pode ser afetado por um número diferente de outros memes.

A.2.1 O modelo

Denominamos nosso modelo de Redes Meméticas. Este nome reflete a característica principal do modelo: a inspiração na difusão de memes em redes sociais. A idéia central é ter múltiplas buscas paralelas trocando informações sobre soluções através de uma rede. Cada busca pode ser compreendida como um container de memes, enquanto memes são propagados pela rede para outros containers. Um meme nesse modelo é qualquer informação sobre o estado mantido por uma busca, que inclui o estado em si (e.g. uma solução candidata para um problema) e meta-informação sobre o estado (e.g. a avaliação da solução candidata). Cada busca agrega memes recebidos de alguma maneira, possivelmente adicionando informação local, e torna o resultado disponível para a rede novamente.

Para obter as propriedades desejadas, o modelo deve ser capaz de agregar qualquer número de memes em uma única solução candidata. O número de memes sendo recebidos por qualquer vértice na rede, e o número de vértices que recebem um meme, não são especificados e dependem apenas da topologia da rede. Adicionalmente, exigimos algum nível de autonomia para cada nó, de forma que uma Rede Memética composta de um único nó seja ainda capaz de realizar buscas e melhorar soluções. Isto é importante, pois entendemos a rede como um facilitador que permite um melhor uso para múltiplas buscas paralelas, e não como uma condição necessária para realizar busca.

Nosso modelo especifica como a rede é formada e como ela usada para realizar buscas. O modelo utiliza um grafo $G = \langle V, E \rangle$, onde V é um conjunto de vértices representando possíveis soluções para o problema sendo tratado e E é um conjunto de arestas $(u, v)|u, v \in V$ representando adjacências entre os vértices. Estes vértices não são valorados, mas podem ser orientados. Adicionalmente, uma função de avaliação $eval$ mapeia um valor real para cada solução, de acordo com sua adequação ao problema.

O conjunto E define a topologia da rede e suas propriedades, i.e. a dinâmica *da* rede. Este pode ser construído de duas formas: estaticamente ou dinâmicamente. No caso estático, E é pré-definido e permanece fixo durante a execução do algoritmo. No segundo caso, E pode ser alterado em tempo de execução, seguindo critérios de conexão. Consideramos ambos os casos nesta tese.

O conjunto V é composto de *containers* para memes, que codificam soluções para o problema. Cada vértice representa exatamente uma solução completa. Uma vez que E esteja definido, as soluções em V são atualizados utilizando informações que são acessíveis através de E. Para melhor especificar esses procedimentos, dividimos o modelo geral em três passos, descritos a seguir.

Passo de Conexão

O passo de conexão é responsável por definir a estrutura da rede. Pode ser executado apenas uma vez, no início do algoritmo, definindo uma rede estática, ou a cada iteração, tornando possível redes dinâmicas. De forma geral, a inicialização e quaisquer modificações em E são parte do Passo de Conexão.

Passo de Agregação

Neste passo, os vértices tornam disponíveis suas soluções para a rede através de suas conexões (definidas no Passo de Conexão), e agregam as soluções que lhe são disponibilizadas, construindo uma nova solução. Este passo é a parte "social" da busca, já que vértices podem influenciar a busca de outros vértices, trocando informações sobre os estados sendo visitados.

Este passo define precisamente como múltiplas informações (memes) vindos de diferentes fontes devem ser agregados em uma única solução. A maneira de realizar tal agregação é, como veremos, dependente do problema.

Passo de Apropriação

Este passo é responsável por *apropriar* uma solução agregada, introduzindo nela informações locais ao vértice. É neste passo que informações externar (em relação ao passo de Agregação) podem influenciar a construção da solução. Tal informação local pode ser uma heurística ou informação disponível apenas para um único vértice. Por exemplo, este passo pode incluir a aplicação de alguma heurística de busca local, como subida de encosta, a solução antes de disponibilizá-la para a rede.

Este passo representa uma busca independente - i.e. a busca que seria realizada por cada vértice se não houvesse a possibilidade de comunicação pela rede. Se este passo for vazio, então o Passo de Agregação torna-se o único responsável pela busca. No outro extremo, o Passo de Agregação pode ser vazio, e o sistema comporta-se como $|V|$ buscas independentes.

A.2.2 Instanciações do modelo

O modelo descrito até então apenas fornece um *framework* para a construção de algoritmos de busca em rede. Uma implementação do modelo exige a especificação de cada passo descrito. Chamamos uma instanciação de Algoritmo de Rede Memética (MNA, na sigla em inglês).

Adiamos a discussão do Passo de Conexão para adiante, já que um dos focos desta tese é o estudo da influência deste passo no desempenho de um MNA. O Passo de Agregação é central para o modelo e discutimos algumas opções para sua implementação.

O Passo de Agregação é responsável por usar informação de nós vizinhos para construir uma nova solução. Assim, a implementação depende da codificação utilizada no problema. Como argumentamos, o conceito de memes is baseado na idéia de imitação e cópia. Este conceito leva a um método de agregação simples que se baseia em copiar boas soluções dos vizinhos:

Definição *Agregação por Cópia do Melhor Vizinho* Seja A o conjunto de vértices adjacentes a qualquer vértice v; seja $u = argmax_{x \in A} eval(x)$. Se mais de um vértice satisfizer esta conição, u é escolhido aleatoriamente entre estes. Então, faça $v \leftarrow u$, caso contrário v é inalterado.

Este método, para cada vértice, copia a solução completa do melhor vértice adjacente. Naturalmente, a maneira que uma solução é copiada depende de detalhes de implementação, mas também é claro que esta idéia funcionna para pelo menos uma grande quantidade de problemas diferentes. Enquanto a *vizinhança física*, composta pelos potenciais parceiros na agregação, é definida pela topologia da rede, a *vizinhança de fato* é restrita a um único nó - o melhor nó da vizinhança física.

Esta *Agregação por Cópia* é simples e de fato usa informação de todos vizinhos. No entanto, exceto para o melhor vizinho, apenas a avaliação dos demais é utilizada, não os memes que compõem as soluções individuais. É parte central da idéia de memes a possibilidade de recombinação e união de memes e exploramos este conceito em seguida.

A.2.2.1 Otimização de Parâmetros e Valores Reais

Neste cenário, uma função com múltiplos parâmetros arbitrária $f(\vec{x})$ é dada para a qual queremos encontrar $\vec{x}*$ de forma que $f(\vec{x}*)$ é o menor valor possível para a função (i.e. um problema de minimização). Uma solução candidata para o problema é representada por um vetor $\vec{x}$ de valores reais dentro de valores determinados.

Um método de agregação possível para este tipo de codificação faz uso de informações nos vizinhos e explora a representação: uma nova solução é composta tomando a média sobre todos elementos das soluções.

Definição *Agregação por Média sobre Vizinhos* Para cada vértice v, faça $A(v) = u|(v,u) \in E \wedge eval(u) \succ eval(v)$. Seja $\vec{x_i}$ a solução candidata contida no vértice i e $x_{i,j}$ o j-ésimo componente desta solução. Crie uma nova solução agregada $\vec{x_0}$ tal que $\vec{x_{0,j}} = \frac{1}{A}\sum_{i \in A} x_{i,j}$.

Um método mais geral, mas que potencialmente utiliza informações de todos vizinhos, é uma variação da Agregação por Cópia, que faz a cópia de partes específicas de uma solução.

Definição *Agregação por Combinação sobre Vizinhos* Para cada vértice v, faça $A(v) = u|(v,u) \in E \wedge eval(u) \succ eval(v)$. Seja $\vec{x_i}$ a solução candidata contida no vértice i e $x_{i,j}$ o j-ésimo componente desta solução. Crie uma nova solução agregada $\vec{x_0}$ tal que $\vec{x_{0,j}} = x_{k,j}$, onde para cada valor de j, $k \in A$ é aleatoriamente selecionado.

Neste método, uma nova solução é criada combinando partes de soluções de vértices vizihos melhores avaliados. É mais geral pois não exige que $\vec{x}$ seja composto de valores numéricos, ao contrário da Agregação por Média.

Há também diversas opções de implementação para o Passo de Apropriação. Neste, podemos inserir técnicas de busca local, tais como subida de encosta ou busca por gradiente. No entanto, a forma mais simples de modificar localmente uma solução é realizar uma simples busca aleatória.

Definição *Apropriação por Aleatorização* Para cada parâmetro, com probabilidade p_n, substitua o parâmetro com um valor aleatório dentro da faixa de valores possíveis do domínio.

Esta especificação implementa uma caminhada aleatória para valores baixos de p_n. Naturalmente, este não é um método que, sozinho, é eficiente para realizar buscas. Ainda assim, utilizamos tal apropriação na maior parte dos nossos experimentos.

Os métodos descritos podem ser modificados para outras representações. Por exemplo, se representarmos uma solução por uma cadeia de bits, devemos modificar os passos de Agregação e Apropriação para refletir tal mudança. A Agregação por Combinação é trivialmente modificada para lidar com bits, simplesmente tomando como parâmetro cada bit individual. Nesse caso, a granularidade do meme torna-se menor - ao invés de um meme representar um parâmetro completo, este passa a representar apenas parte do parâmetro (um bit). A Apropriação por Aleatorização pode ser modificada de forma idêntica.

Outras representações podem exigir implementações bastante diferentes. Se estivermos interessados em otimização combinatório, devemos levar em conta que certos elementos não podem se repetir e, portanto, a simples cópia de elementos de outras soluções torna-se inviável.

Tomando como exemplo o Problema do Caixeiro Viajante (TSP, na sigla em inglês), uma solução pode ser representada por um vetor de inteiros indicando a ordem que cidades devem ser visitadas. É claro que o mesmo valor não pode aparecer duas vezes na mesma solução, levando ao problema de como extrair informações úteis dos vizinhos para compor uma nova solução. Um método viável é como segue.

Definição *Agregação por Transições Frequentes* Para cada vértice v, faça $A(v) = u|(v,u) \in E \wedge eval(u) \succ eval(v)$. Inicie um novo caminho com uma cidade arbitrária c_0. A próxima cidade c_1 será aquela que segue c_0 com maior frequência na soluções em $A(v)$. O processo é repetido até que um caminho completo esteja formado. Empates são resolvidos escolhendo aleatoriamente entre as opções.

O Passo de Apropriação também deve ser modificado, o que pode ser feito de forma trivial:

Definição *Apropriação por Troca* Para cada cidade na solução, troque-a de posição com outra cidade em uma posição aleatória com probabilidade p_n.

A.2.2.2 Experimentos

Para compreender melhor o papel de cada componente de um MNA, conduzimos experimentos comparando a eficácia do algoritmo quando desligamos certos passos. Apresentamos aqui os principais resultados obtidos.

A Figura A.1 mostra o resultado de uma instância de MNA que utiliza Agregação por Combinação e Apropriação por Aleatorização, executada sobre uma rede em forma de anel quando tentando minimizar a função Griewank.

Observa-se que o algoritmo completo encontra soluções muito melhores do que com qualquer passo desativado. Quando o Passo de Agregação está desativado, o algoritmo é reduzido a uma busca aleatória e nenhuma comunicação se faz presente; cada nó realiza uma busca independente. Sem o Passo de Apropriação o algoritmo tem sua habilidade de explorar o espaço de estados limitado pelo conjunto de soluções geradas no início do algoritmo, resultando em uma busca muito ineficaz.

A.3 Busca em Redes Estáticas

Iniciamos nossos estudos analisando como diferentes tipos de redes estáticas afetam o desempenho de Redes Meméticas. Experimentar com redes estáticas permite um maior controle de propriedades das redes estudadas. Para obtê-las, o Passo de Conexão é executado uma única vez, no início do algoritmo, definindo uma rede que se mantém inalterada no decorrer de uma busca.

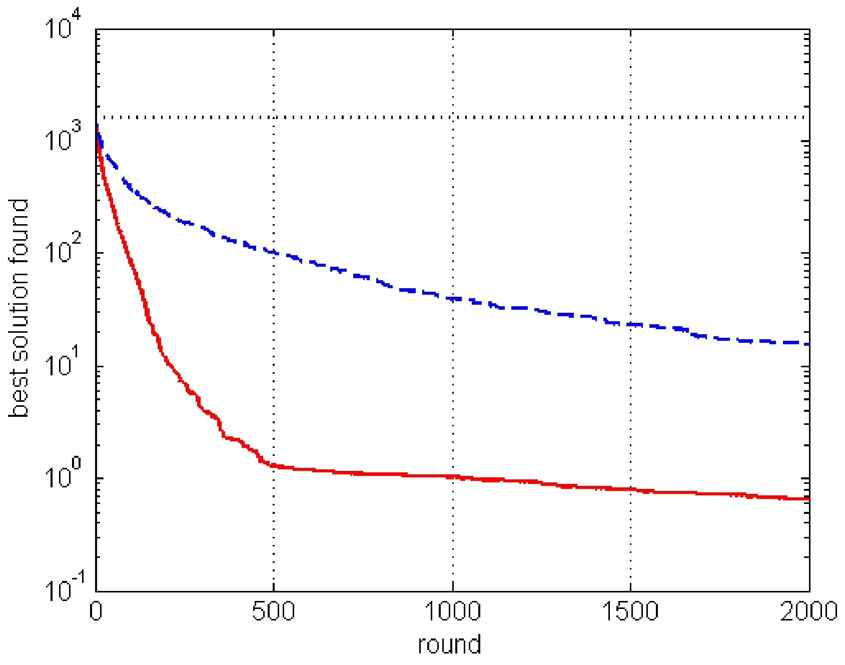


Figure A.1: Avaliação da melhor solução encontrada para cada rodada do algoritmo completo (linha sólida), do algoritmo sem o passo de conexão (linha tracejada) e sem o passo de apropriação (linha pontilhada).

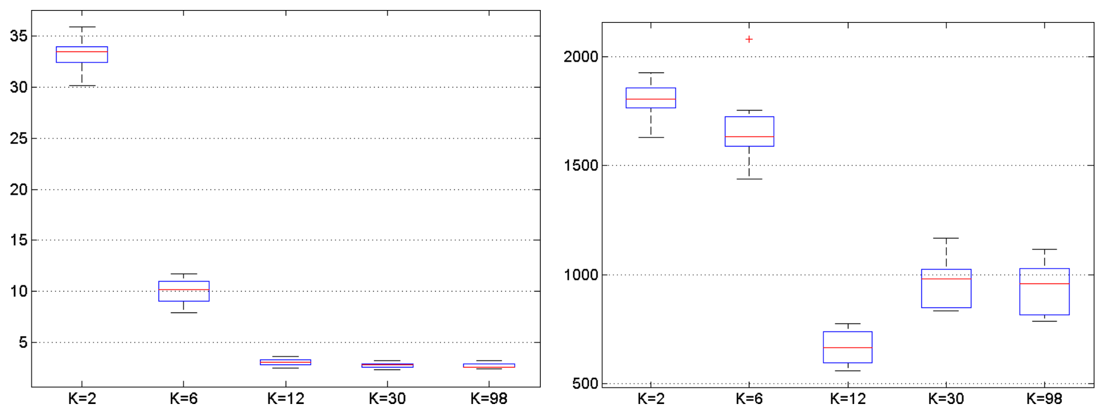
A.3.1 Busca em Redes de Mundo Pequeno

Uma Rede de Mundo Pequeno tem como principais características um alto grau de *clusterização* e um baixo comprimento médio de caminhos. Para estudar este tipo de rede, utilizamos o modelo proposto por Watts. Iniciamos com uma rede em forma de anel, onde cada vértice é vizinho de K vizinhos mais próximos. Então, com probabilidade β, cada aresta (u, v) é redirecionada para conectar o vértice u a um novo vértice w escolhido aleatoriamente.

Assim, dois parâmetros podem ser analisados: K e β. K define o tamanho da vizinhança na rede e especifica o número de vértices que podem compor novas soluções. A Figura A.2 mostra resultados usando Agregação por Combinação em dois momentos do algoritmo aplicado a função Griewank. Observamos que K tem grande influência no algoritmo. O pior caso ($K = 2$) gera soluções que, em média, são mais de uma ordem de magnitude pior que o melhor caso ($K = 98$).

Observa-se também dois comportamentos diferentes quando analisamos diferentes momentos do algoritmo. Para um grande número de rodadas, valores maiores de K sempre levam a soluções que são melhores ou tão boas quanto as encontradas para valores menores de K. Porém, no curto prazo, para um pequeno número de rodadas, observa-se que o melhor desempenho é obtido para um valor intermediário de K. Valores maiores ou menores que este valor ótimo levam a uma degradação no desempenho do algoritmo.

Testamos também os efeitos de variar β, a probabilidade de redirecionar arestas.



(a) T=2000 (b) T=200

Figure A.2: Melhores valores encontrados quando otimizando a função Griewank com 200 dimensões, para diferentes valores de K e em diferentes momentos do algoritmo.

Quanto maior o valor de β, mais a rede se aproxima de uma rede aleatória. Baixos valores, por outro lado, levam a uma rede altamente estruturada.

Observamos que o comportamento quando variamos β depende da densidade da rede, isto é, do valor de K. Para valores baixos de K (e.g. $K = 2$), um aumento de β sempre leva a um melhor desempenho do algoritmo. Porém, para valores maiores de K (e.g. $K = 6$), observamos que, para um pequeno número de rodadas do algoritmo, o comportamento inverso é obtido: um aumento de β leva a um pior desempenho.

A.4 Busca em Redes Sem Escala

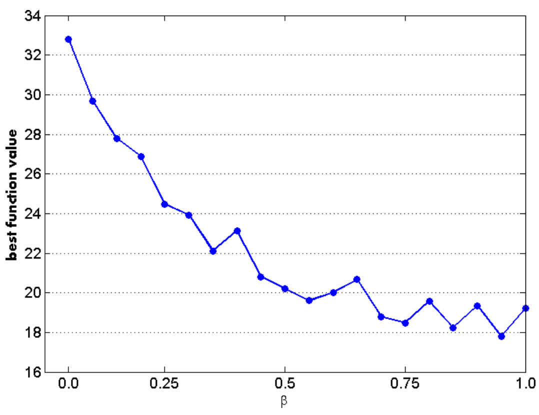
Redes Sem Escala são redes cuja distribuição de graus dos vértices não segue uma distribuição normal, mas sim uma curva de lei de potência. Isso leva a existência de *hubs*, nós altamente conectados.

Utilizamos o modelo de construção proposto em (BARABÁSI; ALBERT, 1999). Introduzimos os vértices um a um. Cada vértice conecta-se a M vértices já existentes na rede. A probabilidade de um novo vértice u conectar-se a um vértice v é proporcional ao grau do vértice v.

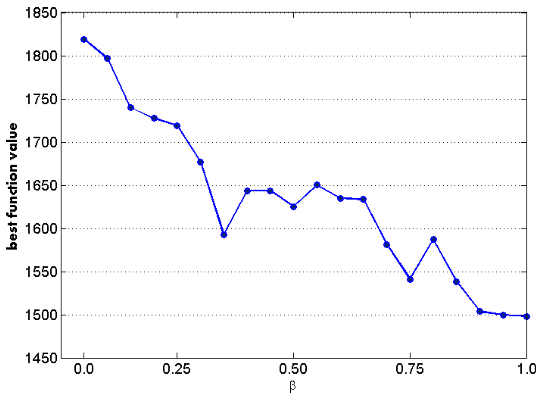
Além de M, definimos um segundo parâmetro, H, que define o número máximo de arestas que qualquer vértice pode fazer parte - isto é, o número máximo de conexões possíveis. O parâmetro H controla o tamanho máximo dos *hubs*.

As Figuras A.5 e A.6 mostram os resultados da variação destes parâmetros no desempenho do algoritmo. É possível observar que maiores valores de M sempre levam a um melhor desempenho do algoritmo. Isto é, redes sem escala densas se saem melhor do que redes esparsas. Já M tem pouca influência sobre o algoritmo. Apenas para redes muito esparsas ($M < 4$) o tamanho máximo do hub tem alguma influência.

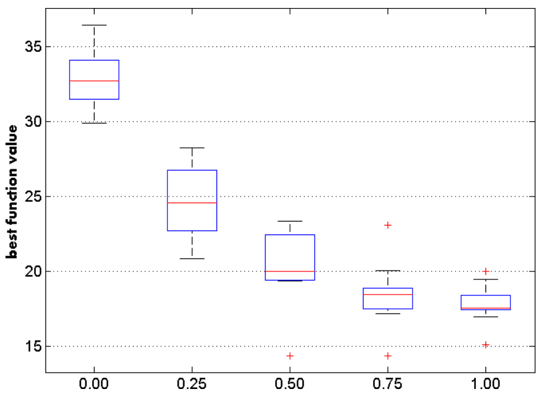
Interessantemente, o grau individual dos diferentes vértices tem influência no desempenho individual destes vértices. Em geral, quanto maior o grau de um vértice, melhor o seu desempenho médio. A Figura A.7 mostra tal relação.



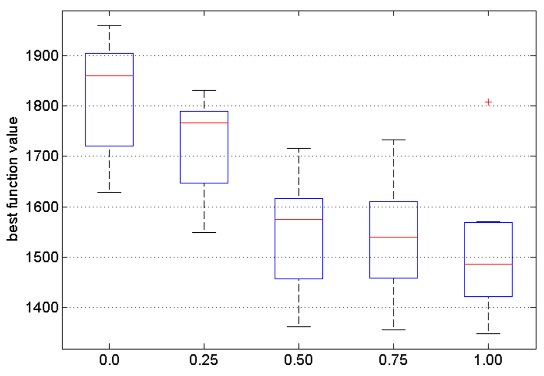
(a) T=2000



(b) T=200



(c) Boxplot for T=2000



(d) Boxplot for T=200

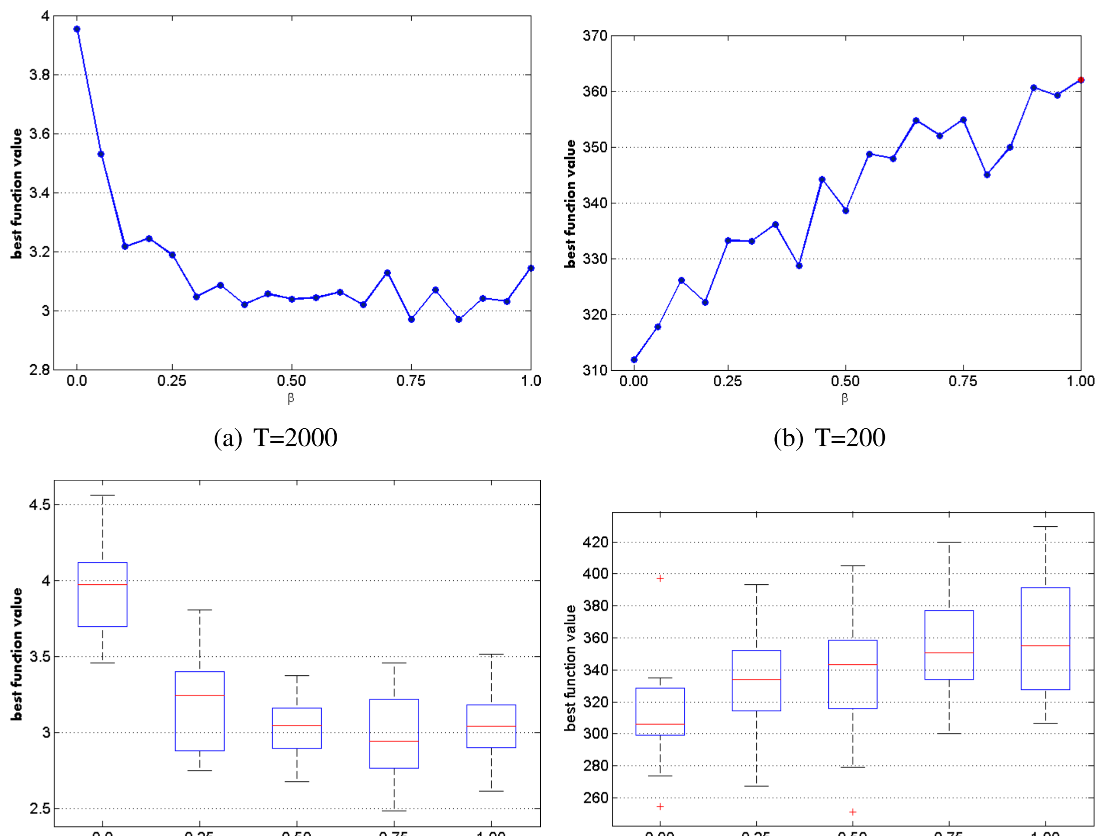
Figure A.3: Avaliação da melhor solução para a função Griewank, com 200 dimensões, e diferentes valores de β e diferentes momentos do algoritmo. $K = 2$.

A.5 Busca em Redes Dinâmicas

Ao permitirmos que a rede subjacente se altere durante a execução de um MNA, estamos permitindo que os diversos vértices reconectem-se de alguma forma. A introdução de tais reconexões também introduzem outros parâmetros que potencialmente influenciam o desempenho do algoritmo. Neste capítulo, exploramos as implicações da introdução de redes dinâmicas.

O Passo de Conexão agora é executado a cada rodada do algoritmo e novos vértices podem ser introduzidos e destruídos. Não há mais uma preocupação em induzir uma topologia específica e cada nó é mais egocêntrico, decidindo criar ou eliminar conexões conforme lhe convém. Ainda assim, cada aresta deve refletir de alguma forma o benefício de criar tal conexão. Isto é, uma aresta deve apenas ser criada se isso for benéfico para o nó que a está criando. Assim, definimos um Passo de Conexão que introduz tal raciocínio sem se preocupar com a topologia que poderá gerar.

Definição *Passo de Conexão Dinâmico* Seja o grafo $G =< V, E >$, onde V é o conjunto de vértices e E o conjunto de arestas direcionadas. Este último é definido como: $E = (u, v)|u, v \in E \wedge eval(v) \succ eval(u) \wedge indegree(v) < maxIn \wedge outdegree(u) < maxOut$.



(a) T=2000 (b) T=200

(c) Boxplot for T=2000 (d) Boxplot for T=200

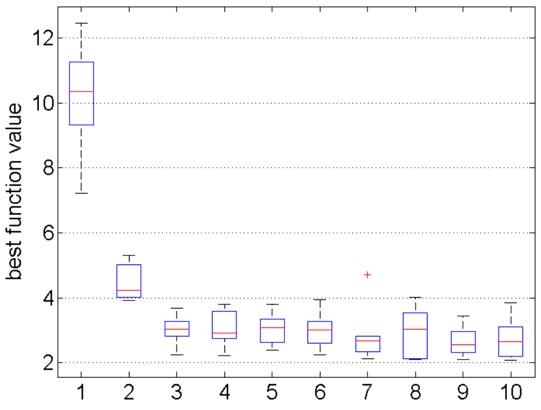
Figure A.4: Avaliação da melhor solução para a função Griewank, com 200 dimensões, e diferentes valores de β e diferentes momentos do algoritmo. $K = 6$.

Nessa definição, $outdegree(u)$ representa o grau de saída do vértice u; $indegree(v)$ representa o grau de entrada do vértice v; $maxIn$ e $maxOut$ são parâmetros que determinam, respectivamente, o máximo grau de entrada e saída para qualquer vértice na rede.

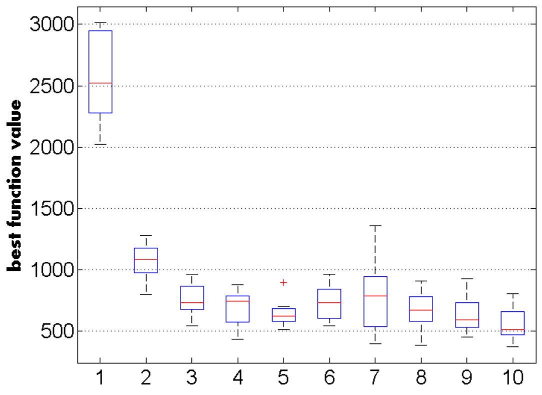
Cada vértice na rede, portanto, conecta-se a outros vértices que são, segundo a função de avaliação $eval$, melhores o próprio, respeitadas as limitações de grau. A idéia por trás é que há um interesse dos vértices em conectarem-se a outros vértices melhores avaliados de forma a extrair informações úteis destes.

Mantemos inicialmente $maxOut = N$ e variamos $maxIn$. A Figura 7.1 mostra que valores intermediários deste parâmetro levam aos melhores resultados, mas há pouca ou nenhuma diferença entre os diversos valores intermediários. Porém, valores extremos levam a resultados bastante ruins. Para rodadas iniciais, vemos que um valor baixo (mas não limítrofe) para $maxIn$ leva aos melhores resultados. Assim, para os valores testados, vemos que $maxIn = 10$ garante um bom desempenho tanto no curto prazo como no longo prazo.

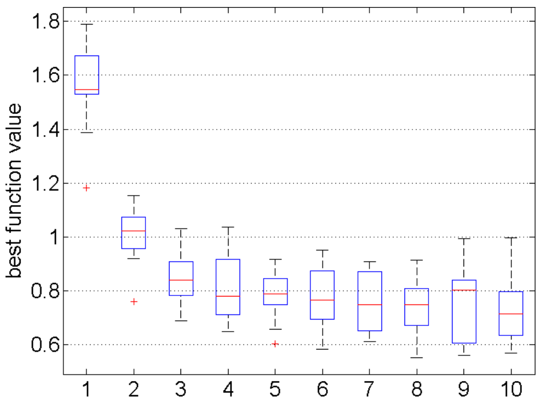
Já a Figura A.9 mostra resultados para diversos valores de $maxOut$ e $maxIn$. Vemos que o comportamento para os dois estados mais ineficientes ($maxIn = 1$ e



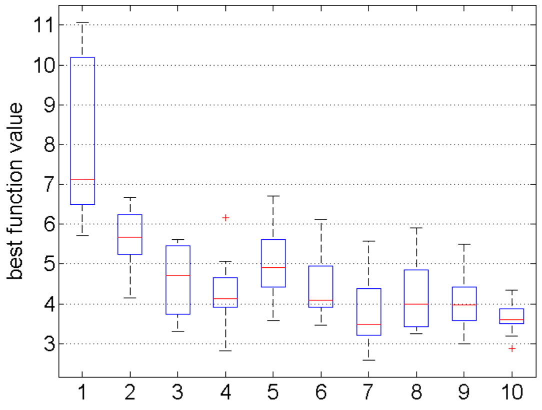
(a) Sphere function



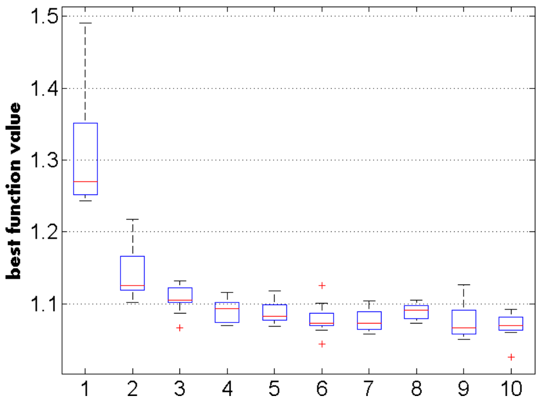
(b) Hyper-Ellipsoid function



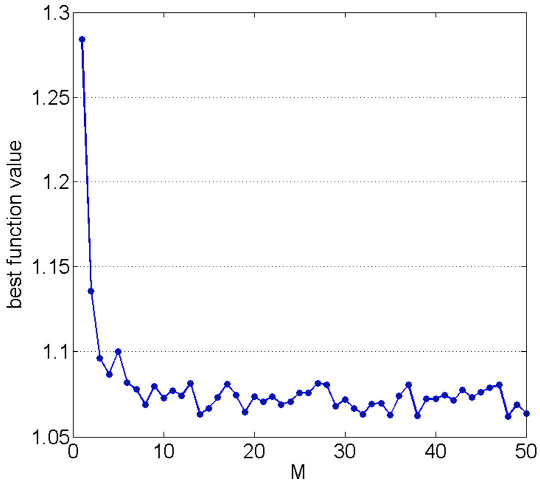
(c) Ackley function



(d) Schwefel function



(e) Griewank function



(f) Average for Griewank function

Figure A.5: Melhores avaliações das funções após 2000 rodadas usando diferentes funções e diferentes valores de M.

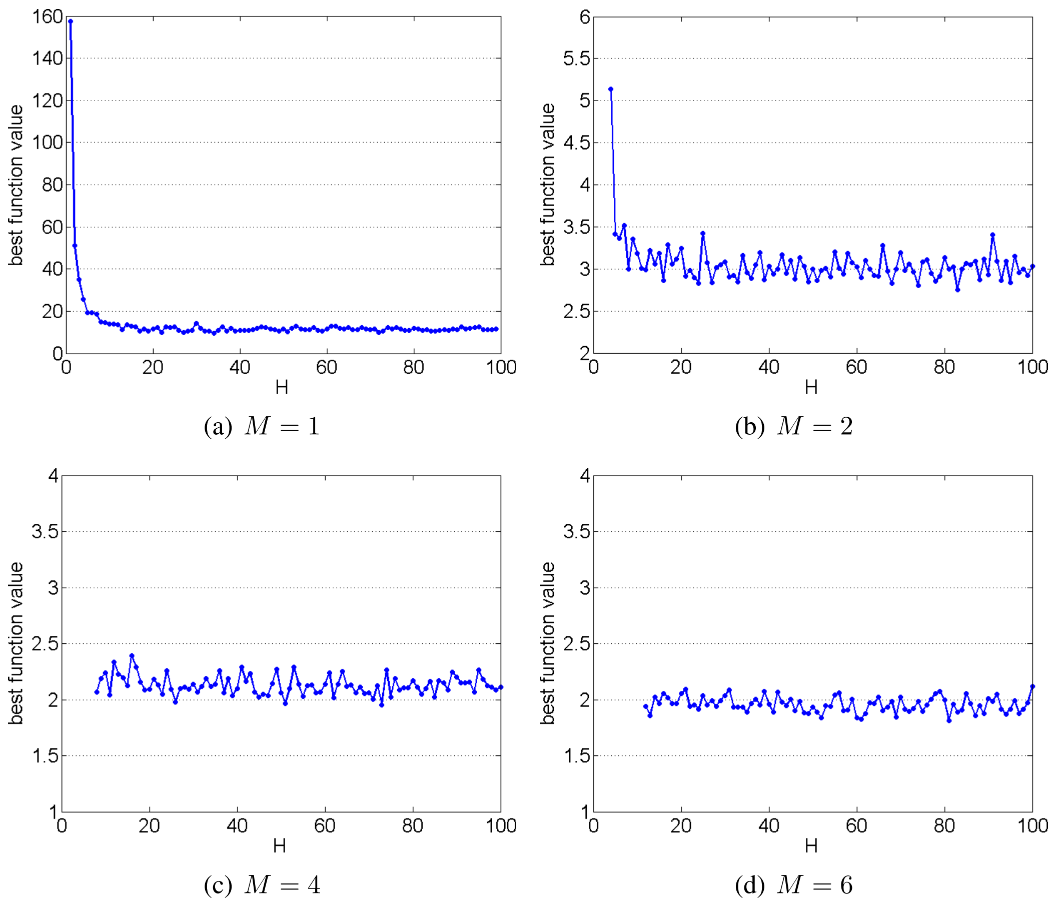


Figure A.6: Melhor avaliação da função Griewank com 100 dimensções e diferentes valores de H.

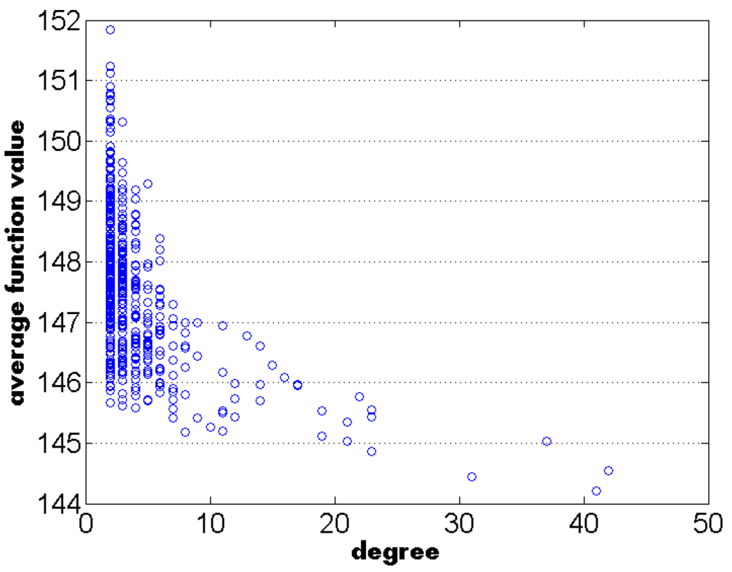
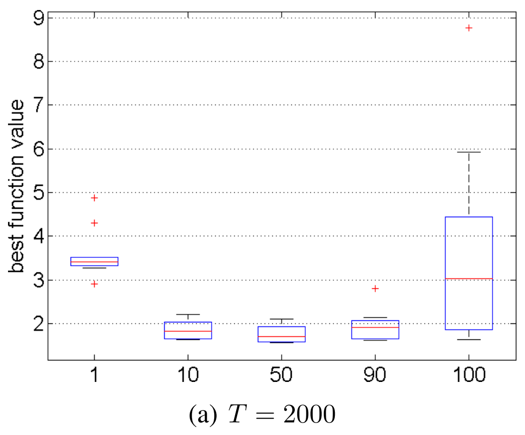
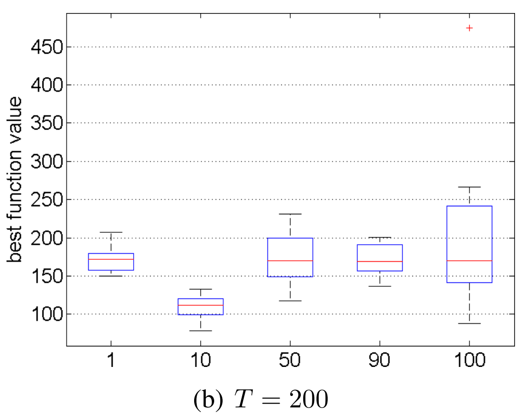


Figure A.7: Relação entre o desempenho média de cada nó em uma SFN com 500 vértices e o grau correspondente. O coeficiente de correlação é de $-0,497$



(a) $T = 2000$ (b) $T = 200$

Figure A.8: Melhores avaliações para a função Griewank para diferentes valores máximos de grau de entrada.



$maxIn = N$) é bastante diferente. Para $maxIn = 1$, $maxOut$ influencia de forma significativa o desempenho do algoritmo, mas tal influencia se limita a valores baixos de $maxOut$. Para $maxOut > 3$, não há diferença significativa de desempenho. Para $maxin = 100$, porém, $maxOut$ não tem influencia alguma e diferentes valores para este parâmetro levam a um mesmo desempenho ruim.

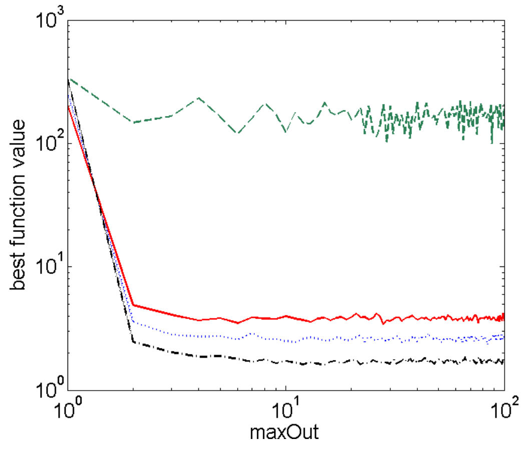


Figure A.9: Avaliação média da função Griewank de 200 dimensões para diferentes valores de $maxOut$. A linha sólida representa o caso onde $maxIn = 1$. A linha pontilhada representa o caso onde $maxIn = 10$. A linha tracejada-e-pontilhada mostra o caso para $maxIn = 50$. A linha tracejada mostra o caso para $maxIn = N$.

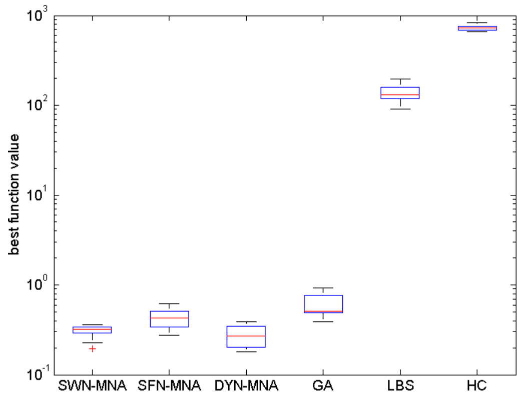
A.6 Comparações e Aplicações

Comparamos os algoritmos apresentados até então com outros algoritmos tradicionais para otimização. Três algoritmos de Rede Memética são utilizados. SWN-MNA implementa a versão de MNA que utiliza redes de mundo pequeno. SFN-MNA implementa a versão que utiliza redes sem escala. Já DYN-MNA utiliza redes dinâmicas. Cada uma dessas versões utiliza os melhores parâmetros encontrados durante os testes apresentados nas respectivas seções.

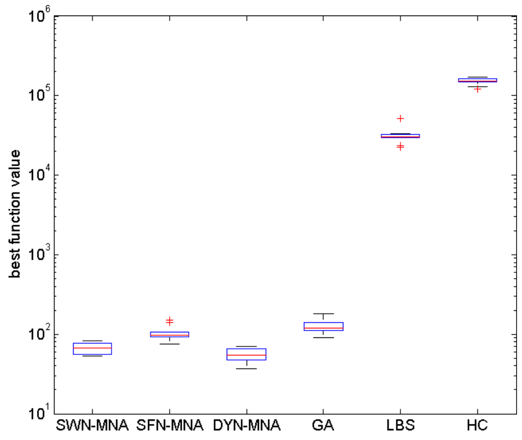
Três algoritmos tradicionais são utilizados para comparação: Subida de Encosta (*Hill-Climbing* - HC), Busca em Feixe Local (*Local Beam Search* - LBS) e Algoritmos Genéticos (*Genetic Algorithsm* - GA).

Estes seis algoritmos foram aplicados a dois cenários, contendo diversos problemas em cada. O primeiro cenário é o de otimização de funções com variáveis reais. O segundo cenário é o Problema do Caixeiro Viajante (TSP).

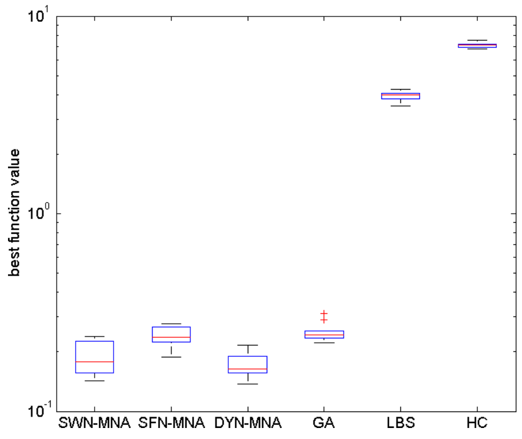
A Figura 8.1 mostra os resultados para a otimização de quatro funções. Nota-se que os MNA se saem melhor que os três algoritmos tradicionais. O GA, no entanto, tem desempenho bastante similar aos MNA. A diferença observada é estatisticamente significativa.



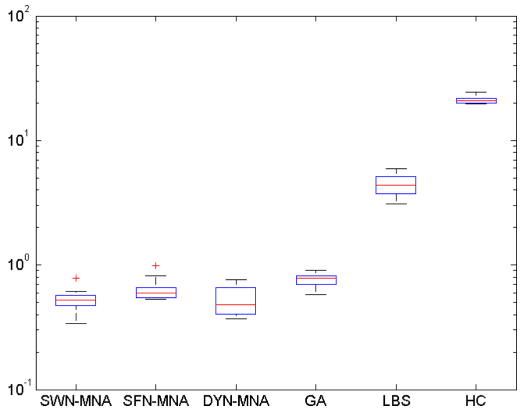
(a) Sphere function



(b) Hyper-Ellipsoid Function



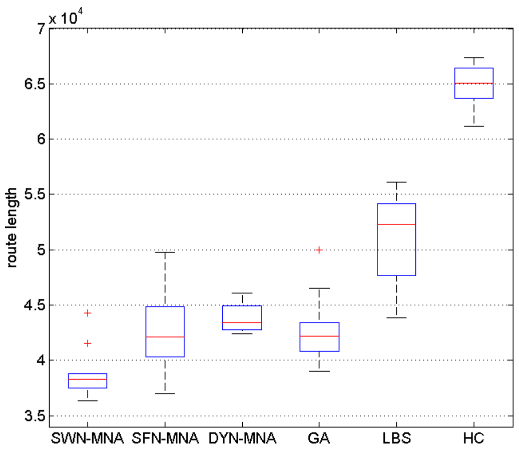
(c) Ackley function



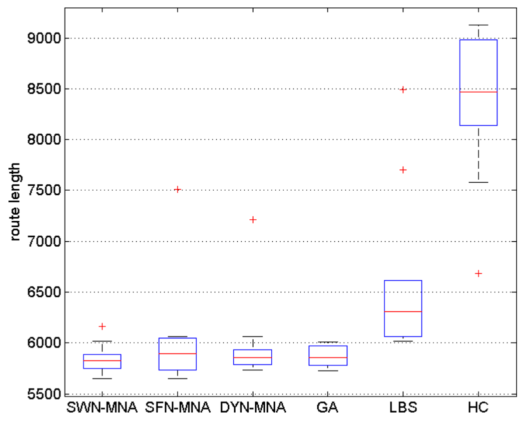
(d) Griewank Function

Figure A.10: Melhores avaliações após 5000 rodadas ao otimizar diferentes funções com diferentes algoritmos.

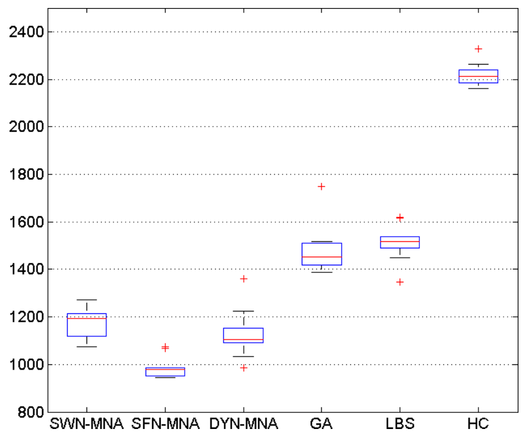
Para o TSP, cujos resultados encontram-se na Figura A.11, observamos comportamento similar. Desta vez, porém, as diferenças são mais acentuadas, especialmente para o maior problema utilizado (com 131 pontos). Nesse caso, o SFN-MNA obteve o mel-



(a) ulysses22



(b) att48



(c) xqf131

Figure A.11: Melhores rotas encontradas para diferentes instâncias do TSP após 1000 rodadas, utilizando diferentes MNAs: Small-World Networks (SWN-MNA), Scale-Free Networks, (SFN-MNA), Dynamic Networks (DYN-MNA). Também são mostrados os resultados para Algoritmos Genéticos (GA), Busca Local em Feixe (LBS) e Subida de Encosta (HC).

hor desempenho entre todos os algoritmos. Para o problema de tamanho intermediário (48 pontos), todos os MNA e o GA tiveram exatamente o mesmo desempenho. Para o menor problema, observou-se uma grande variância, mas o SWN-MNA obteve consistentemente os melhores resultados com baixo desvio padrão.

A.7 Conclusões

Esta tese apresentou um modelo de busca social aplicado a tarefas de encontrar soluções para problemas. O modelo proposto é baseado no conceito de *memes* e na troca de informações através de uma rede como o principal mecanismo de busca. Nesta abordagem, possíveis soluções para um problema (estados) são codificados em vértices de uma rede e novas soluções são geradas pela agregação de elementos de soluções já existentes na rede; tal interação entre vértices é restrita pela topologia da rede.

O principal objetivo do modelo de Rede Memética é prover uma base sobre a qual podem ser exploradas os principais objetivos da tese, que envolve testar a hipótese de que propriedades de rede podem afetar a capacidade de solucionar problemas de sistemas compostos de múltiplos atores sociais. Este objetivo foi motivado pelo cada vez maior número de sistemas que almejam solucionar problemas e que requerem interação social de alguma forma, seja por atores artificiais, como em Sistemas Multiagente, or por seres humanos, como em *crowdsourcing*. Ajustar a rede social correta para melhorar a qualidade e velocidade de convergência desses sistemas e, portanto, nossa principal preocupação.

Através da construção de diversas instâncias de Redes Meméticas, os Algoritmos de Redes Meméticas (MNA), conduzimos experimentos com o objetivo de verificar como propriedades de rede afetam o desempenho dos algoritmos. Mostramos que a presença de uma rede pode ser frequentemente benéfica e sistemas que não permitem comunicação convergem mais lentamente que um sistema que permite tais interações. Assim, com relação a Hipótese 1, concluímos que permitir a comunicação *pode* ser benéfico, e que isso é verdadeiro para uma grande diversidade de configurações de rede; ainda assim, para redes com determinadas propriedades, o desempenho do sistema pode ser arbitrariamente ruim.

Também ficou claro que diversas propriedades de rede tem um papel fundamental em o quão bom um sistema pode ser, confirmando a Hipótese 2. Na maioria dos casos, observamos que permitir um excesso de comunicação é danoso para o desempenho do algoritmo e alterar a rede para restringir taiis interações pode melhorar consideravelmente o resultado final. Por exemplo, no modelo popular de Redes de Mundo Pequeno, descobrimos que o melhor desemepnho é obtido quando o tamanho da vizinhança de cada vértice é pequeno. Ao analisarmos redes dinâmicas, vimos que há uma assimetria em como a informação é difundida: enquanto soluções devem ser construídas usando informações de muitas fontes, há benefícios em restringir o alcance de boas informações na rede. Argumentamos que a maioria desses resultados podem ser explicados pela troca entre o uso de estados conhecidos (*exploitation*) e a exploração de estados não-visitados (*exploration*) no espaço de busca.

Enquanto o modelo de Redes Meméticas é útil para prover informações sobre como atores sociais complexos são influenciados pela rede social subjacente, também mostramos evidências que instâncias do modelo aplicados a problemas de otimização são úteis como algoritmos de otimização de propósito geral. Comparamos MNAs com outros algoritmos de busca tradicionais em funções de teste e cenários reais. Os MNAs mostraram um desempenho melhor que Algoritmos Genéticos na maioria das funções, mas nenhum MNA específico se saiu melhor em todos os cenários, uma evidência de que a rede ideal é dependente do problema em mãos.

Em resumo, as principais contribuições desta tese são:

- Um novo modelo de busca social baseado na troca de memes em uma rede social foi proposto. Até onde sabemos, nenhuma tentativa anterior foi feita em tentar capturar em um modelo as características sociais capazes de solucionar problemas de uma forma geral. Este modelo é útil como um ponto de início em direção a entender o papel de relações sociais em sistemas sociais que visam a solução de problemas.

- Instanciações do modelo proposto foram providas. Algoritmos implementando diversas variações de cada passo do modelo foram propostos, em conjunto com resultados de experimentos validando as implementações e explorando seus diferentes parâmetros. Tais instanciações podem ser diretamente aplicadas a diversos cenários de solução de problemas, tais como otimização numérica e combinatória.

- Resultados foram apresentados sobre o papel de propriedades de redes no desempenho de buscas. Tanto Redes de Mundo Pequeno como Redes Sem Escala foram consideradas e suas principais propriedades foram relacionadas ao desempenho dos algoritmos em diferentes tarefas. Dada a ubiquidade destes modelos de rede em sistemas sociais, estes resultados são passos iniciais em direção a uma teoria mais completa sobre solução de problemas em sistemas sociais.

- Resultados sobre a eficácia de usar o modelo como um algoritmo de busca de propósito geral foram apresentados. Instâncias do modelo mostraram-se melhores que outros algoritmos tradicionais em diversos cenários. Estes resultados, em conjunto com a hipótese de Dawkins de que evolução memética procede em uma velocidade maior que a evolução genética, são evidências de que mais estudos em busca social são interessntes de serem conduzidos como uma maneira de construir solucionadores de problemas ou sistemas de processamento social de informações.

REFERENCES

ADAMIC, L.; ADAR, E. How to search a social network. **Social Neworks**, [S.l.], v.27, n.3, p.187–203, July 2005.

AHN, L. von; DABBISH, L. Labeling images with a computer game. In: SIGCHI-04 CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 2004, New York, NY, USA. **Proceedings...** ACM, 2004. p.319–326.

AHN, L. von; DABBISH, L. Designing games with a purpose. **Commun. ACM**, New York, NY, USA, v.51, n.8, p.58–67, 2008.

AHN, L. von; LIU, R.; BLUM, M. Peekaboom: a game for locating objects in images. In: CHI '06: PROCEEDINGS OF THE SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 2006, New York, NY, USA. **Anais...** ACM, 2006. p.55–64.

ALBERT, R.; JEONG, H.; BARABÁSI, A.-L. Error and attack tolerance of complex networks. **Nature**, [S.l.], v.406, n.6794, p.378–382, July 2000.

APPLEGATE, D.; BIXBY, R.; CHVATAL, V.; COOK, W. **The Traveling Salesman Problem**: a computational study. [S.l.]: Princeton University Press, 2007.

ARAÚJO, R.; LAMB, L. An Information-Theoretic Analysis of Memory Bounds in a Distributed Resource Allocation Mechanism. In: INTERNATIONAL JOINT CONFERENCE IN ARTIFICIAL INTELLIGENCE 2007 (IJCAI-07), 2007. **Proceedings...** [S.l.: s.n.], 2007.

ARAUJO, R. M.; LAMB, L. C. On the effects of network structure in population-based optimization. In: IEEE INTERNATIONAL CONFERENCE ON TOOLS WITH ARTIFICIAL INTELLIGENCE, 20., 2008, Los Alamitos, CA. **Proceedings...** IEEE Computer Society Press, 2008. p.268–271.

ARAUJO, R. M.; LAMB, L. C. Distributed Problem Solving by Memetic Networks. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE (GECCO-2008), 2008, New York. **Proceedings...** ACM Press, 2008. p.599–600.

ARAUJO, R. M.; LAMB, L. C. On the Role of Structured Information Exchange in Supervised Learning. In: EUROPEAN CONFERENCE ON ARTIFICIAL INTELLIGENCE (ECAI 2008), 18., 2008, Patras, Greece. **Proceedings...** IOS Press, 2008. p.120–122.

ARAUJO, R. M.; LAMB, L. C. Memetic Networks: analyzing the effects of network properties in multi-agent performance. In: TWENTY-THIRD AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE (AAAI-08), 2008. **Proceedings...** AAAI Press, 2008. p.3–8.

ARAUJO, R. M.; LAMB, L. C. On the use of memory and resources in minority games. **ACM Trans. Auton. Adapt. Syst.**, New York, NY, USA, v.4, n.2, p.1–23, 2009.

BARABÁSI, A.-L. **Linked**: the new science of networks. [S.l.]: Plume/Penguin Books, 2003.

BARABÁSI, A.-L.; ALBERT, R. Emergence of scaling in random networks. **Science**, [S.l.], v.286, p.509–512, 1999.

BERNERS-LEE, T.; KAGAL, L. The Fractal Nature of the Semantic Web. **AI Magazine**, [S.l.], v.29, n.3, p.29–34, 2008.

BIN, M. V.; YU, B.; SINGH, M. P. Trust and Reputation Management in a Small-World Network. In: IN PROCEEDINGS OF FOURTH INTERNATIONAL CONFERENCE ON MULTIAGENT SYSTEMS, 2000. **Anais...** [S.l.: s.n.], 2000. p.449–450.

BLACKMORE, S. **The Meme Machine**. [S.l.]: Oxford University Press, 2000.

BONABEAU, E.; DORIGO, M.; THERAULAZ, G. **Swarm Intelligence**: from natural to artificial systems. [S.l.]: Oxford University Press, 1999.

BRABHAM, D. C. Crowdsourcing as a Model for Problem Solving: an introduction and cases. **Convergence**, [S.l.], v.14, n.1, p.75–90, 2008.

DASGUPTA, D. (Ed.). **Artificial Immune Systems and Their Applications**. [S.l.]: Springer-Verlag, Inc., 1999.

DAVIS, L. Applying adaptive algorithms to epistatic domains. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE (IJCAI), 9., 1985. **Proceedings...** [S.l.: s.n.], 1985. p.162–164.

DAWKINS, R. **The Selfish Gene**. [S.l.]: Oxford University Press, 1976.

DELGADO, J.; PUJOL, J. M.; SANGÜESA, R. Emergence of coordination in scale-free networks. **Web Intelli. and Agent Sys.**, Amsterdam, The Netherlands, The Netherlands, v.1, n.2, p.131–138, 2003.

DISTIN, K. **The Selfish Meme**: a critical reassessment. [S.l.]: Cambridge University Press, 2004.

DORIGO, M.; BIRATTARI, M.; STÜTZLE, T. Ant Colony Optimization: artificial ants as a computational intelligence technique. **IEEE Computational Intelligence Magazine**, [S.l.], v.1, p.28–39, 2006.

DORIGO, M.; COLORNI, A. The Ant System: optimization by a colony of cooperating agents. **IEEE Transactions on Systems, Man and Cybernetics**, [S.l.], v.26, n.1, p.1–13, 1996.

DORIGO, M.; GAMBARDELLA, L. Ant Colonies for the Traveling Salesman Problem. **BioSystems**, [S.l.], v.43, p.73–81, 1997.

EBERHART, R.; KENNEDY, J. A new optimizer using particle swarm theory. In: SIXTH INTERNATIONAL SYMPOSIUM ON MICRO MACHINE AND HUMAN SCIENCE, 1995. **Proceedings...** [S.l.: s.n.], 1995. p.39–43.

FARMER, J. D.; PACKARD, N. H.; PERELSON, A. S. The immune system, adaptation, and machine learning. **Phys. D**, Amsterdam, The Netherlands, The Netherlands, v.2, n.1-3, p.187–204, 1986.

FISHER, R. A. The use of multiple measurements in taxonomic problems. **Annals Eugen.**, [S.l.], v.7, p.179–188, 1936.

FOGEL, D. **Evolutionary Computation**: toward a new philosophy of machine intelligence. [S.l.]: New York: IEEE, 2000.

FORTUNATO, S. Damage spreading and opinion dynamics on scale-free networks. **Physica A: Statistical Mechanics and its Applications**, [S.l.], v.348, p.683 – 690, 2005.

FU, F.; LIU, L.; WANG, L. Evolutionary Prisoner's Dilemma on heterogeneous Newman-Watts small-world network. **The European Physical Journal B**, [S.l.], v.56, p.367, 2007.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability**: A guide to the theory of NP-completeness. [S.l.]: W. H. Freeman, 1979.

GLOVER, F.; GUTIN, G.; YEO, A.; ZVEROVICH, A. Construction heuristics for the asymmetric TSP. **European Journal of Operational Research**, [S.l.], v.129, p.555–568, 2000.

GLOVER, F. W.; KOCHENBERGER, G. A. **Handbook of Metaheuristics (International Series in Operations Research & Management Science)**. [S.l.]: Springer, 2003.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. [S.l.]: Boston: Addison-Wesley, 1989.

HART, W. E. **Adaptive global optimization with local search**. 1994. Tese (Doutorado em Ciência da Computação) — , La Jolla, CA, USA.

HAYKIN, S. **Neural Networks**: a comprehensive foundation. 2nd.ed. [S.l.]: Prentice Hall, 1998.

HAYKIN, S. **Redes Neurais**: princípios e prática. 2.ed. [S.l.]: Bookman, 2001.

HOFFMANN, L. Crowd Control. **Communications of the Association for Computing Machinery**, [S.l.], v.52, n.3, p.16–17, 2009.

HOLLAND, J. Escaping brittleness: the possibilities of general purpose machine learning algorithms applied to parallel rule-based systems. In: **Machine Learning**: an artificial intelligence approach. [S.l.]: Kaufmann, 1986.

HOLLAND, J. **Adaptation in Natural and Artificial Systems**: an introductory analysis with applications to biology, control, and artificial intelligence. [S.l.]: The MIT Press, 1992.

KEELING, M. J. The effects of local spatial structure on epidemiological invasions. **Proc Biol Sci**, [S.l.], v.266, n.1421, p.859–867, Apr 1999.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by Simulated Annealing. **Science**, [S.l.], v.220, n.4598, p.671–680, 1983.

KIRLEY, M. Evolutionary minority games with small-world interactions. **Physica A: Statistical Mechanics and its Applications**, [S.l.], v.365, n.2, p.521 – 528, 2006.

KLEINBERG, J.; TARDOS, E. Balanced outcomes in social exchange networks. In: STOC '08: PROCEEDINGS OF THE 40TH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, 2008, New York, NY, USA. **Anais...** ACM, 2008. p.295–304.

KLEINFELD, J. S. The small world problem. **Society**, [S.l.], v.39, n.2, p.61–66, January 2002.

KUPERMAN, M.; ABRAMSON, G. Small world effect in an epidemiological model. **Phys Rev Lett**, [S.l.], v.86, n.13, p.2909–2912, Mar 2001.

LAZER, D.; FRIEDMAN, A. The hare and the tortoise: the network structure of exploration and exploitation. In: PROCEEDINGS OF THE 2005 NATIONAL CONFERENCE ON DIGITAL GOVERNMENT RESEARCH, 2005., 2005. **Anais...** Digital Government Society of North America, 2005. p.253–254.

LAZER, D.; PENTLAND, A.; ADAMIC, L.; ARAL, S.; BARABASI, A.-L.; BREWER, D.; CHRISTAKIS, N.; CONTRACTOR, N.; FOWLER, J.; GUTMANN, M.; JEBARA, T.; KING, G.; MACY, M.; ROY, D.; ALSTYNE, M. V. SOCIAL SCIENCE: computational social science. **Science**, [S.l.], v.323, n.5915, p.721–723, February 2009.

LEAKE, D. (Ed.). **AI Magazine**. [S.l.]: AAAI Press, 2008. v.29, n.3.

LERMAN, K. Social Information Processing in News Aggregation. **IEEE Internet Computing**, [S.l.], v.11, n.6, p.16–28, 2007.

LESKOVEC, J.; ADAMIC, L. A.; HUBERMAN, B. A. The dynamics of viral marketing. In: EC '06: PROCEEDINGS OF THE 7TH ACM CONFERENCE ON ELECTRONIC COMMERCE, 2006, New York, NY, USA. **Anais...** ACM, 2006. p.228–237.

LILIJEROS, F.; EDLING, C.; AMARAL, L.; STANLEY, E.; ÅBERG, Y. The web of human sexual contacts. **Nature**, [S.l.], v.411, p.907–908, 2001.

MACNISH, C.; YAO, X. Direction matters in high-dimensional optimisation. In: IEEE CONGRESS ON EVOLUTIONARY COMPUTATION (CEC) 2008., 2008. **Proceedings...** [S.l.: s.n.], 2008. p.2372–2379.

MAGDON-ISMAIL, M.; ATIYA, A. F. The Early Restart Algorithm. **Neural Comput.**, Cambridge, MA, USA, v.12, n.6, p.1303–1312, 2000.

MASON, W.; JONES, A.; GOLDSTONE, R. Propagation of Innovations in Networked Groups. **Journal of Experimental Psychology: General**, [S.l.], v.137, n.3, p.422–433, 2008.

MASUDA, N.; AIHARA, K. Spatial prisoner's dilemma optimally played in small-world networks. **Physics Letters A**, [S.l.], v.313, p.55–61, 2007.

MATSUMOTO, M.; NISHIMURA, T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. **ACM Trans. Model. Comput. Simul.**, New York, NY, USA, v.8, n.1, p.3–30, 1998.

MCGRATH, C.; KRACKHARDT, D.; BLYTHE, J. Visualizing complexity in networks: seeing both the forest and the trees. **Connections**, [S.l.], v.25, p.37–47, 2003.

MENCZER, F.; WU, L.-S.; AKAVIPAT, R. Intelligent Peer Networks for Collaborative Web Search. **AI Magazine**, [S.l.], v.29, n.3, p.35–46, 2008.

MICHALEWICZ, Z. **Genetic Algorithms + Data Structures = Evolution Programs**. 3rd.ed. [S.l.]: Berlin: Springer, 1996.

MICHALEWICZ, Z.; FOGEL, D. B. **How to Solve It**: modern heuristics. [S.l.]: Springer, 2004.

MILGRAM, S. The Small World Problem. **Psychology Today**, [S.l.], v.1, n.1, p.60–67, 1967.

MITCHELL, M. Complex systems: network thinking. **Artif. Intell.**, [S.l.], v.170, n.18, p.1194–1212, 2006.

MITCHELL, T. **Machine Learning**. [S.l.]: McGraw-Hill, 1997.

MOORE, C.; NEWMAN, M. Epidemics and percolation in small-world networks. **Phys. Rev. E**, [S.l.], v.61, p.5678–5682, 2000.

MOSCATO, P. **On evolution, search, optimization, genetic algorithms and martial arts**: towards memetic algorithms. [S.l.]: Caltech Concurrent Computation Program, California Institute of Technology, 1989. (826).

MUSELLI, M. A Theoretical Approach to Restart in Global Optimization. **J. of Global Optimization**, Hingham, MA, USA, v.10, n.1, p.1–16, 1997.

NEWMAN, M.; BARABÁSI, A.-L.; WATTS, D. (Ed.). **The Structure and Dynamics of Networks**. [S.l.]: Princeton University Press, 2006.

OLFATI-SABER, R.; FAX, J. A.; MURRAY, R. M. Consensus and Cooperation in Networked Multi-Agent Systems. **Proceedings of the IEEE**, [S.l.], v.95, n.1, p.215–233, 2007.

OLORUNDA, O.; ENGELBRECHT, A. Measuring exploration/exploitation in particle swarms using swarm diversity. In: IEEE CONGRESS ON EVOLUTIONARY COMPUTATION (CEC 2008), 2008. **Proceedings...** [S.l.: s.n.], 2008. p.1128–1134.

ONER, M. K.; GARIBAY, I. I.; WU, A. S. Mating networks in steady state genetic algorithms are scale free. In: GECCO '06: PROCEEDINGS OF THE 8TH ANNUAL CONFERENCE ON GENETIC AND EVOLUTIONARY COMPUTATION, 2006, New York, NY, USA. **Anais...** ACM, 2006. p.1423–1424.

PARAMESWARAN, M.; WHINSTON, A. B. Social Computing: an overview. **Communications of the Association for Information Systems**, [S.l.], v.19, p.762–780, 2007.

PEARL, J. **Heuristics**: intelligent search strategies for computer problem solving. [S.l.]: Addison-Wesley Pub, 1984.

RADEV, D. R.; MIHALCEA, R. Networks and Natural Language Processing. **AI Magazine**, [S.l.], v.29, n.3, p.16–28, 2008.

RARDIN, R. L.; UZSOY, R. Experimental Evaluation of Heuristic Optimization Algorithms: a tutorial. **Journal of Heuristics**, [S.l.], v.7, n.3, p.261–304, May 2001.

RODRÍGUEZ-TRELLES, F.; TARRO, R.; AYALA, F. J. Origin and Evolution of Spliceosomal Introns. **Annu Rev Genet**, [S.l.], Jun 2006.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence**: a modern approach. [S.l.]: Prentice Hall, 2002.

SCHNEEBERGER, A.; MERCER, C. H.; GREGSON, S. A.; FERGUSON, N. M.; NYAMUKAPA, C. A.; ANDERSON, R. M.; JOHNSON, A. M.; GARNETT, G. P. Scale-free networks and sexually transmitted diseases: a description of observed patterns of sexual contacts in britain and zimbabwe. **Sex Transm Dis**, Department of Infectious Disease Epidemiology, Faculty of Medicine, Imperial College, London, UK., v.31, n.6, p.380–387, June 2004.

SPALL, J. C. **Introduction to Stochastic Search and Optimization**. [S.l.]: Wiley-Interscience, 2003.

STROGATZ, S. **Sync**: the emerging science of spontaneous order. [S.l.]: New York: Theia, 2003.

SUROWIECKI, J. **The Wisdom of Crowds**. [S.l.]: Anchor, 2005.

TAMHANE, A.; DUNLOP, D. **Statistics and Data Analysis**: from elementary to intermediate. [S.l.]: Prentice Hall, 1999.

WANG, J.; GONZALEZ, K. D.; SCARINGE, W. A.; TSAI, K.; LIU, N.; GU, D.; LI, W.; HILL, K. A.; SOMMER, S. S. Evidence for mutation showers. **Proc Natl Acad Sci U S A**, [S.l.], v.104, n.20, p.8403–8408, May 2007.

WATTS, D. **Six Degrees**: the science of a connected age. [S.l.]: W. W. Norton and Company, 2003.

WATTS, D. J. **Small Worlds - The Dynamics of Networks between Order and Randomness**. [S.l.]: Princeton University Press, 1999.

WATTS, D. J.; STROGATZ, S. H. Collective dynamics of 'small-world' networks. **Nature**, [S.l.], v.393, n.6684, p.440–442, Jun 1998.

WILLINGER, W.; ALDERSON, D.; DOYLE, J. C. Mathematics and the Internet: a source of enormous confusion and great potential. **Notices of the AMS**, [S.l.], v.56, n.5, May 2009.

WOLPERT, D. H.; MACREADY, W. G. No free lunch theorems for optimization. **IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION**, [S.l.], v.1, n.1, p.67–82, 1997.