Jeannette Janssen
Paweł Prałat (Eds.)

Combinatorial and Algorithmic Aspects of Networking

LNCS 4852

4th Workshop, CAAN 2007
Halifax, Canada, August 2007
Revised Papers

Springer

Lecture Notes in Computer Science 4852

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison
Lancaster University, UK

Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler
University of Surrey, Guildford, UK

Jon M. Kleinberg
Cornell University, Ithaca, NY, USA

Friedemann Mattern
ETH Zurich, Switzerland

John C. Mitchell
Stanford University, CA, USA

Moni Naor
Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz
University of Bern, Switzerland

C. Pandu Rangan
Indian Institute of Technology, Madras, India

Bernhard Steffen
University of Dortmund, Germany

Madhu Sudan
Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos
University of California, Los Angeles, CA, USA

Doug Tygar
University of California, Berkeley, CA, USA

Moshe Y. Vardi
Rice University, Houston, TX, USA

Gerhard Weikum
Max-Planck Institute of Computer Science, Saarbruecken, Germany

Jeannette Janssen Paweł Prałat (Eds.)

Combinatorial and Algorithmic Aspects of Networking

4th Workshop, CAAN 2007
Halifax, Canada, August 14, 2007
Revised Papers

Springer

Volume Editors

Jeannette Janssen
Paweł Prałat
Dalhousie University
Department of Mathematics and Statistics
Halifax, NS, B3H 3J5, Canada
E-mail: {janssen,pralat}@mathstat.dal.ca

Library of Congress Control Number: 2007941336

CR Subject Classification (1998): F.1.1, F.2.1-2, C.2, G.2.1-2, E.1

LNCS Sublibrary: SL 5 – Computer Communication Networks and Telecommunications

ISSN 0302-9743
ISBN-10 3-540-77293-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-77293-4 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2007
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12204793 06/3180 5 4 3 2 1 0

Preface

The advent of the Internet has opened up a wealth of applications, but also given rise to a host of new problems. Many of those problems have led to exciting new research directions in mathematics and theoretical computer science, especially in the areas of combinatorics and algorithms. The Fourth Workshop on Combinatorial and Algorithmic Aspects of Networking (CAAN 2007) was organized to be a place where the latest research developments on all aspects of networking could be presented. The topics covered were diverse, with talks on strategies for searching in networks, for cleaning networks of unwanted intruders, on different routing strategies, and on scheduling and load balancing. The workshop started with an invited lecture by Peter Winkler of Dartmouth College, who gave a general talk on a topic related to probability, a concept central to network modeling and managing. The afternoon opened with a short invited talk by Alejandro López-Ortiz, who gave an overview of various issues in designing resilient backbone networks.

CAAN 2007 took place on August 14, 2007, at Dalhousie University in Halifax, Nova Scotia, Canada, co-located with the Workshop on Algorithms and Data Structures (WADS 2007). Three previous CAAN workshops were held in Chester, UK (CAAN 2006), Waterloo, Ontario, Canada (CAAN 2005), and in Banff, Alberta, Canada (CAAN 2004), respectively.

In response to the call for paper we received 17 submissions. Each submission was reviewed by three referees. Almost all submissions were relevant to the topic of the workshop, and most contained interesting ideas. Based on the reviews of the referees we accepted nine papers for presentation at the workshop and inclusion in this volume. The volume also includes an abstract of the invited talk by Peter Winkler and an invited paper by Alejandro López-Ortiz.

We would like to thank all those that helped to make this workshop a success, with special thanks to Anne Publicover, our administrative assistant. Thanks to the Atlantic Association for Research in Mathematics (AARMS) for financial support. Many thanks to Andrei Voronkov for providing the EasyChair conference system; with this system, managing the electronic submissions and the refereeing process has been a breeze. Thanks also to Norbert Zeh, the Local Arrangements Chair of WADS 2007. Finally, we thank all participants in the workshop, all authors of the contributed papers, and especially the invited speakers for their contribution in making CAAN 2007 into a collegial and stimulating platform for new ideas about networks.

October 2007

Jeannette Janssen
Paweł Prałat

Organization

Steering Committee

Andrei Broder	Yahoo! Inc., USA
Angèle Hamel	Wilfrid Laurier University, Canada
Srinivasan Keshav	University of Waterloo, Canada
Alejandro López-Ortiz	University of Waterloo, Canada
Rajeev Motwani	Stanford University, USA
Ian Munro	University of Waterloo, Canada

Program Committee

Dimitris Achlioptas	UC Santa Cruz, USA
Anthony Bonato	Wilfrid Laurier University, Canada
Allan Borodin	University of Toronto, Canada
Colin Cooper	King's College, UK
Erik Demaine	MIT, USA
Thomas Erlebach	University of Leicester, UK
Angèle Hamel	Wilfrid Laurier University, Canada
Jan van den Heuvel	London School of Economics, UK
Klaus Jansen	Universität Kiel, Germany
Jeannette Janssen	Dalhousie University, Canada
Valerie King	University of Victoria, Canada
Danny Krizanc	Wesleyan University, USA
Stefano Leonardi	Università di Roma "La Sapienza", Italy
Alejandro López-Ortiz	University of Waterloo, Canada
Lata Narayanan	Concordia University, Canada
Richard Nowakowski	Dalhousie University, Canada
Paweł Prałat	Dalhousie University, Canada
Sunil Shende	Rutgers-Camden, USA
Eva Tardos	Cornell University, USA

Table of Contents

Invited Lectures (Abstracts)

Contributed Papers

Luck vs. Skill
(Long Invited Talk)

Peter Winkler

Dartmouth College, Hanover, NH, USA
peter.winkler@dartmouth.edu

Abstract. Recent legislation in the US regarding gambling over the web has led to renewed interest in the question of which games are games of skill. We take a statistical approach to the problem, defining the skill index of a game to be the average amount of playing time after which variance due to chance and variance due to skill differences are equal.

We then look at tournament results for championship-level duplicate bridge, PGA golf, and duplicate poker, as well as some simulated toy games, to see how their skill indices compare.

Biography

Peter Winkler is Professor of Mathematics and Computer Science at Dartmouth College and Albert Bradley Third Century Professor in the Sciences.

A winner of the Mathematical Association of America's Lester R. Ford Award for mathematical exposition, Dr. Winkler is the author of about 125 mathematical research papers and holds a dozen patents in computing, cryptology, holography, optical networking and marine navigation. His research papers are primarily in combinatorics, probability and the theory of computing, with forays into statistical physics.

Dr. Winkler received his BA from Harvard summa cum laude in mathematics, then after a stint in the US Navy, his PhD from Yale as a student of Abraham Robinson and Angus Macintyre. He joined the faculties of Stanford and then Emory University, where he became Professor and Chairman of Mathematics and Computer Science. In 1989 he left academia for industry, returning in 2004.

When not proving theorems or enjoying his family, Winkler is generally found on a squash court or playing and composing ragtime piano music. He collects puzzles both mechanical and mathematical, the latter appearing in two popular books. In some circles Winkler is notorious as the inventor of cryptologic techniques for the game of bridge, which have now been declared illegal for tournament play in most of the western world.

J. Janssen and P. Prałat (Eds.): CAAN 2007, LNCS 4852, p. 1, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Valiant Load Balancing, Benes Networks and Resilient Backbone Design
(Short Invited Talk)

Alejandro López-Ortiz

University of Waterloo, Waterloo, ON, Canada
alopez-o@uwaterloo.ca

Abstract. At any given time, the traffic on the network can be described using a traffic matrix. Entry $a_{i,j}$ in the matrix denotes the traffic originating in i with destination j currently in the network. As traffic demands are dynamic, the matrix itself is ever changing. Traditionally network capacity has been deployed so that it can support any traffic matrix with high probability, given the known traffic distribution patterns. Recently the need for resilience and reliabilibility of the network for mission critical data has brought the need for backbone capacity that can support all traffic matrices. In this talk we give an overview of the state of the art on networks and routing schemes with this property.

Biography

Alejandro Lpez-Ortiz received his B.Math. degree from the National University of Mexico (UNAM) in 1989, and his M.Math. and Ph.D. from the University of Waterloo in 1990 and 1996 respectively. In his research he has combined the development of theoretical tools and efficient algorithms with real life applications. He has been a faculty member in the School of Computer Science, University of Waterloo since 2001 (promoted to Associate professor with tenure in 2004) and was Director of Research at Internap network services corporation in Seattle. His research addresses questions of both theoretical and practical relevance such as robot navigation, search engines, data streams and the internet. He is currently co-chair of the DIMACS three year special focus on Algorithmic Foundations of the Internet, jointly with Jennifer Rexford from Princeton University and Rebecca Wright of Rutgers University.

J. Janssen and P. Prałat (Eds.): CAAN 2007, LNCS 4852, p. 2, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Valiant Load Balancing, Capacity Provisioning and Resilient Backbone Design

Alejandro López-Ortiz

Cheriton School of Computer Science
University of Waterloo
Waterloo, ON N2L 3G1 Canada
alopez-o@uwaterloo.ca

Abstract. The two main alternatives for achieving high QoS on the public internet are (i) admission control and (ii) capacity overprovisioning. In the study of these alternatives the implicit (and sometimes explicit) message is that ideally, QoS issues should be dealt with by means of sophisticated admission control (AC) algorithms, and only because of their complexity providers fall on the simpler, perhaps more cost-effective, yet "wasteful" solution of capacity overprovisioning (CO) (see e.g. Olifer and Olifer [Wiley&Sons, 2005], Parekh [IWQoS'2003], Milbrandt et al. [J.Comm. 2007]). In the present survey we observe that these two alternatives are far from being mutually exclusive. Rather, for data critical applications, a substantial amount of "overprovisioning" is in fact a fundamental step of any safe and acceptable solution to QoS and resiliency requirements. We observe from examples in real life that in many cases large amounts of overprovisioning are already silently deployed within the internet domain and that in some restricted network settings they have become accepted practice even in the academic literature. Then we survey the main techniques currently in use to compute the provisioning capacities required in a resilient high QoS network.

1 Introduction

In the quality-of-service literature (QoS) two main alternatives are given for achieving high QoS on the public internet. These are (i) admission control and (ii) capacity overprovisioning. In the study of these alternatives the implicit (and sometimes explicit) message is that ideally, QoS issues should be dealt with by means of sophisticated admission control (AC) algorithms, and only because of their complexity providers fall on the simpler, perhaps more cost-effective, yet "wasteful" solution of capacity overprovisioning (CO) (see e.g. [22,23,20]). AC researchers often express the hope that this situation will eventually remedy itself and that sophisticated AC algorithms will do away with the need for bandwidth overprovisioning (e.g. [8]). Only recently Menth et al. in a SIGCOMM'06 paper gave evidence that CO might not be as undesirable as previously thought [19].

In the present survey we observe that these two alternatives are far from being mutually exclusive. Rather, for data critical applications, a substantial amount

J. Janssen and P. Prałat (Eds.): CAAN 2007, LNCS 4852, pp. 3–12, 2007.
© Springer-Verlag Berlin Heidelberg 2007

of "overprovisioning" is in fact a fundamental step of any safe and acceptable solution to QoS requirements. Indeed, a survey of common practices in the field suggests that this observation has been arrived to independently and empirically by network engineers in various settings within the Internet and otherwise, yet the QoS literature so far does not reflect this discovery nor has it attempted to explain its root causes.

We observe from examples in real life that in many cases large amounts of overprovisioning are already silently deployed within the internet domain and that in some restricted network settings they have become accepted practice even in the academic literature. In other words, distate for overprovisioning is not a universally held belief outside the QoS domain. In fact, the telephony network which is considered a classical example of AC is in practice heavily overprovisioned and actual AC policy is rarely relied upon even though it is deployed on the network [21]. Then we survey the main techniques currently in use to compute the provisioning capacities required in a resilient high QoS network. We term this amount rightprovisioning. Lastly, we give reasons why QoS over a rightprovisioned network has different needs and goals than those currently addressed by admission control and other such mechanisms.

2 Internet QoS

The two main mechanisms for achieving a desired level of service on the internet are admission control and capacity overprovisioning. QoS on the network allows the user to make choices as to the level of service it requires. Typical parameters are: data rate (bandwidth), availability, end-to-end delay (latency), variation of end-to-end delay (jitter), and packet loss rate [8].

2.1 Capacity Overprovisioning

Capacity overprovisioning consists in increasing available bandwidth until it is large enough to sustain the vast majority of peaks in demand. Depending on the level of reliability desired this can be as low as 25% above average data rate to handle 95% of all traffic demands without loss, 50% extra bandwidth to carry 99% of traffic, and double the average bandwidth to meet 99.99% or higher of all traffic demands without loss (see e.g [32]). This last choice, meaning 50% utilization of the pipe, is often anecdotically referred to as the upper limit of utilization currently acceptable by large ISPs, with the load on an average link often being well below that [7,8].

In contrast, in the QoS literature overprovisioning is considered a simple but wasteful solution to QoS demands. For example, to quote from a computer networks textbook [22]:

> Overprovisioned services keep the network infrastructure simple (no additional tools and configurations) but are wasteful as 60-70% of potential network resources are not in use. Under such conditions the best-effort

service on a standard IP network turns out to be good enough for all network applications including time-sensitive ones.

Indeed the term "overprovisioning" itself has the implication that more capacity than what was required was provisioned and hence it ends up being wasted. Yet, subutilization of a resource alone does not imply it was provisioned. In fact, most mission critical applications such as avionics routinely rely on highly redundant configurations, which under normal operational procedures are not used. For example an ocean liner arriving safely to port did not utilize its life boats, yet no one would argue that they were thus "overprovisioned".

2.2 Admission Control

Admission control is mostly about using resource reservation and limits on traffic volume to preven overload on the network. It is predicated on the basis that not all network traffic is time-sensitive and mission critical. The AC alternative to overprovisioning is denying resources to non crucial flows. Typical examples of time-sensitive traffic are real time flows (e.g. video/audio streaming, IP telephony) and high value transactions (stock trades, last bid at an online auction). Packets are assigned a priority value with higher priority packets being given preferencial service. Yet a look at the historical development of the internet suggests that, over the years, the majority of the traffic overtime has become more time sensitive and mission critical. Recall that in the original internet the majority of traffic was smtp (email) and nntp (usenet) based. These protocols have acceptable delay tolerances from several minutes to as long as days. Web traffic which is served interactively has acceptable delays in the 10 second or less range. VoIP and other streaming traffic have subsecond delay tolerances.

As more of the nation infrastructure migrates to the public internet, a disruption in the network has larger consequences. The financial, defense, telephone, commerce, government, and business infrastructure now rely on the availability of the Internet to operate properly. Even a seemingly non-mission critical application such as a standard home network connection which might have been initially deployed for one parent's non-time sensitive email (smtp) traffic later on became used by the kids for highly time-sensitive gaming and audio streaming as well as by a parent bidding in online auctions for objects worth thousands of dollars, and as of recently is being used as a carrier for VoIP services which means that emergency calls (911 or to the family doctor) are routed over it. These last type of calls are both time-sensitive mission critical. Thus, it is not far-fetched to envision a world in which the majority of the traffic will be labeled as time sensitive and hence the savings from AC would be minimal, since not many flows can be dropped. This would make packet classification schemes at admission control points progressively more difficult and less useful, the majority of the traffic is critical to start with.

This suggests that as more data exchanges migrate to the Internet infrastructure, the need for higher reliability will further increase while the ability to differentiate between types of traffic will continue to decrease.

3 Rightprovisioning

Capacity overprovisioning is common place in the current internet [1,8,18,12]. AC based solutions remain unused while anecdotal evidence suggests that CO is the prefered method for QoS delivery in the commercial internet. Currently QoS due to CO is such that no packets are dropped in the backbones [8,15,4]. Packet loss occurs mostly in the interface between the end points of the network and the large ISP providers. As providers have focused on ensuring that there is sufficient deployed capacity rather than on implementing admission control solutions. ISPs will go to the extent of delaying by several months the start of connectivity for a new customer to ensure that there is enough capacity on the network to support the bandwidth demands of the new customer (this can be argued is a crude form of admission control). In other words, currently ISPs find that CO is a cost effective way to achieve QoS.

While most of the literature is critical of CO as a solution of QoS, recent developments suggest that even in theory its performance is better than originally thought. Bhagat observes that in certain settings overprovisioning seems to be a better answer to the performance needs from users, and indeed he goes as far as questioning the need for admission control based QoS solutions [6]. In a recent breakthrough paper in SIGCOMM'06 Menth et al. [19] show that if overprovisioned capacity is also used to achieve resilience against network failures, then the demands in terms of bandwidth of failure-resilient AC and CO schemes are comparable, as the overprovisioned capacity can be deployed for various uses depending on the type of congestion and/or failure detected. In sum, so far we have argued that

1. selective admission as required by AC is becoming increasingly less of an option at the backbone level since traffic is increasingly time and mission critical,
2. that CO in large trunks is already in place and provides excellent QoS within the core of the network,
3. that as such its effectiveness is well supported by established practice, and that
4. the academic literature has started to explain why CO is such an effective solution.

The question then remains what is the proper level of overprovisioning, i.e. rightprovisioning. Currently the model most commonly in use is a statistical guarantee of the probability of connection denial. We argue that the right metric is to provide enough capacity so that any valid traffic matrix can be realized.

Definition 1. $e_1, \ldots, e_n$ n s_i r_i traffic matrix $A = [a_{ij}]$ a_{ij} e_i e_j

Definition 2. valid $\sum_{j=1}^{n} a_{ij} \leq s_i$ $\sum_{i=1}^{n} a_{ij} \leq r_j$

In the past providers have deployed enough capacity to handle the average traffic matrix or a percentage of traffic matrix configurations (say 95% of the time the traffic matrix should cause no loss in traffic). Since the aim is to provide connectivity for the worst case traffic matrix we need to determine what is the minimum or most cost efficient capacity that satisifies this requirement. We could simply consider the sum of all contracted capacity by users, however this does not take into account that currently connectivity is provided in an average fashion, typically at a certain average rate per month with a maximum burstable rate.

In the new regime, two types of traffic would be provisioned. Traffic of type A, which is mission critical and always available at the contracted capacity and traffic of type B, at an average contracted capacity but rate-controlled depending on connectivity characteristics. In essence this could be thought as rate modulation over a pipe carrying type B traffic, not unlike in nature and effects to that performed by a modem in the presence of high levels of line noise. Observe that this establishes a very simple form of admission control. Traffic of type A would be unavailable at most on the order of subsecond to few seconds per year range (seven to eight nines of reliability). At the same time the entire contracted capacity should be generally available, with traffic of type B being flow rate controlled in the order of a half a minute to a few minutes a year (five to six nines range). This last is the current level of service reliability that the telephone network claims to have, even though arguably telephone traffic is less time critical than many of the current uses of the network. It is worthy of note that the telephone network operates at 33% capacity [21] and that the amount of admission control is minimal. For example "on Monday, Dec. 2, 1991, which was the busiest day for the AT&T network until then, of 157.5 million calls, only 228 were blocked on intercity connections" (from [3] as quoted by [21]). Our proposal parallels this design choice.

Interestingly enough, worst-case traffic matrix $n \times n$ capacity already exists in certain network settings. In the LAN the proper amount of overprovisioning has evolved to be such that, given n nodes on an Ethernet, a complete set of $n/2$ disjoint pairs can communicate at full speed. Recall that this was not always the case, as the original co-axial ethernet only had sufficient capacity for a single pair to communicate freely at full capacitywithout collision; eventually star switches with higher capacity buses became commonplace, and currently common $n \times n$ crossbar or Beneš network switches have the ability to sustain $n/2$ disjoint pairs of communication [2]. Similarly Network Access Points (NAPs) as well as cores of large corporate networks often consist of an optical ring providing enough capacity for all possible crossconnects. This is not unique to the internet. In the 1970s telephone networks deployed switches with $n \times n$ capacity at certain critical points of the infrastructure [26].

For statistical guarantees the law of large numbers can be used to determine the maximum simultaneous demand that may originate, on the aggregate, from a neighborhood of nodes sharing an entry point to the ISP backbone. This is repeated for all entry points into the backbone and then a full $n \times n$ bandwidth capability over those averages can be deployed. The size of such an $n \times n$ network is well understood. We discuss in detail the various known alternatives in Section 5.

Lastly, as Menth et al. observed, redundant equipment can be deployed for multiple purposes, so long as the probability of failure of such equipment is independent [19]. This amortizes the additional cost of redundant equipment. In particular redundant capacity can be used to circumvent router and link failures (digging). This has been observed to reduce the amount of apparent "subutilization". As well, secondary sources of traffic which can be quenched at the source point can be sent over the spare capacity. Examples of this are CDN content and remote backup data which are resilient under short time delays. Anecdotal evidence suggests that spam traffic is delivered at off-peak times by certain ISPs using deployed overcapacity.

4 QoS and AC in a Rightprovisioned World

Observe that we do not claim that overprovisioning at the backbone is sufficient to achieve all QoS requirements, nor would it make AC trivial. This is in contraposition to claims to that end in the literature, e.g. "only when the ratio of resources at the edges of a network to those available in the core of a network becomes high is the problem of service differentiation interesting, [...] when this ratio is low, any QoS mechanism appears redundant as most users receive the service they require anyway, and so the cost introduced by a QoS scheme appears unjustified, and research into QoS mechanisms appears unnecessary" [8].

For one, as the network is used for more life-critical operations such as VoIP phone calls (911), financial transactions (stock exchange), remote surgery, and air traffic system, perhaps even carrier grade reliability is not good enough. It is not hard to envision demands for reliability reaching into the 99.999999% range (in fact today it is possible to provision bandwidth with a stated 100% reliability guarantee in the sense that . . amount of downtime is contractually heavily penalized). Such high levels of reliability will require overprovisioning, multihoming, redundancy, admission control and intelligent routing, though the types of solutions required, their price/performance ratio and their goals change. As well, end users will still, on occasion, attempt to send or receive more time critical data that is feasible given their available network connectivity. Admission control in such situations will be needed to prioritize say, a 911 VoIP call (type A traffic) over downloading email (type B traffic).

Admission control starts from the assumption that congestion will always take place at the edge given the reduced capacities of the endpoint as compared to the capacity of the entire network (i.e. the need to send or receive more data than

what we have capacity for). What this work argues is that congestion should only take place at the edge and that CO is the way to ensure this.

The model we propose assumes that all packets reaching the network core are assumed to be critical and hence failure of delivery is not an option. Within the core there would be no differentiated services with AC taking place as a weak and simplified form of resource reservation: if the packet is admitted, it can be delivered. The end node would send data in one of two modes: normal mode in which all traffic is accepted without need for any AC intervention and exceptional mode in which the application/user is alerted of a temporary service disruption and given the choice to proceed with the communication at full speed or throttle down for a few seconds (type A or B classification). Incentives such as price differentials can be built in to ensure that the user delays non-essential traffic.

Given the reliability needs detailed above this would occur with a very low probability, in the range of thirty seconds to a few minutes of service disruption per year. Such a rare occurrence means that only the simplest of differentiated services and admission control policies can be justified from the perspective of economic viability. As it has been observed [8] a weak form of AC already takes place in the edges in that providers delay customer activation to ensure that enough capacity is present to satisfy demand. This is a crude yet effective form of denying a transmission request.

As well routing in an overprovisioned network is more complicated as the multiplicity of paths allows for an intelligent choice. This determination does not involve the end point as the network makes best effort for all packets.

5 Valiant Load Balancing and Beneš Networks

Claude Shannon pioneered the study of networks that support $n \times n$ communication pairs. He proved that if the proving that a fabric of $n \log n$ switches is necessary so long as total deployed capacity is linear. Beneš introduced the later termed Beneš networks which match Shannon's lower bound switch [2,10,11,26]. Arora et al. combined Beneš networks with the butterfly network to obtain a similar topology that supports all cross connects in an online fashion, while preserving the efficiency in terms of deployed capacity.

If there are no restrictions in the total deployed capacity then other alternative realizations are possible. Two of the most common being a central high capacity ring and the $n \times n$ crossbar. Pippenger extensively studied the topology of telephone switches that support $n \times n$ connection patterns [24,25,26,27,28].

Valiant proposed an elegant network topology in the context of interprocessor communication networks for parallel computers [33,34]. The network starts with the complete graph on n nodes which trivally can support all independent connection pairs. However it is an inefficient solution as it requires n^2 contracted capacity. Valiant's key observation is that a two phase communication protocol on a complete graph in which every link has capacity $2/n$ suffices. This reduces the total deployed capacity to $2n$ which is a constant times the deployed capacity.

Extensions and generalizations of both Pippenger's and Valiant's work have been the subject of intense study within theoretical computer science [9,14,2] as well as the networks community [17,30,31,29]. The field is now refered to as a Valiant Load Balancing Network and/or as a Virtual Private Network load balancing. The term VPN comes from the fact that VPNs were one the earliest users of the internet requiring high degree of reliability. Shepherd et al. and Prasad et al. have run simulations to determine the effect of VPN load balancing in existing networks, and have observed that peak traffic loads are lowered down while resilience is improved. Many open questions remain, among them

- how to efficiently design an overprovisioned network under realistic cost measures?
- design an overprovisioned network which readily scales under incremental growth?
- given a pre-existing network infrastucture compute the lowest cost links that must be added for the network to support the worst case traffic matrix
- how to add links to an existing network infrastucture in a way that the can serve the dual purpose of worst case traffic matrix provisioning and resiliency under link cuts?
- how to implement the desired routing patterns using the current routing protocols (BGP/IGP/OSPF)?

6 Conclusions

We have argued that given the evolution path of internet traffic, higher levels of reliability will be required. As such admission control schemes which refuse connections are no longer feasible. At the same time we give evidence that capacity overprovisioning with a high probabilistic guarantee of delivery for $n \times n$ traffic is already in place in the internet, though not generally recognized. We also observed that in other network settings such large capacity has been openly, purposely deployed with the full acceptance of theory and practice. We noted that "overprovisioned" capacity can be put to other uses as others have shown [19], and CO is more efficient than generally believed. We give general bounds on the amount of traffic that is required for service guarantees and we term this the network. Lastly we argued that there is still need for QoS and AC policies at the network edge.

References

1. Armitage, G.J.: Revisiting IP QoS: Why do we care, what have we learned. ACM 2003 RIPOS Workshop Report, ACM SIGCOMM Comp. Comm. Rev. vol. 33(5) (October 2003)
2. Arora, S., Leighton, F.T., Maggs, B.M.: On-line Algorithms for Path Selection in a Nonblocking Network. In: Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, pp. 149–158 (May 1990)

3. Ash, G.R.: Dynamic Routing in Telecommunications Networks. McGraw-Hill, New York (1998)
4. Atkinson, R.: QoS vs Bandwidth Overprovisioning. End-to-End mailing list (April 2001)
5. Beneš, V.E.: Optimal rearrangeable multistage connecting networks. Bell System Technical Journal 43, 1641–1656 (1964)
6. Bhagat, S.: QoS: Solution Waiting for a Problem, position paper, Dept of Comp. Sci. Rutgers University
7. Casner, S., Alaettinoglu, C., Kuan, C.-C.: A Fine-Grained View of High-Performance Networking, NANOG 22, http://www.nanog.org/mtg-0105/casner.html
8. Crowcroft, J., Hand, S., Mortier, R., Roscoe, T., Warfield, A.: QoS's Downfall: At the bottom, or not at all! In: Proceedings of the Workshop on Revisiting IP QoS (RIPQoS), at ACM SIGCOMM 2003, August 27, 2003, Karlsruhe, Germany (2003)
9. Dellamonica Jr., D., Kohayakawa, Y.: An algorithmic Friedman–Pippenger theorem on tree embeddings and applications to routing. In: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pp. 1038–1044 (2006)
10. Feldman, P., Friedman, J., Pippenger, N.: Non-blocking networks. In: Proceedings of the 18th Annual ACM Symposium on Theory of Computing, pp. 247–254 (May 1986)
11. Feldman, P., Friedman, J., Pippenger, N.: Wide-sense nonblocking networks. SIAM Journal of Discrete Mathematics 1, 158–173 (1988)
12. Fraleigh, C., Tobagi, F., Diot, C.: Provisioning IP Backbone Networks to Support Latency Sensitive Traffic. In: Proceedings of IEEE Infocom 2003, San Francisco, USA (2003)
13. Gibbens, R., Kelly, F.: Resource pricing and the evolution of congestion control. Automatica 35 (1999)
14. Gupta, A., Kleinberg, J.M., Kumar, A., Rastogi, R., Yener, B.: Provisioning a virtual private network: a network design problem for multicommodity flow. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, pp. 389–398 (2001)
15. Van Jacobson: A New View of Networking, Google Tech Talk (2007)
16. Keshav, S.: An Engineering Approach to Computer Networking. Addison-Wesley, Reading
17. Keslassy, I., Chang, C.-S., McKeown, N., Lee, D.-S.: Optimal load-balancing. In: Proceedings of IEEE Infocom, pp. 1712–1722 (2005)
18. Martin, R., Menth, M., Charzinski, J.: Comparison of Border-to-Border Budget Based Network Admission Control and Capacity Overprovisioning. In: Boutaba, R., Almeroth, K.C., Puigjaner, R., Shen, S., Black, J.P. (eds.) NETWORKING 2005. LNCS, vol. 3462, pp. 1056–1068. Springer, Heidelberg (2005)
19. Menth, M., Martin, R., Charzinski, J.: Capacity Overprovisioning for Networks with Resilience Requirements. In: Proceedings of SIGCOMM 2006, September 11-15 (2006)
20. Milbrandt, J., Menth, M., Junker, J.: Experience-Based Admission Control in the Presence of Traffic Changes. Journal of Communications 2(1) (January 2007)
21. Odlyzko, A.: Data Networks are Lightly Utilized, and will Stay that Way. The Review of Network Economics 2 (2003)
22. Olifer, N., Olifer, V.: Computer Networks: Principles, Technologies and Protocols for Network Design. John Wiley & Sons, Chichester (2005)

23. Parekh, A.: Why there is no QoS and what to do about it. In: Jeffay, K., Stoica, I., Wehrle, K. (eds.) IWQoS 2003. LNCS, vol. 2707, Springer, Heidelberg (2003)
24. Pippenger, N.: Information Theory and the Complexity of Switching Networks. In: Proceedings of FOCS, pp. 113–118 (1975)
25. Pippenger, N.: On Rearrangeable and Non-Blocking Switching Networks. Journal of Computer Systems and Sciences 17(2), 145–162 (1978)
26. Pippenger, N.: Telephone Switching Networks. In: Proceedings of Symposia in Applied Mathematics, vol. 26, pp. 101–133 (1982)
27. Pippenger, N., Valiant, L.G.: Shifting Graphs and Their Applications. Journal of the ACM 23(3), 423–432 (1976)
28. Pippenger, N., Yao, A.C.: Rearrange-able networks with limited depth. SIAM Journal Algebraic Discrete Methods 3(4), 411–417 (1982)
29. Prasad, R.S., Winzer, P.J., Borst, S., Thottan, M.K.: Queuing Delays in Randomized Load Balanced Networks. In: Proceedings of IEEE Infocom 2007 (2007)
30. Rui, Z.-S., McKeown, N.: Designing a Predictable Internet Backbone with Valiant Load-Balancing. In: de Meer, H., Bhatti, N. (eds.) IWQoS 2005. LNCS, vol. 3552, pp. 178–192. Springer, Heidelberg (2005)
31. Shepherd, F.B., Winzer, P.J.: Selective randomized load balancing and mesh networks with changing demands. J. Opt. Netw. 5, 320–339 (2006)
32. Telkamp, T.: Traffic Characteristics and Network Planning. In: ISMA 2002 (October 7-11, 2002)
33. Valiant, L.G., Brebner, G.J.: Universal Schemes for Parallel Communication STOC 1981, pp. 263–277 (1981)
34. Valiant, L.G.: A Scheme for Fast Parallel Communication. SIAM J. Comput. 11(2), 350–361 (1982)

Cleaning Random d-Regular Graphs with Brushes Using a Degree-Greedy Algorithm

Margaret-Ellen Messinger[1], Paweł Prałat[1], Richard J. Nowakowski[1,⋆], and Nicholas Wormald[2,⋆⋆]

[1] Department of Mathematics and Statistics, Dalhousie University, Halifax NS, Canada
{messnger,pralat,rjn}@mathstat.dal.ca
[2] Department of Combinatorics and Optimization, Waterloo University, Waterloo ON, Canada
nwormald@uwaterloo.ca

Abstract. In the recently introduced model for *cleaning* a graph with brushes, we use a degree-greedy algorithm to clean a random d-regular graph on n vertices (with dn even). We then use a differential equations method to find the (asymptotic) number of brushes needed to clean a random d-regular graph using this algorithm. As well as the case for general d, interesting results for specific values of d are examined. We also state various open problems.

Keywords: cleaning process, random d-regular graphs, degree–greedy algorithm, differential equations method.

1 Introduction

The cleaning model, introduced in [5,6], considers a network of pipes that must be periodically cleaned of a contaminant that regenerates, for example, algae in water pipes. This is accomplished by having cleaning agents, colloquially, 'brushes', assigned to some vertices. To reduce the recontamination, when a vertex is 'cleaned', a brush must travel down each contaminated edge. Once a brush has traversed an edge, that edge has been *cleaned*. A graph G has been *cleaned* once every edge of G has been cleaned. McKeil [5] considered the model where more than one brush can travel down an edge and brushes can travel down cleaned edges. In [6] and this paper only one brush is allowed to travel along an edge and a brush is not allowed to travel down an edge that has already been cleaned.

Explicitly, every edge and vertex of a graph is initially *dirty* and a fixed number of brushes start on a set of vertices. At each step, a vertex v and all its incident edges which are dirty may be *cleaned* if there are at least as many brushes on v

⋆ Research partially supported by NSERC and MITACS.
⋆⋆ Research partially supported by the Canada Research Chairs Program and NSERC.

J. Janssen and P. Prałat (Eds.): CAAN 2007, LNCS 4852, pp. 13–26, 2007.
© Springer-Verlag Berlin Heidelberg 2007

as there are incident dirty edges. When a vertex is cleaned, every incident dirty edge is traversed (i.e. cleaned) by one and only one brush, moreover, brushes cannot traverse a clean edge. This cleaning process is a combination of the chip-firing game and edge-searching on a simple finite graph. The approach in [6], and taken here, is that a *graph is cleaned when every vertex, and hence every edge, has been cleaned.* This may result in vertices with no dirty edges being cleaned in which case no brushes move but this approach simplified much of the analysis in [6]. See Figure 1 for an example of this cleaning process. The initial configuration has only 2 brushes, both at a. The solid edges are dirty and the dotted edges are clean. The circle indicates which vertex is cleaned next.

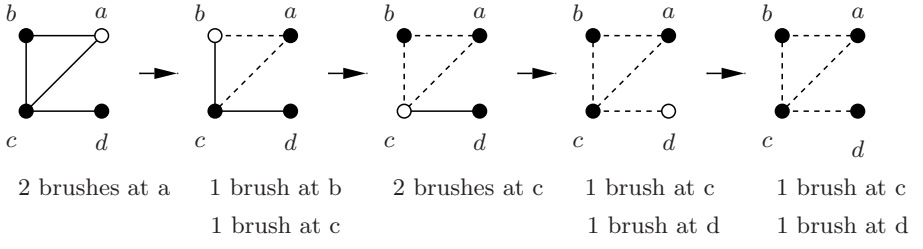


Fig. 1. An example of the cleaning process for graph G

One condition that this model has, like chip-firing but not searching, is that the cleaning process is to be automatic, i.e. a union of 'vertex firing' sequences where each sequence cleans the graph, continuing on for the lifetime of the network. Therefore, the problems to solve are: firstly, a brush configuration and corresponding vertex firing sequence that cleans the graph; and secondly, having the final configuration of brushes be a starting configuration for another vertex firing sequence that also cleans the graph; and so on. In [6], we show that the final configuration of any cleaning sequence is a valid starting configuration of another cleaning sequence.

In this paper, we are interested in the asymptotic number of brushes needed to clean random d-regular (finite, simple) graphs. At one extreme, the graph could consist of disjoint copies of K_{d+1}. From [6], K_{d+1} requires essentially $d^2/4$ brushes so that the whole graph requires approximately $nd/4$. At the lower end, if d is even then a ring of bipartite graphs $K_{d/2,d/2}$ chained together (see Figure 2 for the case $d = 4$) require only $d^2/4$ brushes regardless of the number of vertices (by working around the ring). If d is odd then every vertex has at least one brush in either the original or final configuration (see [6] for more details) so that a graph on n vertices requires at least $n/2$ brushes.

We propose a linear time algorithm to clean d-regular graphs and an a.a.s. upper bound u_d on the number of brushes required by the algorithm. The asymptotically almost sure lower bound l_d follows from the fact that a.a.s. all sets of size $\lfloor n/2 \rfloor$ have at least l_d edges going to its complement.

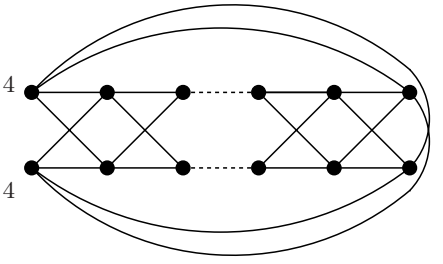


Fig. 2. An example of the cleaning process for a 4-regular graph requiring 8 brushes

In Section 4 we observe that if $d = 2$, then the brush number (asymptotically) is $(1+o(1)) \log n$; for $d = 3$, the brush number is equal to $n/2+2$ a.a.s.; for $d = 4$, $(1 + o(1))n/3$ brushes are enough to clean a graph a.a.s.; and for $d = 5$, roughly $0.644n$. For larger d, numerical evidence suggests that each brush on average cleans between 2 and 2.5 edges. In order to get an asymptotically almost sure upper bound on the brush number we use a degree-greedy algorithm, [9], to clean the graph and then use the differential equation method, studied in [12] to find the asymptotic number of brushes required.

In Section 2 we introduce the formal definitions for the cleaning process and a description of the pairing model of random regular graphs which is used instead of working directly with in the uniform probability space.

2 Definitions

The following cleaning algorithm and terminology was recently introduced in [6].

Formally, at each step t, $\omega_t(v)$ denotes the number of brushes at vertex v ($\omega_t : V \to \mathbb{N} \cup \{0\}$) and D_t denotes the set of dirty vertices. An edge $uv \in E$ is dirty if and only if both u and v are dirty: $\{u, v\} \subseteq D_t$. Finally, let $D_t(v)$ denote the number of dirty edges incident to v at step t:

$$D_t(v) = \begin{cases} |N(v) \cap D_t| & \text{if } \ v \in D_t \\ 0 & \text{otherwise.} \end{cases}$$

Definition 1. *The* ***cleaning process*** $\mathfrak{P}(G, \omega_0) = \{(\omega_t, D_t)\}_{t=0}^{T}$ *of an undirected graph* $G = (V, E)$ *with an* ***initial configuration of brushes*** ω_0 *is as follows:*

(0) *Initially, all vertices are dirty:* $D_0 = V$*; Set* $t := 0$

(1) *Let* α_{t+1} *be any vertex in* D_t *such that* $\omega_t(\alpha_{t+1}) \geq D_t(\alpha_{t+1})$*. If no such vertex exists, then stop the process* ($T = t$)*, return the* ***cleaning sequence*** $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_T)$*, the* ***final set of dirty vertices*** D_T*, and the* ***final configuration of brushes*** ω_T

(2) *Clean* α_{t+1} *and all dirty incident edges by moving a brush from* α_{t+1} *to each dirty neighbour. More precisely,* $D_{t+1} = D_t \setminus \{\alpha_{t+1}\}$, $\omega_{t+1}(\alpha_{t+1}) = \omega_t(\alpha_{t+1}) - D_t(\alpha_{t+1})$, *and for every* $v \in N(\alpha_{t+1}) \cap D_t$, $\omega_{t+1}(v) = \omega_t(v) + 1$ *(the other values of* ω_{t+1} *remain the same as in* ω_t*)*

(3) $t := t + 1$ *and go back to* **(1)**

Note that for a graph G and initial configuration ω_0, the cleaning process can return different cleaning sequences and final configurations of brushes; consider, for example, an isolated edge uv and $\omega_0(u) = \omega_0(v) = 1$. It has been shown (see Theorem 2.1 in [6]), however, that the final set of dirty vertices is determined by G and ω_0. Thus, the following definition is natural.

Definition 2. *A graph* $G = (V, E)$ ***can be cleaned*** *by the initial configuration of brushes* ω_0 *if the cleaning process* $\mathfrak{P}(G, \omega_0)$ *returns an empty final set of dirty vertices* ($D_T = \emptyset$).

Let the brush number, $b(G)$, *be the minimum number of brushes needed to clean* G, *that is,*

$$b(G) = \min_{\omega_0 : V \to \mathbb{N} \cup \{0\}} \Big\{ \sum_{v \in V} \omega_0(v) : G \text{ can be cleaned by } \omega_0 \Big\}.$$

Similarly, $b_\alpha(G)$ *is defined as the minimum number of brushes needed to clean* G *using the cleaning sequence* α.

It is clear that for every cleaning sequence α, $b_\alpha(G) \geq b(G)$ and $b(G) = \min_\alpha b_\alpha(G)$. (The last relation can be used as an alternative definition of $b(G)$.) In general, it is difficult to find $b(G)$, but $b_\alpha(G)$ can be easily computed. For this, it seems better not to choose the function ω_0 in advance, but to run the cleaning process in some order, and compute the initial number of brushes needed to clean a vertex. We can adjust ω_0 along the way

$$\omega_0(\alpha_{t+1}) = \max\{2D_t(\alpha_{t+1}) - \deg(\alpha_{t+1}), 0\}, \qquad \text{for } t = 0, 1, \ldots, |V| - 1, \quad (1)$$

since that is how much brushes we have to add over and above what we get for free.

Our results refer to the probability space of random d-regular graphs with uniform probability distribution. This space is denoted $\mathcal{G}_{n,d}$, and asymptotics (such as "asymptotically almost surely", which we abbreviate to a.a.s.) are for $n \to \infty$ with $d \geq 2$ fixed, and n even if d is odd.

Instead of working directly in the uniform probability space of random regular graphs on n vertices $\mathcal{G}_{n,d}$, we use the *pairing model* of random regular graphs, first introduced by Bollobás [1], which is described next. Suppose that dn is even, as in the case of random regular graphs, and consider dn points partitioned into n labeled buckets $v_1, v_2, \ldots, v_n$ of d points each. A *pairing* of these points is a perfect matching into $dn/2$ pairs. Given a pairing P, we may construct a multigraph $G(P)$, with loops allowed, as follows: the vertices are the buckets $v_1, v_2, \ldots, v_n$, and a pair $\{x, y\}$ in P corresponds to an edge $v_i v_j$ in $G(P)$ if x

and y are contained in the buckets v_i and v_j, respectively. It is an easy fact that the probability of a random pairing corresponding to a given simple graph G is independent of the graph, hence the restriction of the probability space of random pairings to simple graphs is precisely $\mathcal{G}_{n,d}$. Moreover, it is well known that a random pairing generates a simple graph with probability asymptotic to $e^{(1-d^2)/4}$ depending on d, so that any event holding a.a.s. over a probability space of random pairings also holds a.a.s. over the corresponding space $\mathcal{G}_{n,d}$. For this reason, asymptotic results over random pairings suffice for our purposes. The advantage of using this model is that the pairs may be chosen sequentially so that the next pair is chosen uniformly at random over the remaining (unchosen) points. For more information on this model, see [10].

3 Some Lower Bounds

When a graph G is cleaned using the cleaning process described in Definition 1, each edge of G is traversed exactly once and by exactly one brush.

Definition 3. *Given some initial configuration ω_0 of brushes, suppose $G = (V, E)$ admits a cleaning sequence $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_T)$ which cleans G. As each edge in G is traversed exactly once and by exactly one brush, an orientation of the edges of G is permitted such that for every $\alpha_i\alpha_j \in E(G)$, $\alpha_i \to \alpha_j$ if and only if $i < j$.*

*The **brush path** of a brush b is the oriented path formed by the set of edges cleaned by b (note that a vertex may not be repeated in a brush path). Then G can be decomposed into $b_\alpha(G)$ oriented brush paths (note that no brush can stay at its initial vertex in the minimal brush configuration).*

The minimum number of paths into which graph G can be decomposed yields a lower bound for $b(G)$; only a lower bound because some path decompositions would not be valid in the cleaning process. For example, K_4 can be decomposed into two edge-disjoint paths, but $b(K_4) = 4$.

Following Definitions 1 and 3, every vertex of odd degree in a graph G will be the endpoint of (at least) one brush path. This leads to a natural lower bound for $b(G)$ since any graph with d_o odd vertices, can be decomposed into a minimum of $d_o/2$ paths (see [6] for more details).

Theorem 1. *Given initial configuration ω_0, suppose G can be cleaned yielding final configuration ω_T. Then for every vertex v in G with odd degree, either $\omega_0(v) > 0$ or $\omega_T(v) > 0$. In particular, $b(G) \geq d_o(G)/2$ where $d_o(G)$ denotes a number of vertices of odd degree.*

The result can be improved a little if there is a lower bound on the vertex degrees (see Section 4.3 for details).

Another general lower bound for d-regular graphs can be obtained as follows. By [6, Theorem 3.2],

$$b(G) \geq \max_j \min_{S \subseteq V, |S|=j} \{jd - 2|E(G[S])|.\}$$

(The proof is simply to observe that the minimum is a lower bound on the number of edges going from the first j vertices cleaned to elsewhere in the graph.) So, suppose that x and y are such that the expected number of sets S of $xn + o(n)$ vertices in $G \in \mathcal{G}_{n,d}$ with $yn + o(n)$ edges to the complementary $V(G) \setminus S$ is $o(1)$. Then this theorem, together with the first moment principle, gives that the brush number is a.a.s. at least $yn + o(n)$. Some standard calculations using the pairing model then give us the following lower bounds a.a.s.: $0.220n$ for $d = 4$, $0.365n$ for $d = 5$ (although this can easily be improved to the lower bound of 0.5n), $0.52n$ for $d = 6$, and $0.687n$ for $d = 7$. We omit further details from this paper.

4 Cleaning Random d-Regular Graphs

The differential equations method (described in [12]) is used here to find an upper bound on the number of brushes needed to the clean the graph using a degree-greedy algorithm. We consider $d = 2$ first, then state some general results, and apply them to the special cases of $3 \leq d \leq 5$ before discussing higher values of d.

4.1 2-Regular Graphs

Let $Y = Y_n$ be the total number of cycles in a random 2-regular graph on n vertices. Since exactly two brushes are needed to clean one cycle, we need $2Y_n$ brushes in order to clean a 2-regular graph.

We know that the random 2-regular graph is a.a.s. disconnected; by simple calculations we can show that the probability of having a Hamiltonian cycle is asymptotic to $\frac{1}{2}e^{3/4}\sqrt{\pi}n^{-1/2}$ (see, for example, [10]).

We also know that the total number of cycles Y_n is sharply concentrated near $(1/2)\log n$. It is not difficult to see this by generating the random graph sequentially using the pairing model. The probability of forming a cycle in step i is exactly $1/(2n-2i+1)$, so the expected number of cycles is $(1/2)\log n + O(1)$. The variance can be calculated in a similar way. So we get that a.a.s. the brush number for a random 2-regular graph is $(1 + o(1))\log n$.

4.2 d-Regular Graphs ($d \geq 3$) — The General Setting

In this subsection, we assume $d \geq 3$ is fixed with dn even. In order to get an asymptotically almost sure upper bound on the brush number, we study an algorithm that cleans random vertices of minimum degree. This algorithm is called *degree-greedy* because the vertex being cleaned is chosen from those with the lowest degree.

We start with a random d-regular graph $G = (V, E)$ on n vertices. Initially, all vertices are dirty: $D_0 = V$. In every step t of the cleaning process, we clean a random vertex α_t, chosen uniformly at random from those vertices with the lowest degree ($D_t = D_{t-1} \setminus \{\alpha_t\}$) in the induced subgraph $G[D_{t-1}]$. In the first step, d brushes are needed to clean random vertex α_1 (we say that this is "phase

zero"). Note that this is a.a.s. the only vertex whose degree in D_t is d at the time of cleaning. Indeed, if α_t ($t \geq 2$) has degree d in $G[D_{t-1}]$, then $G[D_{t-1}]$ consists of a connected component(s) of G and thus G is disconnected. It was proven independently in [2,11] that G is disconnected with probability $o(1)$ and later extended to d growing with n in [4]. The induced subgraph $G[D_1]$ now has d vertices of degree $d-1$ and $n-d-1$ vertices of degree d.

In the second step, $d-2$ extra brushes are needed to clean a random vertex α_2 of degree $d-1$. Typically, in the third step, a vertex of degree $d-1$ is cleaned and in each subsequent step, a vertex of degree $d-1$ is cleaned until some vertex of degree $d-2$ is produced in the subgraph induced by the set of dirty vertices. After cleaning the first vertex of degree $d-2$, we typically return to cleaning vertices of degree $d-1$, but after a some more steps of this type we may clean another vertex of degree $d-2$. When vertices of degree $d-1$ become plentiful, vertices of lower degree are more commonly created and these hiccups occur more often. When vertices of degree $d-2$ take over the role of vertices of degree $d-1$, we say (informally!) that the first phase finishes and we begin the second phase. In general, in the kth phase a mixture of vertices of degree $d-k$ and $d-k-1$ are cleaned.

It is usually difficult to study the behaviour of a greedy algorithm at the end of the process. Fortunately, in this case we need to study the first $\lfloor (d-1)/2 \rfloor$ phases since the rest of vertices are cleaned 'for free'. The details have been omitted, but can be found in [9].

For $0 \leq i \leq d$, let $Y_i = Y_i(t)$ denote the number of vertices of degree i in $G[D_t]$. (Note that $Y_0(t) = n - t - \sum_{i=1}^{d} Y_i(t)$ so $Y_0(t)$ does not need to be calculated, but it is useful in the discussion.) Let $S(t) = \sum_{l=1}^{d} lY_l(t)$ and for any statement A, let δ_A denote the Kronecker delta function

$$\delta_A = \begin{cases} 1 & \text{if } A \text{ is true,} \\ 0 & \text{otherwise.} \end{cases}$$

It is not difficult to see that

$$\begin{aligned} \mathbb{E}(Y_i(t) - Y_i(t-1) \mid G[D_{t-1}] \ \wedge \ \deg_{G[D_{t-1}]}(\alpha_t) = r) \\ &= f_{i,r}((t-1)/n, Y_1(t-1)/n, Y_2(t-1)/n, \ldots, Y_d(t-1)/n) \\ &= -\delta_{i=r} - r\frac{iY_i(t-1)}{S(t-1)} + r\frac{(i+1)Y_{i+1}(t-1)}{S(t-1)}\delta_{i+1\leq d} \end{aligned} \tag{2}$$

for $i, r \in [d]$ such that $Y_r(t) > 0$. Indeed, α_t has degree r, hence the term $-\delta_{i=r}$. When a pair of points in the pairing model is exposed, the probability that the other point is in a bucket of degree i (that is, the bucket contains i unchosen points) is asymptotic to $iY_i(t-1)/S(t-1)$. Thus $riY_i(t-1)/S(t-1)$ stands for the expected number of the r buckets found adjacent to α_t which have degree i. This contributes negatively to the expected change in Y_i, whilst buckets of degree $i+1$ which are reached contribute positively (of course, only if this type of vertices (buckets) exist in a graph; thus $\delta_{i+1\leq d}$). This explains (2).

Suppose that at some step t of the phase k, cleaning a vertex of degree $d-k$ creates, in expectation, β_k vertices of degree $d-k-1$ and cleaning a vertex of degree $d-k-1$ decreases, in expectation, the number of vertices of degree $d-k-1$ by τ_k. After cleaning a vertex of degree $d-k$, we expect to then clean (on average) β_k/τ_k vertices of degree $d-k-1$. Thus, in phase k, the proportion of steps which clean vertices of degree $d-k$ is $1/(1+\beta_k/\tau_k) = \tau_k/(\beta_k+\tau_k)$. If τ_k falls below zero, vertices of degree $d-k-1$ begin to build up and do not decrease under repeated cleaning vertices of this type and we move to the next phase.

From (2) it follows that

$$\beta_k = \beta_k(x, y_1, y_2, \ldots, y_d) \;=\; f_{d-k-1,d-k}(x, y_1, y_2, \ldots, y_d) \;=\; f_{d-k-1,d-k}(x, \mathbf{y}),$$

$$\begin{aligned}\tau_k = \tau_k(x, y_1, y_2, \ldots, y_d) \;&=\; -\, f_{d-k-1,d-k-1}(x, y_1, y_2, \ldots, y_d)\\ &=\; -\, f_{d-k-1,d-k-1}(x, \mathbf{y}),\end{aligned}$$

where $x = t/n$ and $y_i(x) = Y_i(t)/n$ for $i \in [d]$. This *suggests* (see [12] for more information on the differential equations method) the following system of differential equations

$$\frac{dy_i}{dx} = F(x, \mathbf{y}, i, k)$$

where

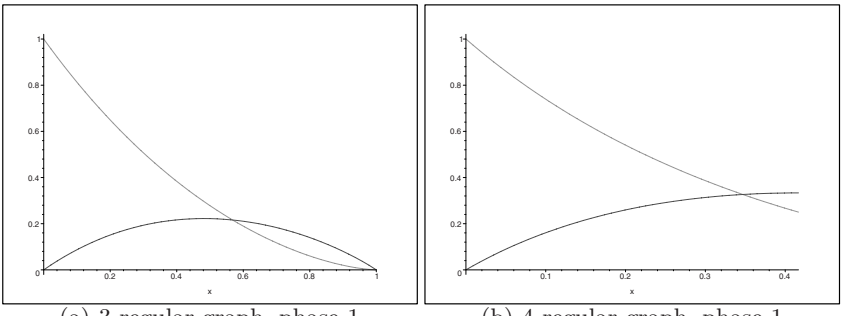
$$F(x, \mathbf{y}, i, k) = \begin{cases} \frac{\tau_k}{\beta_k+\tau_k} f_{i,d-k}(x, \mathbf{y}) + \frac{\beta_k}{\beta_k+\tau_k} f_{i,d-k-1}(x, \mathbf{y}) & \text{for } k \le d-2,\\ f_{i,1}(x, \mathbf{y}) & \text{for } k = d-1. \end{cases}$$

At this point we may formally define the interval $[x_{k-1}, x_k]$ to be phase k, where the termination point x_k is defined as the infimum of those $x > x_k$ for which at least one of the following holds: $\tau_k \le 0$ and $k < d-1$; $\tau_k + \beta_k = 0$ and $k < d-1$; $y_{d-k} \le 0$. Using final values $y_i(x_k)$ in phase k as an initial values for phase $k+1$ we can repeat the argument inductively moving from phase to phase starting from phase 1 with obvious initial conditions $y_d(0) = 1$ and $y_i(0) = 0$ for $0 \le i \le d-1$.

The general result [9, Theorem 1] studies a deprioritized version of degree-greedy algorithms, which means that the vertices are chosen to process in a slightly different way, not always the minimum degree, but usually a random mixture of two degrees. Once a vertex is chosen, it is treated the same as in the degree-greedy algorithm. The variables Y are defined in an analogous manner. The hypotheses of the theorem are straightforward to verify. The conclusion is that, for a certain algorithm using a deprioritized 'mixture' of the steps of the degree-greedy algorithm, with variables Y_i defined as above, we have that a.a.s.

$$Y_i(t) = n y_i(t/n) + o(n)$$

for $1 \le i \le d$ for phases $k = 1, 2, \ldots, m$, where m denotes the smallest k for which either $k = d-1$, or any of the termination conditions for phase k hold



(a) 3-regular graph, phase 1 (b) 4-regular graph, phase 1

Fig. 3. Solution to the differential equations

at x_k apart from $x_k = \inf\{x > x_{k-1} : \tau_k \leq 0\}$. We omit all details pointing the reader to [9] and the general survey [12] about the differential equations method which is a main tool in proving [9, Theorem 1]. In addition, the theorem gives information on an auxiliary variable such as, of importance to our present application, the number of brushes used. Instead of quoting this precisely, we use it merely as justification for being able to use the above equations as if they applied to the greedy algorithm. (This is no doubt the case, but it is not actually proved in [9].) The solutions to the relevant differential equations for $d = 3$ and 4 are shown in Figure 3.

In the kth phase a mixture of vertices of degree $d-k$ and $d-k-1$ are cleaned. Since $\max\{2l - d, 0\}$ brushes are needed to clean vertex of degree l (see (1)), we need

$$u_d^k = (1 + o(1))n\left(\max\{d - 2k, 0\} \int_{x_{k-1}}^{x_k} \frac{\tau_k}{\tau_k + \beta_k} dx + \max\{d - 2k - 2, 0\} \int_{x_{k-1}}^{x_k} \frac{\beta_k}{\tau_k + \beta_k} dx \right)$$

brushes in phase k. Thus, the total number of brushes needed to clean a graph using the degree-greedy algorithm is equal to

$$u_d = \sum_{k=1}^{\lfloor (d-1)/2 \rfloor} u_d^k = (1 + o(1))n\left(\sum_{k=1}^{\lfloor (d-1)/2 \rfloor} \left((d - 2k - 2)(x_k - x_{k-1}) + 2 \int_{x_{k-1}}^{x_k} \frac{\tau_k}{\tau_k + \beta_k} dx \right) + \delta_{d \text{ is odd}} \int_{x_{k-1}}^{x_k} \frac{\beta_k}{\tau_k + \beta_k} dx \right).$$

4.3 3-Regular Graphs

Let $G = (V, E)$ be any 3-regular graph on n vertices. The first vertex cleaned must start three brush paths, the last one terminates three brush paths, and

all other vertices must start or finish at least one brush path, so the number of brush paths is at least $n/2 + 2$.

The result mentioned above can be shown to result in an upper bound of $n/2 + o(n)$ for the brush number of a random 3-regular (i.e. cubic) graph. We do not provide details because of the following stronger result. It is known [8] that a random 3-regular graph a.a.s. has a Hamilton cycle. The edges not in a Hamilton cycle must form a perfect matching. Such a graph can be cleaned by starting with three brushes at one vertex, and moving along the Hamilton cycle with one brush, introducing one new brush for each edge of the perfect matching. Hence the brush number of a random 3-regular graph with n vertices is a.a.s. $n/2 + 2$.

4.4 4-Regular Graphs

For 4-regular graphs, we are interested in phase 1 only: we need two brushes to clean vertices of degree 3, but vertices of degree 2 are cleaned 'for free'. Note that $y_1(x) = y_2(x) = 0$. We have the following system of differential equations

$$\frac{dy_4}{dx} = \frac{-6y_4(x)}{3y_3(x) + 2y_4(x)}$$
$$\frac{dy_3}{dx} = \frac{-3y_3(x) + 4y_4(x)}{3y_3(x) + 2y_4(x)}$$

with the initial conditions $y_4(0) = 1$ and $y_3(0) = 0$. The particular solution (see Figure 3 (b)) to these differential equations is

$$y_4(x) = 5 - 4\sqrt{1+3x} + 3x$$
$$y_3(x) = \frac{4(-3 + 3\sqrt{1+3x} - 5x + x\sqrt{1+3x})}{2 - \sqrt{1+3x}},$$

so $\beta_1 = -3 + 3\sqrt{1+3x}$ and $\tau_1 = 3 - 2\sqrt{1+3x}$. Thus phase 1 finishes at time $t_1 = 5n/12$ ($x_1 = 5/12$ is a root of the equation $\tau_1(x) = 0$) and the number of vertices of degree 3 cleaned during this phase is asymptotic to

$$n \int_0^{5/12} \frac{\tau_1}{\tau_1 + \beta_1} dx = n/6\,.$$

Since we need 2 brushes to clean one such vertex we get an asymptotically almost sure upper bound of $u_4 = (1 + o(1))n/3$.

On the other hand, it is true that a.a.s. a random 4-regular graph can be decomposed into two edge-disjoint Hamilton cycles [3], and hence four paths.

Note that the following two problems can be asked in general for any $d \geq 3$.

Problem 1. Is it true that for the random case it is best to clean lowest degree vertices?

In other words, if one is going to choose a random vertex of given degree then one might as well choose a random vertex of minimum degree.

If Problem 1 is proven to be true, then the following problem should be considered. To get the brush number one might (in fact, probably should) choose non-random vertices during the cleaning process. But it might be true that a.a.s. one cannot save more than $o(n)$ brushes compared to the greedy algorithm under consideration.

Problem 2. Is it true that a.a.s. the brush number for a random d-regular graph is $u_d(1 - o(1))$?

4.5 5-Regular Graphs

In order to study the brush number for 5-regular graphs yielded by the degree-greedy algorithm, we cannot consider phase 1 only as before; we need 3 brushes to clean vertices of degree 4 but also 1 brush to clean vertices of degree 3. Thus two phases must be considered.

In phase 1, $y_1(x) = y_2(x) = y_3(x) = 0$ and we have the following system of differential equations

$$\begin{aligned}\frac{dy_5}{dx} &= \frac{-20y_5(x)}{8y_4(x) + 5y_5(x)}\\ \frac{dy_4}{dx} &= \frac{-8y_4(x) + 15y_5(x)}{8y_4(x) + 5y_5(x)}\end{aligned}$$

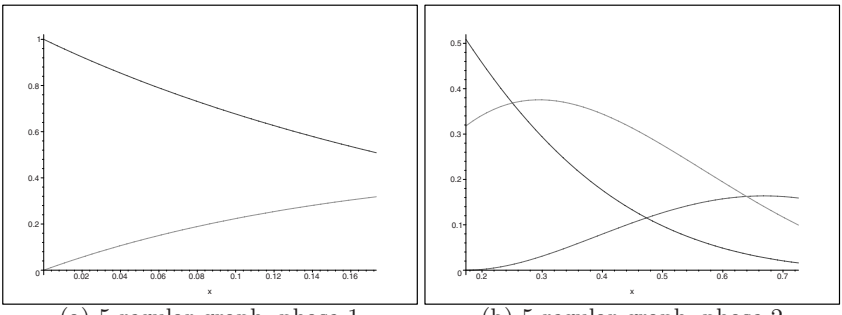
with the initial conditions $y_5(0) = 1$ and $y_4(0) = 0$. The numerical solution (see Figure 4 (a)) *suggests* that the phase finishes at time $t_1 = 0.1733n$. The number of brushes needed in this phase is asymptotic to (the numerical solution)

$$\begin{aligned}u_5^1 &= (1 + o(1))\left(3n\int_0^{t_1/n}\frac{\tau_1}{\tau_1 + \beta_1}dx + n\int_0^{t_1/n}\frac{\beta_1}{\tau_1 + \beta_1}dx\right)\\ &= (1 + o(1))\left(t_1 + 2n\int_0^{t_1/n}\frac{\tau_1}{\tau_1 + \beta_1}dx\right) \approx 0.3180n\,.\end{aligned}$$

In the phase 2, $z_1(x) = z_2(x) = 0$ and we have another system of differential equations

$$\begin{aligned}\frac{dz_5}{dx} &= \frac{-15z_5(x)}{6z_3(x) + 4z_4(x) + 5z_5(x)}\\ \frac{dz_4}{dx} &= \frac{-3(4z_4 - 5z_5(x))}{6z_3(x) + 4z_4(x) + 5z_5(x)}\\ \frac{dz_3}{dx} &= \frac{-6z_3(x) + 8z_4(x) - 5z_5(x)}{6z_3(x) + 4z_4(x) + 5z_5(x)}\end{aligned}$$

with the initial conditions $z_5(t_1/n) = y_5(t_1/n) = 0.5088$, $z_4(t_1/n) = y_4(t_1/n) = 0.3180$ and $z_3(t_1/n) = 0$. The numerical solution (see Figure 4 (b)) *suggests* that



(a) 5-regular graph, phase 1 (b) 5-regular graph, phase 2

Fig. 4. Solution to the differential equations

the phase finishes (approximately) at time $t_2 = 0.7257n$. The number of brushes needed in this phase is asymptotic to (the numerical solution)

$$u_5^2 = (1 + o(1))n \int_{t_1/n}^{t_2/n} \frac{\tau_2}{\tau_2 + \beta_2} dx \approx 0.3259n\,.$$

Finally, we get an asymptotically almost sure upper bound of $u_5 = u_5^1 + u_5^2 \approx 0.6439n$.

4.6 *d*-Regular Graphs of Higher Order

Note that the lower bound for $d = 4$ (see Section 3) will be considerably lower than the lower bound of $n/2 + 2$ for $d = 3$, whereas the upper bound we have been discussing is the same degree-greedy algorithm in all cases. However, the upper bound is also sensitive to the parity of d. For the 4-regular case, vertices of degree 2 are processed 'for free' and so one only really worries about degree 3 vertices and there are fewer of those processed than degree 2 vertices when

Table 1. Upper bounds on the brush number for some d values

d	$\lim_{n\to\infty} u_d/n$	d	$\lim_{n\to\infty} u_d/n$	d	$\lim_{n\to\infty} u_d/n$	d	$\lim_{n\to\infty} u_d/n$
3	0.5	13	2.078	23	4.159	99	21.422
4	0.334	14	2.248	24	4.358	100	21.653
5	0.644	15	2.482	25	4.589	149	33.169
6	0.684	16	2.661	26	4.791	150	33.404
7	0.949	17	2.893	27	5.022	199	45.036
8	1.057	18	3.079	28	5.227	200	45.273
9	1.305	19	3.311	29	5.457	249	56.979
10	1.444	20	3.502	30	5.664	250	57.217
11	1.684	21	3.733	31	5.895	299	68.975
12	1.842	22	3.928	32	6.104	300	69.215

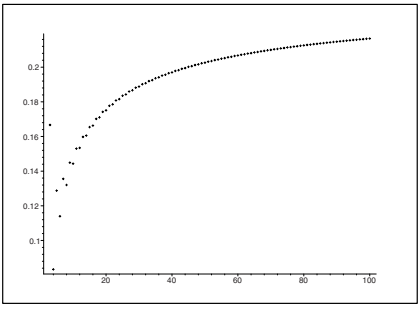


Fig. 5. A graph of $\lim_{n\to\infty} u_d/dn$ versus d (from 3 to 100)

$d = 3$. But it seems that the parity of d does not affect the value of u_d for d big enough (see Figure 5 and Table 1).

Problem 3. Does $\lim_{d\to\infty} \lim_{n\to\infty} u_d/dn$ exist?

In Figure 5, the values of $\lim_{n\to\infty} u_d/dn$ have been presented for all d-values up to 100, although we have only listed the first 30 and a few more values for higher d in Table 1. The computations presented in the paper were performed by using Maple$^{\text{TM}}$ [7]. The worksheets can be found at the following address: `http://www.mathstat.dal.ca/~pralat/`.

Finally, the most important open question is clearly the following:

Problem 4. Let $G \in G(n, d)$. Is there a constant c such that the brush number is asymptotically cdn?

4.7 Other Models

In this section, we present more open problems.

Problem 5. What is the brush number for binomial random graphs $G(n, p)$? What is a lower/upper bound? How about other random graph models, for example models that give power law degree distribution or d-regular graphs generated by the d-process?

Another version of the cleaning process was introduced in [5]. In this version, when a vertex is cleaned multiple brushes are allowed to traverse each dirty edge. Thus, the brush number $B(G)$ of this generalized version is at most the classic one $b(G)$. Using the degree-greedy algorithm to clean a random d-regular graph for d even, no brush 'gets stuck' in the first $\lfloor (d-1)/2 \rfloor$ phases, there is no point to introduce more brushes in the initial configuration, and vertices in the last phases are cleaned 'for free'. So the upper bound obtained is the same as before. For d odd, it is clear that one can save some brushes at phase $(d-1)/2$ but the following is still open.

Problem 6.

- Is it true that for $G \in \mathcal{G}_{n,d}$, d even, $b(G) - B(G) = o(n)$ a.a.s.?
- Is it true that for $G \in \mathcal{G}_{n,d}$, d odd, $b(G) - B(G) = \Theta(n)$ a.a.s.? How far apart are they?

References

1. Bollobás, B.: A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. European Journal of Combinatorics 1, 311–316 (1980)
2. Bollobás, B.: Random graphs, Combinatorics. In: Temperley, H.N.V. (ed.) London Mathematical Society Lecture Note Series, vol. 52, pp. 80–102. Cambridge University Press, Cambridge (1981)
3. Kim, J.H., Wormald, N.C.: Random matchings which induce Hamilton cycles and hamiltonian decompositions of random regular graphs. Journal of Combinatorial Theory, Series B 81, 20–44 (2001)
4. Łuczak, T.: Sparse random graphs with a given degree sequence. In: Frieze, A., Łuczak, T. (eds.) Random Graphs, vol. 2, pp. 165–182. Wiley, New York (1992)
5. McKeil, S.: Chip Firing Cleaning Processes. MSc Thesis, Dalhousie University (2007)
6. Messinger, M.E., Nowakowski, R.J., Prałat, P.: Cleaning a Network with Brushes. Theoretical Computer Science (accepted)
7. Monagan, M.B., Geddes, K.O., Heal, K.M., Labahn, G., Vorkoetter, S.M., McCarron, J., DeMarco, P.: Maple 10 Programming Guide. Maplesoft, Waterloo, Canada (2005)
8. Robinson, R.W., Wormald, N.C.: Almost all cubic graphs are hamiltonian. Random Structures and Algorithms 3, 117–125 (1992)
9. Wormald, N.C.: Analysis of greedy algorithms on graphs with bounded degrees. EuroComb 2001. Discrete Mathematics 273, 235–260 (2003)
10. Wormald, N.C.: Models of random regular graphs. In: Lamb, J.D., Preece, D.A. (eds.) Surveys in Combinatorics. London Mathematical Society Lecture Note Series, vol. 276, pp. 239–298. Cambridge University Press, Cambridge (1999)
11. Wormald, N.C.: The asymptotic connectivity of labelled regular graphs. Journal of Combinatorial Theory, Series B 31, 156–167 (1981)
12. Wormald, N.C.: The differential equation method for random graph processes and greedy algorithms. In: Karoński, M., Prömel, H.J. (eds.) Lectures on Approximation and Randomized Algorithms. PWN, Warsaw, pp. 73–155 (1999)

Nonadaptive Selfish Routing with Online Demands

Tobias Harks[1] and László A. Végh[2,⋆]

[1] Institute of Mathematics, Technical University Berlin, 10623 Berlin, Germany
harks@math.tu-berlin.de
[2] Department of Operations Research, Eötvös University, Budapest, Hungary, H-1117
veghal@cs.elte.hu

Abstract. We study the efficiency of selfish routing problems in which traffic demands are revealed *online*. We go beyond the common Nash equilibrium concept in which possibly all players reroute their flow and form a new equilibrium upon arrival of a new demand.

In our model, demands arrive in n sequential games. In each game, the new demands form a Nash equilibrium and their routings remain unchanged afterwards. We study the problem both with nonatomic and atomic player types and with continuous and nondecreasing latency functions on the edges. For polynomial latency functions, we give constant upper and lower bounds on the competitive ratio of the resulting online routing in terms of the maximum degree, the number of games and in the atomic setting the number of players. In particular, for nonatomic players and affine latency functions we show that the competitive ratio is at most $\frac{4n}{n+2}$. Finally, we present improved upper bounds for the special case of two nodes connected by parallel arcs.

1 Introduction

Recent contributions in the field of algorithmic game theory provided much insight into the structure and efficiency of Nash equilibria in networks that lack a central coordination. Among others, a prominent result in this field states that the *price of anarchy* for a nonatomic selfish routing game, is bounded by a small constant depending on the class of feasible latency functions, see Roughgarden and Tardos [30], Roughgarden [29], and Correa Schulz, and Stier-Moses [11]. It is well known that this kind of games applies to the source routing concept in telecommunication networks, see Qiu, Yang, Zhang, and Shenker [25] and Friedman [19] for an engineering perspective and Roughgarden [28] and Altman, Basar, Jimenez, and Shimkin [1] for a theoretical perspective on this topic. In the source routing model, sources are responsible for selecting paths to route data to the corresponding sink.

⋆ Supported by the Hungarian National Foundation for Scientific Research, OTKA K60802 and NK 67867 and by European MCRTN ADONET, Grant Number 504438. Work was done while visiting the Konrad Zuse Institute in Berlin.

J. Janssen and P. Prałat (Eds.): CAAN 2007, LNCS 4852, pp. 27–45, 2007.
© Springer-Verlag Berlin Heidelberg 2007

The main focus of the research done so far regarding the source routing concept is to quantify the efficiency loss of a Nash equilibrium compared to the system optimum. Here, one assumption is crucial: if the traffic matrix changes, all sources may possibly change their routes and converge to a new equilibrium, see Even-Dar and Mansour [16] for a further discussion about the convergence behavior. This assumption, however, has some important implications: Each source would have to *continuously* maintain the current state of all available routes, which in turn introduces additional traffic overhead by signaling these needed informations. Furthermore, frequent rerouting attempts during data transmission may not only produce transient load oscillations as observed by Fischer and Vöcking [18], but may also interfere with the widely used congestion control protocol TCP that determines the data rate, as reported by La, Walrand, and Anantharam in [24]. For these reasons, rerouting attempts in reaction to traffic changes in the network are not necessarily beneficial and efficient.

In this paper, we study a different model in which demands of players are released in n sequential games in an online fashion. In each game, the new demands form a Nash equilibrium, and their routing remains unchanged afterwards, that is, the routing becomes *nonadaptive*.

We can interpret this model as follows. Let us introduce a cost for each player quantifying the *cost of rerouting* after some initial time frame. Within the standard equilibrium concept, rerouting comes at no cost. On the other hand, if this rerouting cost is sufficiently large for each player, then, fixing the initial equilibrium routing is the best response strategy.

If rerouting is not allowed in general, then, the problem of finding efficient routings becomes an *online* optimization problem. In this regard, nonadaptive selfish routing constitutes an online algorithm, where the goal is to minimize average congestion cost for all commodities. We present two *distributed* online algorithms, called NSEQNASH and ASEQNASH for this setting. Upon release of a set of commodities (network game), the online algorithm NSEQNASH routes the commodity such that the flow is at Nash equilibrium provided *nonatomic* agents are carrying the flow. The *atomic* splittable variant is given by ASEQNASH.

1.1 Related Work

The fact that the cost of a Nash equilibrium may strictly exceed that of a system optimum is well known in the transportation literature, see Braess [6] and Dubey [15]. A first successful attempt to exactly quantify this so called "price of anarchy" is given by Papadimitriou and Koutsoupias [23] in the context of a load balancing game in communication networks. Roughgarden and Tardos [30] studied the price of anarchy in nonatomic selfish routing games. In nonatomic games, a large number of players is assumed, each consuming an infinitesimal part of the resources. In particular, they proved for affine latency functions a bound of $\frac{4}{3}$ on the price of anarchy. A series of several other follow-up papers analyzed the price of anarchy for more general cost functions and model features; see for example Czumaj and Vöcking [13], Correa Schulz, and Stier-Moses [11], and Roughgarden [28].

For atomic routing games, that is, some players may control a significant part of the entire demand, Roughgarden and Tardos [30] examined the price of anarchy for unsplittable flow. Awerbuch, Azar, and Epstein [2] and Christodoulou and Koutsoupias [9] studied the price of anarchy for linear atomic congestion games. Cominetti, Correa, and Stier-Moses [10] presented new bounds on the price of anarchy for splittable atomic routing games that revised previous work of Roughgarden [29] and Correa, Schulz, and Stier-Moses [12]. Hayrapetyan, Tardos, and Wexler [22] improved these bounds for special network topologies.

In the online routing field, several papers considered online load balancing in the context of machine scheduling. Awerbuch et al. [3] considered a greedy online load balancing strategy, where the goal is to minimize the L_2 norm of the aggregated server loads. Similar to this paper, Suri et al. [31] and Caragiannis et al. [7] studied Nash solutions for every released job and showed that the resulting online algorithm outperforms the greedy strategy of [3]. These results, however, are restricted to m parallel arcs and all jobs have to be assigned to exactly one machine. In the paper by Awerbuch, Azar, and Plotkin [4], online routing algorithms are presented to maximize throughput under the assumption that routings are irrevocable. They presented online algorithms whose competitive bounds depend on the number of nodes in the network.

Our work is motivated by the paper by Harks, Heinz, and Pfetsch [21], where online multicommodity routing problems are considered. They considered affine latency functions and presented a greedy online algorithm for a different convex cost function that is $\frac{4K^2}{(1+K)^2}$ competitive, where K is the number of commodities. In their framework, only single demands are released consecutively.

1.2 Our Results and Techniques

We introduce the framework *Online Network Games* (OnlineNG) to analyze *nonadaptive selfish routing* under the assumption that demands (network games) are released *online*. For the online algorithm NSeqnash that is characterized by selfish routing of *nonatomic* players for a sequence of network games, we obtain the following results. The online algorithm NSeqnash that produces a flow that is at Nash equilibrium for every game is $\frac{4n}{2+n}$ -competitive for affine latency functions, where n is the number of games within a given sequence. This result contains the bound on the price of anarchy of $\frac{4}{3}$ for affine latency functions of Roughgarden and Tardos [30] as a special case of our result, where $n = 1$. We prove a lower bound of $\frac{3n-2}{n}$ of NSeqnash showing that for $n = 2$, the upper bound is tight. For linear latency functions, we further improve this bound to $\frac{4n^2}{(1+n)^2}$. For polynomial latency functions with nonnegative coefficients, we prove lower and upper bounds on the competitive ratio of NSeqnash that grow both exponentially in the degree of the considered polynomials. We further show that for parallel arcs, the competitive ratio is significantly lower. In particular, we show that in this case, the competitive ratio of the online algorithm NSeqnash does not exceed the price of anarchy of a related nonatomic network game in which all games of a given sequence are considered at the same time.

Furthermore, we consider online network games in which atomic players route their demand selfishly. Note that the atomic players may split their flow along different paths. The online algorithm ASeqNash, which produces a flow that is at Nash equilibrium for every game is $\min\{\frac{2(3\mathcal{K}+1)n}{n\mathcal{K}+3n+3\mathcal{K}+1}, \frac{5\mathcal{K}+1}{\mathcal{K}+5}, 4.92\}$-competitive for affine latency functions Here, $\mathcal{K}$ denotes the total number of players and n is the number of games within a given sequence. For general polynomial latency functions, we prove lower and upper bounds on the competitive ratio of ASeqNash that grow both exponentially in the degree of the considered polynomials. Finally, we prove better bounds for the parallel arc case for ASeqNash, relating the cost of ASeqNash to the cost of a nonatomic game, generalizing a result of Hayrapetyan, Tardos, and, Wexler [22].

To prove our main results (Theorem 1 and 2) we generalize the variational inequality approach previously used by Correa, Schulz, and Stier-Moses [11], Roughgarden [27], Cominetti, Correa, and Stier-Moses [10], and Harks [20]. The techniques used to prove upper bounds for ASeqNash in the parallel arc case (Theorem 4) are based on ideas of Hayrapetyan, Tardos, and, Wexler [22]. Our extended approach incorporates the known price of anarchy results as a special case with $n = 1$.

Note that the online algorithms NSeqNash and ASeqNash are fully distributed, hence, no coordination mechanism is needed to implement these algorithms. Furthermore, all results for the parallel arc case directly carry over to the online load balancing problem with parallel (splittable) jobs, where the objective is to minimize the L_2 norm of the server loads.

2 Online Network Games

An instance of the *Online Network Game* (OnlineNG) consists of a directed network $D = (V, A)$ together with nondecreasing continuous and convex latency functions $\ell_a : \mathbb{R}_+ \to \mathbb{R}_+$ for each arc $a \in A$. Furthermore, a sequence $\sigma = 1, \dots, n$ of network games are given. A network game i is characterized by a set of commodities $[K_i] := \{i1, \dots, i\, n_i\}$. For each commodity $ij \in [K_i]$, a flow of rate $d_{ij} > 0$ must be routed from the origin s_{ij} to the destination t_{ij}. The routing decision for game i is *online*, that is, it only depends on the routings of previous games $1, \dots, i-1$. Once the commodities of a game have been routed, they remain unchanged. Let $[\mathcal{K}] = \bigcup_{i=1}^{n} [K_i]$ denote the union of the sets $[K_1], \dots, [K_n]$. The total number of commodities is given by $\mathcal{K} = \sum_{i=1}^{n} n_i$.

A routing assignment, or *flow*, for commodity $ij \in [K_i]$ is a nonnegative vector $\boldsymbol{f}^{ij} \in \mathbb{R}_+^A$. This flow is *feasible*, if for all $v \in V$

$$\sum_{a \in \delta^+(v)} f_a^{ij} - \sum_{a \in \delta^-(v)} f_a^{ij} = \gamma_{ij}(v),$$

where $\delta^+(v)$ and $\delta^-(v)$ are the arcs leaving and entering v, respectively; furthermore, $\gamma_{ij}(v) = d_{ij}$, if $v = s_{ij}$, $-d_{ij}$ if $v = t_{ij}$, and 0, otherwise. Alternatively, one can consider a *path flow* for a commodity $ij \in [K_i]$. Let $\mathcal{P}_{ij}$ be the set of all paths

from s_{ij} to t_{ij} in D. A path flow is a nonnegative vector $(f_P^{ij})_{P\in\mathcal{P}_{ij}}$. The corresponding flow on link $a \in A$ for commodity $ij \in [K_i]$ is then $f_a^{ij} := \sum_{P\ni a} f_P^{ij}$. We denote by $f_a^i = \sum_{ij\in[K_i]} f_a^{ij}$ the aggregated flow of game i on link a. The total aggregate flow on link a is given by $f_a = \sum_{i=1}^n f_a^i$. We define $\mathcal{F}_i$ with $i \in [n]$ to be the set of vectors $(\boldsymbol{f}^1, \ldots, \boldsymbol{f}^i)$ such that $\boldsymbol{f}^j$ is a feasible flow for games $j = 1, \ldots, i$. If $(\boldsymbol{f}^1, \ldots, \boldsymbol{f}^i) \in \mathcal{F}_i$, we say that it is *feasible* for the sequence of network games $1, \ldots, i$. The entire flow for the sequence $1, \ldots, n$ of games is denoted by $\boldsymbol{f} = (\boldsymbol{f}^1, \ldots, \boldsymbol{f}^n)$.

The *current* cost of a feasible flow for game i on link $a \in A$ is given by $\ell_a\big(\sum_{j=1}^i f_a^j\big) f_a^i$. This expression can be obtained as the routing cost on arc a for a feasible flow for game i, *given* the flows $(\boldsymbol{f}^1, \ldots, \boldsymbol{f}^{i-1})$ of previous games $1, \ldots, i-1$ and *without* knowing about future games $j = i+1, \ldots, n$. The individual current cost for commodity $ij \in [K_i]$ on arc a is given by $\ell_a\big(\sum_{j=1}^i f_a^j\big) f_a^{ij}$. Note that this individual current cost on arc a may increase if later commodities are routed on a. The *total cost* of all sequentially played games is given by:

$$C(\boldsymbol{f}) = \sum_{a\in A} \ell_a(f_a)\, f_a = \sum_{a\in A} \ell_a\Big(\sum_{i=1}^n f_a^i\Big)\Big(\sum_{i=1}^n f_a^i\Big). \tag{1}$$

This cost function reflects the routing cost provided all commodities of the entire sequence of games have been routed. Thus, the cost of routing commodities of a sequence of games is not separable with respect to the games. That is, if an online algorithm routes flow for the games $i+1, \ldots, n$ along arcs that are used by commodities of games $1, \ldots, i$, the latter commodities may face higher individual cost on these arcs compared to their initial routing costs.

2.1 Player Types

Motivated by the source routing model in telecommunication networks, we focus on selfish behavior of players routing the demands d_{ij}, $ij \in [\mathcal{K}]$. In the following, we use the word commodity ij interchangeably with player ij to indicate that this player decides on the routing assignment $\boldsymbol{f}^{ij}$ for the demand d_{ij}.

In the nonatomic routing variant, we assume infinitely many agents carrying the flow of a player, where each agent controls only an infinitesimal fraction of the flow. This is in contrast to the atomic routing variant, where it is assumed that each player ij controls and coordinates the entire flow for his demand d_{ij}. For a sequence of games, we investigate the online algorithms NSeqNash and ASeqNash (a formal definition follows) that produce a feasible flow $\boldsymbol{f}^1, \ldots, \boldsymbol{f}^n \in \mathcal{F}_n$, where each $\boldsymbol{f}^i$ is at Nash equilibrium for the corresponding network game i.

2.2 Nash Equilibria for Nonatomic and Atomic Players

A flow for game i is at Nash equilibrium, if no player has an incentive to unilaterally change her strategy. We assume that players of game i decide on their strategies without taking future games $j = i+1, \ldots, n$ into account. It is straightforward to check that a Nash flow $\boldsymbol{f}^i$ for nonatomic players minimizes the

potential function $\Phi^i(\boldsymbol{f}) = \sum_{a\in A} \int_0^{f_a^i} \ell_a(\sum_{k=1}^{i-1} f_a^k + z)\, dz$, see for example Roughgarden and Tardos [30]. Furthermore, using convexity of the potential function two different Nash equilibria incur the same cost. The following conditions are necessary and sufficient to characterize a Nash equilibrium for game i.

Lemma 1. *A feasible flow $\boldsymbol{f}^i$ for the nonatomic game i is at Nash equilibrium if and only if it satisfies:*

$$\sum_{a\in A} \ell_a\Big(\sum_{k=1}^{i} f_a^k\Big)\,(f_a^i - x_a^i) \le 0 \text{ for all feasible flows } \boldsymbol{x}^i \text{ for game } i. \tag{2}$$

The proof is based on the first order optimality conditions and the convexity of the potential function $\Phi^i(\boldsymbol{f})$, see Dafermos and Sparrow [14].

Definition 1 (NSeqNash for the OnlineNG). *Consider an instance of the* OnlineNG *with a given sequence σ of n network games. The deterministic online algorithm* NSeqNash *produces a feasible flow $\boldsymbol{f} = (\boldsymbol{f}^1, \dots, \boldsymbol{f}^n) \in \mathcal{F}_n$, such that each flow $\boldsymbol{f}^k$ minimizes $\Phi^k(\boldsymbol{f})$, that is, each $\boldsymbol{f}^k$ is at Nash equilibrium for the corresponding games $k \in [n]$.*

Note that the problem of minimizing $\Phi^k(\boldsymbol{f})$ is well defined and admits an optimal solution with a unique objective value. Hence, NSeqNash is also well defined by this property. Since this convex program may have several different solutions (with the same objective value), the flow that NSeqNash produces is not necessarily unique. As this might contradict the notion of a *deterministic* online algorithm, we can advise a selection rule to make the flow unique. We omit this issue in the following, since our results hold for *every* sequence of Nash flows for the games $1, \dots, n$.

In network games with atomic players, some players may control a significant part of the entire demand. In the following, we characterize the strategy of an atomic player. It is straightforward to see that a best reply strategy for player ij of game i is to minimize its individual current cost $C^{ij}(\boldsymbol{f}) := \sum_{a\in A} \ell_a(\sum_{k=1}^{i} f_a^k) f_a^{ij}$.

The following conditions are necessary and sufficient to characterize a Nash equilibrium for game i.

Lemma 2. *A feasible flow $\boldsymbol{f}^i$ for the atomic game i is at Nash equilibrium if and only if for every player $ij \in [K_i]$ the following inequality is satisfied:*

$$\sum_{a\in A} \Big(\ell_a\Big(\sum_{k=1}^{i} f_a^k\Big) + \ell_a'\Big(\sum_{k=1}^{i} f_a^k\Big) f_a^{ij}\Big)\,(f_a^{ij} - x_a^{ij}) \le 0, \tag{3}$$

for all feasible flows $\boldsymbol{x}^{ij}$ for game i.

The proof relies on the convexity of $\ell_a(z)\, z$.

Definition 2 (ASeqNash for the OnlineNG). *Consider an instance of the* OnlineNG *with a given sequence σ of n network games. The deterministic*

online algorithm ASeqNash *produces a feasible flow* $\boldsymbol{f} = (\boldsymbol{f}^1, \ldots, \boldsymbol{f}^n) \in \mathcal{F}_n$, *such that each flow* $\boldsymbol{f}^{ij}$, $ij \in [K_i]$, $i \in [n]$ *minimizes* $C^{ij}(\boldsymbol{f})$, *that is, each* $\boldsymbol{f}^i$ *is at Nash equilibrium for the corresponding games* $i \in [n]$.

Since we assume convex latency functions, the minimization problem is well defined and admits an optimal solution with a unique objective value. Then, the existence of a flow at Nash equilibrium is guaranteed by the result of Rosen [26]. Hence, ASeqNash is also well defined by this property.

Finally, the *total offline optimum* minimizes the total cost $C(\boldsymbol{f})$ among all feasible flows. For a given sequence σ, we denote by $\textsc{Opt}(\sigma)$ the optimal value of this convex problem.

3 Competitive Analysis

For a solution $\boldsymbol{f}$ produced by an online algorithm Alg for a given sequence of games σ, we denote by $\textsc{Alg}(\sigma) = C(\boldsymbol{f})$ its cost. An online algorithm Alg is called (strictly) *c-competitive*, if the cost of Alg is never larger than c times the cost of an optimal offline solution. The *competitive ratio* of Alg is the infimum over all $c \geq 1$ such that Alg is c-competitive, see for instance Borodin and El-Yaniv [5] and Fiat and Woeginger [17].

3.1 Competitive Analysis for NSeqnash

In order to derive competitive results for NSeqnash for a sequence of games, we make use of the variational inequality (2). Using the notation $\vartheta_a^n(\ell_a, \boldsymbol{f}_a) := \ell_a(f_a)f_a - \sum_{i=1}^n \ell_a\big(\sum_{k=1}^i f_a^k\big) f_a^i$, we define for every $a \in A$, nonnegative vectors $\boldsymbol{f}_a, \boldsymbol{x}_a \in \mathbb{R}_+^{\mathcal{K}}$, and a nonnegative real number $\lambda \geq 0$, the following value (we assume by convention $0/0 = 0$):

$$\omega(\ell_a; n, \lambda) := \sup_{\boldsymbol{x}_a, \boldsymbol{f}_a \geq 0} \frac{\big(\ell_a(f_a) - \lambda\, \ell_a(x_a)\big)x_a + \vartheta_a^n(\ell_a, \boldsymbol{f}_a)}{\ell_a(f_a)f_a}. \tag{4}$$

Figure 1 illustrates the value $\omega(\ell_a; n, \lambda)$ for $n = 3$. For a given class $\mathcal{L}$ of nondecreasing latency functions we further define $\omega(\mathcal{L}; n, \lambda) := \sup_{\ell_a \in \mathcal{L}} \omega(\ell_a; n, \lambda)$. Furthermore, we define the following feasible set for the parameter λ.

Definition 3. *The feasible scaling set for* λ *is defined as*

$$\Lambda(\mathcal{L}, n) := \big\{\lambda \in \mathbb{R}^+ \big| \big(1 - \omega(\mathcal{L}; n, \lambda)\big) > 0\big\}.$$

Theorem 1. *Consider an instance of the* OnlineNG *involving a sequence of* n *games and latency functions in* $\mathcal{L}$. *Then, the competitive ratio of* NSeqnash *is at most*

$$\inf_{\lambda \in \Lambda(\mathcal{L}, n)} \left[\frac{\lambda}{1 - \omega(\mathcal{L}; n, \lambda)}\right].$$

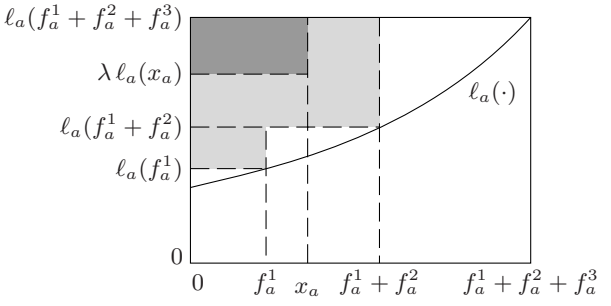


Fig. 1. Illustration of the value $\omega(\ell_a;\lambda,n)$ for $n=3$. The entire shaded area corresponds to the value $\vartheta_a^n(\ell_a,\boldsymbol{f})$. For some $\lambda>1$, the dark-gray shaded rectangle corresponds to the first term $\big(\ell_a(f_a)-\lambda\,\ell_a(x_a)\big)x_a$.

Proof. Let $\boldsymbol{f}$ be the flow generated by NSEQNASH and let $\boldsymbol{x}$ be any feasible flow for a given sequence of games $\sigma=1,\dots,n$. Then, we obtain:

$$\begin{aligned} C(\boldsymbol{f}) &\le \sum_{a\in A}\Big(\ell_a(f_a)\,f_a+\sum_{i=1}^{n}\ell_a\Big(\sum_{j=1}^{i}f_a^j\Big)\big(x_a^i-f_a^i\big)\Big) \qquad (5)\\ &\le \sum_{a\in A}\big(\vartheta_a^n(\ell_a,\boldsymbol{f}_a)+\ell_a(f_a)\,x_a\big)\\ &= \lambda\,C(\boldsymbol{x})+\sum_{a\in A}\big(\vartheta_a^n(\ell_a,\boldsymbol{f}_a)+\big(\ell_a(f_a)-\lambda\,\ell_a(x_a)\big)\,x_a\big)\\ &\le \lambda\,C(\boldsymbol{x})+\omega(\mathcal{L};n,\lambda)\,C(\boldsymbol{f}). \qquad (6)\end{aligned}$$

Here, (5) follows by applying the variational inequality in Lemma 1. The last inequality (6) follows from the definition of $\omega(\mathcal{L};n,\lambda)$ and since $\lambda\in\Lambda(\mathcal{L},n)$. Taking $\boldsymbol{x}$ as the optimal offline solution yields the claim.

In the following we relate the value $\omega(\mathcal{L};n,\lambda)$ to the anarchy value $\alpha(\mathcal{L})$ introduced by Roughgarden in [27], the parameter $\beta(\mathcal{L})$ introduced by Correa, Schulz, and Stier-Moses in [11], and the value $\omega(\mathcal{L};\lambda)$ introduced in Harks [20]. Our definition of $\omega(\mathcal{L},n,\lambda)$ is equal to $\omega(\mathcal{L};1)=\beta(\mathcal{L})=1-\frac{1}{\alpha(\mathcal{L})}$ if we have $\lambda=1$ and $n=1$. For arbitrary $\lambda\ge 0$ and $n=1$, we have $\omega(\mathcal{L},n,\lambda)=\omega(\mathcal{L},\lambda)$ as defined in Harks [20]. The difference between these two values is the nonnegative value $\vartheta_a^n(\ell_a,\boldsymbol{f}_a)$, which accounts for the online setting. It increases for $n\ge 1$ making the value $\omega(\mathcal{L},n,\lambda)$ larger and, hence, increases the competitive ratio.

Upper Bounds for Linear Latency Functions. In the following, we bound the value $\omega(\mathcal{L};n,\lambda)$ for affine linear latency functions. We start with some useful prerequisites.

Lemma 3. *For affine functions $\ell(z)=c_1z+c_0$, $c_1\ge 0, c_0\ge 0$, the value $\omega(\mathcal{L};n,1)$ is at most $\frac{3n-2}{4n}$.*

The proof of the lemma follows from the Cauchy-Schwarz inequality and the inequality $(f - x)\,x \leq \frac{1}{4}\,f^2$.

Equipped with the above lemma, we can apply Theorem 1 to derive an upper bound on the competitive ratio of NSEQNASH for affine latency functions.

Corollary 1. *If the latency functions of the* ONLINENG *are affine, the online algorithm* NSEQNASH *is* $\frac{4n}{n+2}$*-competitive, where* n *is the number of games.*

For $n = 1$, we obtain the bound of $\frac{4}{3}$ for nonatomic network games involving affine latency functions first proved in Roughgarden and Tardos [30].

For purely linear latency functions we can improve the upper bound by defining $\lambda := \frac{n}{n+1}$ below 1.

Corollary 2. *If the latency functions of the* ONLINENG *are linear, the online algorithm* NSEQNASH *is* $\frac{4n^2}{(n+1)^2}$*-competitive, where* n *is the number of games.*

Using a geometric proof as illustrated in Figure 1 the upper bound of 4 also holds for general continuous, nondecreasing, and concave latency functions.

Corollary 3. *If the latency functions of the* ONLINENG *are concave, the online algorithm* NSEQNASH *is* 4*-competitive.*

Upper Bounds for Polynomial Latency Functions. Now we consider the class $\mathcal{L}_d$ of polynomials with nonnegative coefficients and degree at most $d \in \mathbb{N}$:

$$\mathcal{L}_d := \{c_d\,x^d + \cdots + c_1\,x + c_0 \,:\, c_s \geq 0, s = 0, \ldots, d\}.$$

Note that polynomials in $\mathcal{L}_d$ are nonnegative for nonnegative arguments, nondecreasing, and convex. We can easily see that $\sup_{\boldsymbol{f}_a \geq 0} \vartheta_a^n(\ell_a, \boldsymbol{f}_a) \leq \frac{d}{d+1}\,\ell_a(f_a)\,f_a$ for $\ell_a \in \mathcal{L}_d$. Observe that the cost function $C(\boldsymbol{f})$ is linear in each of the latency functions $\ell_a(\cdot)$. Therefore, we can reduce the analysis to monomial price functions by subdividing each arc a into d arcs $a_1, \ldots, a_d$ with monomial latency functions $\ell_{a_s}(x) = c_s\,x^s$ for every $s = 1, \ldots, s$.

Lemma 4. *For the class* $\mathcal{M}_d$ *of monomials* $c_s\,x^s$ *of degree* $1 \leq s \leq d$ *and* $\lambda \geq 1$*, we have*

$$\omega(\mathcal{M}_d; n, \lambda) \leq \max_{0 \leq \mu} \mu - \lambda\,\mu^{d+1} + \frac{d}{d+1}.$$

Proof. For $\ell_a(\cdot) \in \mathcal{M}_d$, we can assume that $\ell_a(f_a)\,f_a > 0$, since otherwise $\omega(\mathcal{M}_d; n, \lambda) = 0$ and the claim is trivially true. By definition, we have

$$\omega(\ell_a; n, \lambda) = \sup_{x_a, f_a \geq 0} \frac{\big(\ell_a(f_a) - \lambda\,\ell_a(x_a)\big)\,x_a + \vartheta_a^n(\ell_a, \boldsymbol{f}_a)}{\ell_a(f_a)\,f_a}.$$

Defining $\mu := \frac{x_a}{f_a}$ (recall that $f_a > 0$), we obtain

$$\omega(\ell_a; n, \lambda) \leq \sup_{0 \leq \mu} \frac{\big(\ell_a(f_a) - \lambda\,\ell_a(\mu\,f_a)\big)\,\mu\,f_a}{\ell_a(f_a)\,f_a} + \frac{d}{d+1}.$$

Consider now the monomial price function $\ell_a(x_a) = c_s\, x_a^s$ of degree $s \in [d]$. To bound the value $\omega(\ell_a; n, \lambda)$ from above, we have to consider:

$$\sup_{0 \le \mu} \frac{(c_s\, f_a^s - \lambda\, c_s\, \mu^s\, f_a^s)\, \mu\, f_a}{c_s\, f_a^{s+1}} = \max_{0 \le \mu} \mu - \lambda\, \mu^{s+1}. \tag{7}$$

Because of the assumption $\lambda \ge 1$ the maximum is attained at a point with $\mu \le 1$. Thus, it follows that $\max_{0 \le \mu} \mu - \lambda\, \mu^{s+1} \le \max_{0 \le \mu} \mu - \lambda\, \mu^{d+1}$. This shows the claim.

Proposition 1. *For polynomial latency functions $\ell \in \mathcal{L}_d$ and $\lambda := (d+1)^{(d-1)}$, the value $\omega(\mathcal{L}; n, \lambda)$ is at most $\frac{d^2+2\,d}{(d+1)^2}$.*

Proof. The unique solution of the maximization problem in Lemma 4 is given by $\mu^* = \frac{1}{d+1}$. Evaluating the objective with $\lambda := (d+1)^{(d-1)}$ proves the claim:

$$\omega(\ell_a, n; \lambda) \le \frac{1}{d+1} - (d+1)^{(d-1)} \left(\frac{1}{d+1}\right)^{d+1} + \frac{d}{d+1} = \frac{d^2+2\,d}{(d+1)^2}.$$

Corollary 4. *Consider the* OnlineNG *with latency functions in $\mathcal{L}_d$. Then, the competitive ratio of the online algorithm* NSeqNash *is at most $(d+1)^{d+1}$.*

Proof. Let the flow $\boldsymbol{f}$ be produced by the online algorithm NSeqNash and let $\boldsymbol{x}$ be an arbitrary feasible flow for the OnlineNG. We define $\lambda := (d+1)^{(d-1)}$ and apply Proposition 1, which yields $\omega(\mathcal{L}; n, \lambda) \le \frac{d^2+2\,d}{(d+1)^2}$. In order to apply Theorem 1, we have to verify that $\lambda \in \Lambda(\mathcal{L}, n)$. What remains to be shown is that $1 - \frac{d^2+2\,d}{(d+1)^2} > 0$ holds. This inequality is equivalent to $\frac{1}{d+1} > 0$. Then, applying Theorem 1 yields

$$C(\boldsymbol{f}) \le \frac{(d+1)^{d-1}}{\left(1 - \frac{d^2+2\,d}{(d+1)^2}\right)} C(\boldsymbol{x}) = (d+1)^{d+1}\, C(\boldsymbol{x}).$$

Taking $\boldsymbol{x}$ as the optimal offline solution proves the claim.

3.2 Competitive Analysis for ASeqNash

In this section, we analyze the efficiency of the online algorithm ASeqNash, which produces a flow $\boldsymbol{f}^i$ that is at Nash equilibrium for every game i provided that we also allow for atomic players. Recently, Cominetti, Correa, and Stier-Moses [10] discovered that the price of anarchy may be quite large in network games with atomic players. Based on the work of Catoni and Pallotino [8], they presented an example, where the price of anarchy in a network game with atomic players is larger than that of the corresponding nonatomic game. As we show in this section, our upper bounds on the competitive ratio of the online algorithm ASeqNash also exceed that of NSeqNash. In the following we only present the main ideas. Complete proofs are left for the full version of this paper.

We define for every $a \in A$, for any nonnegative vectors $\boldsymbol{f}_a, \boldsymbol{x}_a \in \mathbb{R}_+^{\mathcal{K}}$ the value

$$\theta_a(\ell_a; \boldsymbol{f}_a, \boldsymbol{x}_a) := \sum_{i=1}^{n} \Big(\ell_a'(\sum_{k=1}^{i} f_a^k) \sum_{ij \in [K_i]} \big(f_a^{ij}\, x_a^{ij} - f_a^{ij}\, f_a^{ij} \big) \Big).$$

By assuming $0/0 = 0$, we further define

$$\omega(\ell_a; n, \mathcal{K}, \lambda) := \sup_{\boldsymbol{f}_a, \boldsymbol{x}_a \geq 0} \frac{\big(\ell_a(f_a) - \lambda\, \ell_a(x_a)\big) x_a + \vartheta_a^n(\ell_a, \boldsymbol{f}_a) + \theta_a(\ell_a; \boldsymbol{f}_a, \boldsymbol{x}_a)}{\ell_a(f_a) f_a}. \tag{8}$$

For a given class $\mathcal{L}$ of nondecreasing latency functions and a nonnegative real number $\lambda \geq 0$, we further define $\omega(\mathcal{L}; n, \mathcal{K}, \lambda) := \sup_{\ell_a \in \mathcal{L}} \omega(\ell_a, n, \mathcal{K}; \lambda)$. We define the following feasible set for the parameter λ.

Definition 4. *The feasible scaling set for λ is defined as*

$$\Lambda(\mathcal{L}, n, \mathcal{K}) := \big\{ \lambda \in \mathbb{R}^+ | \big(1 - \omega(\mathcal{L}; n, \mathcal{K}, \lambda)\big) > 0 \big\}.$$

Theorem 2. *Consider an instance of the* OnlineNG *involving a sequence of n games with $\mathcal{K}$ players and latency functions in $\mathcal{L}$. Then, the competitive ratio of* ASeqnash *is at most*

$$\inf_{\lambda \in \Lambda(\mathcal{L}, n, \mathcal{K})} \left[\frac{\lambda}{1 - \omega(\mathcal{L}; n, \mathcal{K}, \lambda)} \right].$$

The proof proceeds along the same lines as the proof of Theorem 1 except that the value $\omega(\ell_a; n, \mathcal{K}, \lambda)$ contains derivatives ℓ', which account for the ability of atomic players to coordinate their flow .

Linear and Polynomial Latency Functions. To facilitate the result of Theorem 2, we bound $\omega(\mathcal{L}; n, \mathcal{K}, \lambda)$ for linear latency functions.

Lemma 5. *For affine latency functions $\ell(z) = c_1\, z + c_0,\ c_1 \geq 0, c_0 \geq 0$, and $\lambda \geq 1$ the value $\omega(\mathcal{L}; n, \mathcal{K}, \lambda)$ is less than or equal to $\frac{4(\mathcal{K}-1)}{5\mathcal{K}+1}$.*

Applying Theorem 2 with $\lambda = 1$ yields the following result.

Corollary 5. *If the latency functions of the* OnlineNG *are affine, the online algorithm* ASeqnash *is $\frac{5\mathcal{K}+1}{\mathcal{K}+5}$-competitive, where $\mathcal{K}$ is the total number of players.*

Corollary 5 gives abound that only depends on the total number of players in the sequence σ of games. This bound states that ASeqnash is asymptotically 5-competitive for online atomic network games. By choosing $\lambda = 1.13$ and applying Theorem 2 it is possible to improve the upper bound to 4.92.

In the following, we derive a bound that depends on the number of games.

Corollary 6. *If the latency functions of the* OnlineNG *are affine, the online algorithm* ASeqnash *is $\frac{2(3\mathcal{K}+1)n}{n\mathcal{K}+3n+3\mathcal{K}+1}$-competitive, where n is the number of games and $\mathcal{K}$ is the total number of players.*

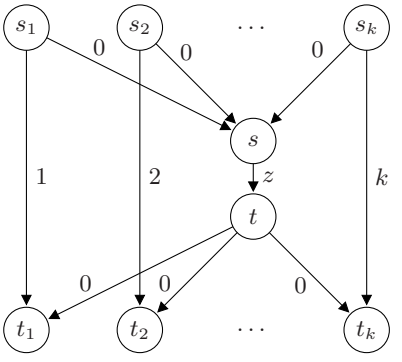


Fig. 2. Graph construction for the proof of the lower bound in Proposition 2 and Corollary 9

This bound is asymptotically 6-competitive. It provides, however, an explicit dependency on the number of games and players involved. For $n = 1$, we obtain a bound of $\frac{3\mathcal{K}+1}{2\mathcal{K}+2}$ for atomic network games with affine latency functions; this bound has previously been established by Cominetti, Correa and Stier-Moses [10]. For $\mathcal{K} \to \infty$ we can establish a bound of $\frac{6\,n}{n+3}$ that only depends on the number of games.

For latency functions in $\mathcal{L}_d$, we can show the following bounds.

Corollary 7. *If the latency functions of the* OnlineNG *are in* $\mathcal{L}_d$*, the competitive ratio of the online algorithm* ASeqNash *is at most* $\left(1 + \frac{5}{4}\,d + \frac{1}{4}\,d^2\right)^{d+1}$.

3.3 Lower Bounds

Based on an an instance presented in Harks, Heinz, and Pfetsch [21], we can easily show that any deterministic online algorithm for the OnlineNG has a competitive ratio greater then or equal to $\frac{4}{3}$ even for linear latencies. In the following, we present an increased lower bound for NSeqNash. Note that all lower bounds for NSeqNash also provide lower bounds for ASeqNash since we can simulate a nonatomic player by infinitely many atomic players each controlling a negligible fraction of the demand.

Proposition 2. *In case of affine latency functions, the online algorithm* NSeqNash *for the* OnlineNG *has a competitive ratio greater than or equal to* $\frac{3n-2}{n}$*, where* n *is the number of games.*

Proof. We consider the network presented in Figure 2 with the latency functions: $\ell_{(s_i,s)}(z) = 0$, $\ell_{(t,t_i)}(z) = 0$, $\ell_{(s_i,t_i)}(z) = i$, $i = 1, \ldots, k$, and $\ell_{(s,t)}(z) = z$. We consecutively release a sequence of games $(1, \ldots, k)$, where in each game j, there is a single player type $j1$. The demand of player type $j1$ is 1 that has to be routed from s_i to t_i, for $i = 1, \ldots, k$. Due to the choice of the affine terms i, NSeqNash routes for every game the corresponding demand over the arc from s to t. Then

we release the $(k+1)$-th game with demand d from s to t. Thus, the total cost for the sequence $\sigma = (1, \ldots, k+1)$ for NSeqnash with the new cost function is given by: $\text{NSeqnash}(\sigma) = (k+d)^2$. The optimal offline algorithm Opt routes the demands of the first k games along the direct arcs from s_i to t_i incurring cost of: $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$. The last demand in game $k+1$ is routed from s to t with cost d^2. The total cost for Opt is given by: $\text{Opt}(\sigma) = \frac{k(k+1)}{2} + d^2$. Replacing $k = n-1$ and setting $d = \frac{n}{2}$ yields

$$\frac{\text{NSeqnash}(\sigma)}{\text{Opt}(\sigma)} = \frac{2(k+d)^2}{k(k+1)+2d^2} = \frac{3n-2}{n}, \tag{9}$$

which proves the theorem.

Remark 1. For $n = 2$, the upper bound given in Corollary 1 is tight.

Based on the same instance, except using linear latency functions, we can prove a lower bound for purely linear latency functions.

Corollary 8. *For linear latency functions, the online algorithm* NSeqnash *for* OnlineNG *has a competitive ratio greater than or equal to* $\frac{33+5\sqrt{33}}{33+\sqrt{33}}$.

Corollary 9. *For latency functions in* $\mathcal{L}_d$*, the online algorithm* NSeqnash *for* OnlineNG *has a competitive ratio greater than or equal to* $\frac{d+1}{d+2}\, 2^{d+1}$.

The proof is again based on the instance in Figure 2 except that we use a monomial z^d for the (s,t) arc and the constant terms i become i^d. The rest of the proof then consists of technical calculations that are omitted.

The construction of the lower bounds show that the first player that routes its demand along the arc (s,t) experiences individual cost of $(k+x)$ after routing all commodities. In the common Nash equilibrium, where all players are adaptive and reroute their demand, this player would route its demand along the direct arc incurring cost of 1. Thus, the ratio of the individual cost of a nonadaptive player and that of an adaptive player is unbounded.

4 Parallel Arcs

For graphs that consist of two nodes and parallel arcs, we can show that NSeqnash performs not worse than a Nash flow for the entire game sequence that is played in parallel. In other words, for a given sequence of games, we compare the cost of NSeqnash to the cost of a Nash flow of a parallel game, where all players of the entire game sequence route their demands simultaneously.

For a given instance of the OnlineNG involving a sequence of games σ, we define the *parallel game* $\bar{\sigma}$ as a single game that contains all players of the sequence σ simultaneously.

Recall from the Wardrop condition [32] that a flow $\boldsymbol{f}$ is at Nash equilibrium if and only if the following condition is satisfied:

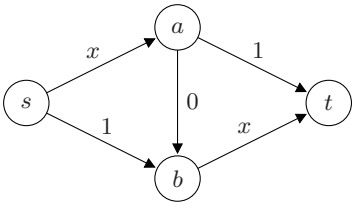


Fig. 3. Bad Example 1 based on the Graph of the Braess Paradox

Lemma 6. *A feasible flow $\boldsymbol{f}$ for the game $\bar{\sigma}$ is a Nash equilibrium if and only if:*

$$\ell_a(f_a) \leq \ell_{\hat{a}}(f_{\hat{a}}), \text{ for all arcs } a, \hat{a} \in A \text{ such that } f_a > 0. \tag{10}$$

Note that for nonatomic network games, Nash equilibria and Wardrop equilibria are the same. A similar condition holds for the flow that is produced by NSeqnash.

Lemma 7. *A feasible flow $\boldsymbol{f}$ for the sequence of games σ is produced by* NSeqnash *if and only if for all $k \in [n]$:*

$$\ell_a\Big(\sum_{i=1}^{k} f_a^i\Big) \leq \ell_{\hat{a}}\Big(\sum_{i=1}^{k} f_{\hat{a}}^i\Big), \text{ for all edges } a, \hat{a} \in A, \text{ such that } f_a^k > 0. \tag{11}$$

Theorem 3. *Let $D = (V, A)$ with $V = \{s, t\}$ and A a set of edges from s to t. We are given a sequence of games $\sigma = 1, \ldots, n$. Let $\boldsymbol{f}$ be a flow produced by* NSeqnash *for the nonatomic* OnlineNG *with a single nonatomic player routing d_i from s to t in every game $i \in [n]$. Let $\boldsymbol{f}^*$ be a flow at Nash equilibrium for the corresponding game $\bar{\sigma}$ with a single player routing $\sum_{i=1}^{n} d_i$ from s to t. Then, $C(\boldsymbol{f}) = C(\boldsymbol{f}^*)$.*

Proof. We prove that the flow $\boldsymbol{f}$ satisfies all conditions of Lemma 6 for the game $\bar{\sigma}$. By the uniqueness of the cost of a Nash equilibrium the claim is proven. The latency of the flow $\boldsymbol{f}$ on edge a is equal $\ell_a(f_a)$. By contradiction assume that there exist edges $a, \hat{a} \in A$ with $\ell_a(f_a) > \ell_{\hat{a}}(f_{\hat{a}})$, with $f_a > 0$. Let $k \in [n]$ be the largest index with $f_a^k > 0$. The existence of such an index k is granted since $f_a = \sum_{i=1}^{n} f_a^i > 0$ is assumed. As in games $k+1, \ldots, n$, the edge a is not used any more, we have that $\ell_a(f_a) = \ell_a\big(\sum_{i=1}^{k} f_a^i\big)$. Using the assumption that latency functions are nondecreasing it follows that $\ell_{\hat{a}}(f_{\hat{a}}) \geq \ell_{\hat{a}}\big(\sum_{i=1}^{k} f_{\hat{a}}^i\big)$. By Lemma 7 for game k, we have $\ell_{\hat{a}}\big(\sum_{i=1}^{k} f_{\hat{a}}^i\big) \geq \ell_a\big(\sum_{i=1}^{k} f_a^i\big)$, thus $\ell_{\hat{a}}(f_{\hat{a}}) \geq \ell_{\hat{a}}\big(\sum_{i=1}^{k} f_{\hat{a}}^i\big) \geq \ell_a\big(\sum_{i=1}^{k} f_a^i\big) = \ell_a(f_a)$, a contradiction.

The intuition of the above proof fails, however, for general networks with a single source and a single destination. To see this, we present an instance, where the cost of a flow $\boldsymbol{f}$ produced by NSeqnash is larger than that of the corresponding Nash flow $\boldsymbol{f}^*$ for the game $\bar{\sigma}$.

Example 1. Consider the graph of Braess's paradox in Figure 3 and two games that are released consecutively. Each game has a single nonatomic player routing one unit $d_1 = 1$, $d_2 = 1$ from s to t. The path system $\mathcal{P}_1$ for the first player contains $P_1 = (s, a, t)$, $P_2 = (s, a, b, t)$, $P_3 = (s, b, t)$. A flow that is at Nash equilibrium for the first game routes 1 unit of flow on P_2, having path latency $\ell_1(f^1) = 2$. In the second game, we route $\frac{1}{2}$ unit on P_1 and $\frac{1}{2}$ on P_3, both having path latency $\ell_2(\boldsymbol{f}) = 2.5$. Now $\ell^2_{P_2}(\boldsymbol{f}) = 3$. Thus, the total cost is $C(\boldsymbol{f}) = 1 \times 2.5 + 1 \times 3 = 5.5$. However, for the game $\bar{\sigma}$ we route 2 units of flow from s to t. Then, a flow $\boldsymbol{f}^*$ at Nash equilibrium routes one unit along paths P_1 and P_3. The path latencies are $\ell_{P_1}(\boldsymbol{f}) = \ell_{P_2}(\boldsymbol{f}) = 2$, thus the total cost is $C(\boldsymbol{f}^*) = 2 \times 2 = 4$.

In the following, we investigate the parallel arc setting for ASeqNash. For the atomic case in a parallel arc network, Hayrapetyan, Tardos and, Wexler [22] have proved a similar result. Assume we have a single game with an arbitrary number of atomic players, and $x\,\ell_a(x)$ is a convex function for all edges a. Then, the equilibrium cost is at most as much as the cost of the nonatomic Nash equilibrium of the total demand. Now we generalize this result for the OnlineNG.

We will compare a sequence of games with an arbitrary number of atomic players with a single game, where nonatomic players routes the flow $\boldsymbol{g}$ such that the total demand of the entire sequence is satisfied. Instead of comparing the usual costs, our result involves another cost function $C_2(\boldsymbol{f})$:

$$C_2(\boldsymbol{f}) = \sum_{i=1}^{n} \sum_{a \in A} \ell_a\Big(\sum_{j=1}^{i} f_a^j\Big) f_a^i. \tag{12}$$

$C_2(\boldsymbol{f})$ means that players in game i have to pay only after their current latency in game i, and no cost according to the games $i+1, \ldots, n$. Note that the relation $C_2(\boldsymbol{f}) \le C(\boldsymbol{f})$ holds because of the monotonicity of ℓ_a. In the following, we will show that $C_2(\boldsymbol{f}) \le C(\boldsymbol{g})$. Then, by defining $\delta := \sup_{\boldsymbol{f}} (C(\boldsymbol{f})/C_2(\boldsymbol{f}))$, we are still able to bound the usual cost: $C(\boldsymbol{f}) \le C_2(\boldsymbol{g}) \le \delta\, C(\boldsymbol{g})$. For latency functions $\ell \in \mathcal{L}_d$, we have for instance $\delta \le d + 1$. Now we are ready to state the following theorem:

Theorem 4. *Consider an instance of* OnlineNG *with a sequence of games $\sigma = 1, \ldots, n$ and the underlying graph $D = (V, A)$ with $V = \{s, t\}$ and A consisting of parallel st-edges. Let $\boldsymbol{f}$ be a flow produced by* ASeqNash *for this game with n_i atomic players in game i. Let $d = \sum_{i=1}^{n} \sum_{j=1}^{n_i} d_{ij}$ denote the total demand over all games. Let $\boldsymbol{g}$ be at Nash equilibrium for a single nonatomic game with a single player routing d units from s to t. Then $C_2(\boldsymbol{f}) \le C(\boldsymbol{g})$.*

For $n = 1$, this gives Theorem 2.3 in [22]. To prove the theorem, we introduce $d_i = \sum_{j=1}^{n_i} d_{ij}$, the total demand in game i. The essence of the proof is the following lemma:

Lemma 8. *Assume we have a sequence of two games with $D = (V, A)$ satisfying the conditions of Theorem 4. In the first game, n_1 atomic players route each d_{1j}*

units of flow from s to t. In the second game, there is a nonatomic player routing z units. Let $\boldsymbol{h} = (h^1, h^2)$ be in equilibrium for this sequence, and let $\boldsymbol{m}$ be in equilibrium for a single game with a single nonatomic player routing $d_1 + z$ from s to t. Then $C_2(\boldsymbol{h}) \leq C(\boldsymbol{m})$.

Before proving this lemma, we show how it implies Theorem 4. The proof is by induction. If $n = 1$, then we apply the lemma for $z = 0$. If $n > 1$, suppose we have proved the theorem for $n - 1$. Fix the flows of the first game, and modify the cost function ℓ_a to $q_a(x) = \ell_a(x + f_a^1)$. Consider the games $2, \ldots, n-1$ with cost function q_a, and let C_2^q denote the modified cost function. By definition, $(f^2, \ldots, f^n)$ is in equilibrium for this sequence, and $C_2(\boldsymbol{f}) = \sum_{a \in A} \ell_a(f_a^1) f_a^1 + C_2^q(f^2, \ldots, f^n)$.

Let $\boldsymbol{g}^q$ be a routing in Nash equilibrium of $z = \sum_{i=2}^n d_i$ units for the cost function q_a; let C^q denote its cost. By induction, $C_2^q(f^2, \ldots, f^n) \leq C^q$. Consider now the game described in the lemma. $\boldsymbol{h} = (f^1, \boldsymbol{g}^q)$ is in equilibrium, and the game $\boldsymbol{m}$ is identical to $\boldsymbol{g}$ as $d_1 + z = d$. By the lemma, $C_2(\boldsymbol{h}) \leq C(\boldsymbol{m}) = C(\boldsymbol{g})$.

On the other hand,

$$C_2(\boldsymbol{h}) = \sum_{a \in A} \ell_a(f_a^1) f_a^1 + C^q \geq \sum_{a \in A} \ell_a^1(f_a^1) f_a^1 + C_2^q(f^2, \ldots, f^n) = C_2(\boldsymbol{f}),$$

and this is what we wanted to prove.

Before proving Lemma 8, we prove two other lemmas, which are motivated by [22].

Lemma 9. *For the game as in Lemma 8, $\boldsymbol{h}$ is in equilibrium if and only if for any $j \in [n_1]$, and any edges $a, \hat{a} \in A$ with $h_a^{1j} > 0$,*

$$\ell_a(h_a^1) + h_a^{1j} \ell_a'(h_a^1) \leq \ell_{\hat{a}}(h_{\hat{a}}^1) + h_{\hat{a}}^{1j} \ell_{\hat{a}}'(h_{\hat{a}}^1). \tag{13}$$

Furthermore, for any edges $a, \hat{a} \in A$, if $h_a^2 > 0$, then $\ell_a(h_a) \leq \ell_{\hat{a}}(h_{\hat{a}})$.

This easily follows as player $1j$ wants to minimize $\sum_{a \in A} \ell_a(h_a^1) h_a^{1j}$. Let T denote the minimum edge latency of game $\boldsymbol{m}$ in Lemma 8. We call an edge a overloaded, if $\ell_a(h_a) > T$ and underloaded if $\ell_a(h_a) \leq T$. The idea is, following [22], that moving a small flow from an overloaded edge to an underloaded increases the cost.

Lemma 10. *Assume a is overloaded and $\hat{a}$ is underloaded with $h_a > 0$. Then $h_a^1 > 0$, and modifying h^1 by moving a small amount of flow from a to $\hat{a}$ does not decrease $C_2(\boldsymbol{h})$.*

Proof. Let T_2 denote the minimum edge latency in the Nash equilibrium of h^2. We show that $T_2 \leq T$. Assume by contradiction that $T_2 > T$. Then by moving flows from edges with latency higher than T_2 to edges of lower latency, we can finally arrive in a Nash-equilibrium of edge latency at least T_2, contradicting the fact that the edge latency is the same in any two different Nash-equilibria, see Roughgarden [28].

By $T_2 \leq T$, if a is overloaded, then $h_a^2 = 0$. This implies $h_a^1 > 0$, and $\ell_a(h_a) = \ell_a(h_a^1)$. Our aim is to prove, that decreasing h_a^1 a little bit and increasing $h_{\hat{a}}^1$ with the same amount, $C_2(\boldsymbol{h})$ does not decrease. The statement follows if we can prove that $\frac{\partial C_2}{\partial h_a^1} \leq \frac{\partial C_2}{\partial h_{\hat{a}}^1}$. This is equivalent with

$$\ell_a(h_a^1) + h_a^1 \ell_a'(h_a^1) + h_a^2 \ell_a'(h_a^1 + h_a^2) \leq \ell_{\hat{a}}(h_{\hat{a}}^1) + h_{\hat{a}}^1 \ell_{\hat{a}}'(h_{\hat{a}}^1) + h_{\hat{a}}^2 \ell_{\hat{a}}'(h_{\hat{a}}^1 + h_{\hat{a}}^2). \quad (14)$$

Let $B = \{j : h_a^{1j} > 0\}$. If we sum (13) for $j \in B$, we get

$$|B|\ell_a(h_a^1) + h_a^1 \ell_a'(h_a^1) \leq |B|\ell_{\hat{a}}(h_{\hat{a}}^1) + \sum_{j \in B} h_{\hat{a}}^{1j} \ell_{\hat{a}}'(h_{\hat{a}}^1)$$

Increasing the right hand side by $\sum_{j \notin B} h_{\hat{a}}^{1j} \ell_{\hat{a}}'(h_{\hat{a}}^1) + h_{\hat{a}}^2 \ell_{\hat{a}}'(h_{\hat{a}}^1 + h_{\hat{a}}^2)$, and adding $h_a^2 \ell_a'(h_a^1 + h_a^2) = 0$ to the left hand side, we get

$$|B|\ell_a(h_a^1) + h_a \ell_a'(h_a) + h_a^2 \ell_a'(h_a^1 + h_a^2) \leq |B|\ell_{\hat{a}}(h_{\hat{a}}^1) + h_{\hat{a}}^1 \ell_{\hat{a}}'(h_{\hat{a}}^1) + h_{\hat{a}}^2 \ell_{\hat{a}}'(h_{\hat{a}}^1 + h_{\hat{a}}^2)$$

As a is overloaded and $\hat{a}$ is underloaded, we have $\ell_a(h_a^1) = \ell_a(h) > \ell_{\hat{a}}(h) \geq \ell_{\hat{a}}(h_{\hat{a}}^1)$. This in turn implies (14).

Now we are ready to prove Lemma 8. We modify $\boldsymbol{h}$ by moving flow amounts from overloaded to underloaded links. We avoid creating new overloaded links: we increase the flow on the underloaded edge $\hat{a}$ so that $\ell_{\hat{a}}(h_{\hat{a}})$ should not exceed T. Applying such a modification maintains $h_a^2 = 0$ on any overloaded edge a. Observe that (14) holds not only for h, but for any $\bar{h}$ satisfying $\bar{h}_a^1 \leq h_a^1$ and $\bar{h}_{\hat{a}}^1 \geq h_{\hat{a}}^1$ and $\bar{h}^2 = h^2$. This is since the monotonicity and convexity of $\ell_a(x)$ implies that $x\ell_a(x + c)$ is as well convex for $c, x \geq 0$.

This ensures that we can always go on by moving flow from overloaded to underloaded links, as far as some modifications are applicable. Suppose no more modifications can be applied. In this case for each edge a, $\ell_a(h_a) \geq T$. If $\ell_a(h_a) = T$ for all edges, then Lemma 8 follows. Otherwise we have some edge a with $\ell_a(h_a) > T$, and for all edges $\hat{a}$ with $\ell_{\hat{a}}(h_{\hat{a}}) = T$, increasing $h_{\hat{a}}$ by an arbitrary small positive amount would result in $\ell_{\hat{a}}(h) > T$. But this is a contradiction, as now the flows can be rerouted to obtain a Nash-equilibrium with edge latency strictly larger than T. Again we use that the edge latency is the same in any two different Nash-equilibria, and T was the latency of the equilibrium state $\boldsymbol{m}$.

References

1. Altman, E., Basar, T., Jimenez, T., Shimkin, N.: Competitive routing in networks with polynomial costs. IEEE Trans. Automat. Control 47(1), 92–96 (2002)
2. Awerbuch, B., Azar, Y., Epstein, A.: The price of routing unsplittable flow. In: Proc. of the thirty-seventh annual ACM symposium on Theory of computing (STOC), pp. 57–66. ACM Press, New York (2005)
3. Awerbuch, B., Azar, Y., Grove, E.F., Kao, M.-Y., Krishnan, P., Vitter, J.S.: Load balancing in the L_p norm. In: IEEE Symposium on Foundations of Computer Science (FOCS), pp. 383–391 (1995)

4. Awerbuch, B., Azar, Y., Plotkin, S.: Throughput-competitive on-line routing. In: Proc. 34th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Palo Alto, pp. 32–40 (1993)
5. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
6. Braess, D.: Über ein Paradoxon der Verkehrsplanung. Unternehmenforschung 11, 258–268 (1968)
7. Caragiannis, I., Flammini, M., Kaklamanis, C., Kanellopoulos, P., Moscardelli, L.: Tight bounds for selfish and greedy load balancing. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 311–322. Springer, Heidelberg (2006)
8. Catoni, S., Pallotino, S.: Traffic equilibrium paradoxes. Transportation Science 25, 240–244 (1991)
9. Christodoulou, G., Koutsoupias, E.: The price of anarchy of finite congestion games. In: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing (STOC), pp. 67–73 (2005)
10. Cominetti, R., Correa, J.R., Stier-Moses, N.: Network games with atomic players. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, Springer, Heidelberg (2006)
11. Correa, J.R., Schulz, A.S., Stier-Moses, N.E.: Selfish routing in capacitated networks. Math. Oper. Res. 29, 961–976 (2004)
12. Correa, J.R., Schulz, A.S., Stier-Moses, N.E.: On the inefficiency of equilibria in congestion games. In: Jünger, M., Kaibel, V. (eds.) Integer Programming and Combinatorial Optimization. LNCS, vol. 3509, pp. 167–181. Springer, Heidelberg (2005)
13. Czumaj, A., Vöcking, B.: Tight bounds for worst-case equilibria. In: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms (SODA), pp. 413–420 (2002)
14. Dafermos, S.C., Sparrow, F.T.: The traffic assignment problem for a general network. J. Res. Natl. Bur. Stand. Sect. B 73, 91–118 (1969)
15. Dubey, P.: Inefficiency of Nash Equilibria. Math. Oper. Res. 11, 1–8 (1986)
16. Even-Dar, E., Mansour, Y.: Fast convergence of selfish rerouting. In: Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA), pp. 772–781 (2005)
17. Fiat, A. (ed.): Online Algorithms. LNCS, vol. 1442. Springer, Heidelberg (1998)
18. Fischer, S., Vöcking, B.: Adaptive routing with stale information. In: Aguilera, M.K., Aspnes, J. (eds.) Proc. 24th Ann. ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing (PODC), Las Vegas, NV, USA, July 2005, pp. 276–283. ACM Press, New York (2005)
19. Friedman, E.J.: Genericity and congestion control in selfish routing. In: Decision and Control, CDC. 43rd IEEE Conference on, pp. 4667–4672 (2004)
20. Harks, T.: On the price of anarchy of network games with nonatomic and atomic players. Technical report, avalaible at Optimization Online (January 2007)
21. Harks, T., Heinz, S., Pfetsch, M.E.: Online multicommodity routing problem. In: Erlebach, T., Kaklamanis, C. (eds.) WAOA 2006. LNCS, vol. 4368, Springer, Heidelberg (2007)
22. Hayrapetyan, A., Tardos, E., Wexler, T.: The effect of collusion in congestion games. In: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing (STOC), pp. 89–98 (2006)
23. Koutsoupias, E., Papadimitriou, C.H.: Worst-case equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)

24. La, R.J., Walrand, J., Anantharam, V.: Issues in TCP Vegas. Electronics Research Laboratory, University of California, Berkeley, UCB/ERL Memorandum, No. M99/3 (January 1999)
25. Qiu, L., Yang, R.Y., Zhang, Y., Shenker, S.: On selfish routing in internet-like environments. IEEE/ACM Trans. on Netw. 14(4), 725–738 (2006)
26. Rosen, J.B.: Existence and uniqueness of equilibrium points for concave n-person games. Econometrica 33, 520–534 (1965)
27. Roughgarden, T.: The price of anarchy is independent of the network topology. Journal of Computer and System Science 67, 341–364 (2002)
28. Roughgarden, T.: Selfish Routing and the Price of Anarchy. The MIT Press, Cambridge (2005)
29. Roughgarden, T.: Selfish routing with atomic players. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 973–974 (2005)
30. Roughgarden, T., Tardos, E.: How bad is selfish routing? Journal of the ACM 49(2), 236–259 (2002)
31. Suri, S., Toth, C., Zhou, Y.: Selfish load balancing and atomic congestion games. Algorithmica 47(1), 79–96 (2007)
32. Wardrop, J.G.: Some theoretical aspects of road traffic research. In: Proceedings of the Institute of Civil Engineers, 1(Part II), pp. 325–378 (1952)

Vertex Pursuit Games in Stochastic Network Models*

Anthony Bonato[1], Paweł Prałat[2], and Changping Wang[1]

[1] Wilfrid Laurier University
Waterloo, Canada
abonato@rogers.com, cwang@wlu.ca
[2] Dalhousie University
Halifax, Canada
pralat@mathstat.dal.ca

Dedicated to the memory of Aubrey C. Hamlyn

Abstract. Random graphs with given expected degrees $G(\mathbf{w})$ were introduced by Chung and Lu so as to extend the theory of classical $G(n,p)$ random graphs to include random power law graphs. We investigate asymptotic results for the game of Cops and Robber played on $G(\mathbf{w})$ and $G(n,p)$. Under mild conditions on the degree sequence $\mathbf{w}$, an asymptotic lower bound for the cop number of $G(\mathbf{w})$ is given. We prove that the cop number of random power law graphs with n vertices is asymptotically almost surely $\Theta(n)$. We derive concentration results for the cop number of $G(n,p)$ for p as a function of n.

1 Introduction

Vertex pursuit games, such as Cops and Robber, may be viewed of as a simplified model for network security. As a general motivation for these games, suppose that an intruder (the robber) is loose on a network, and travels between adjacent vertices in an effort to escape the authorities (the cops). The intruder could be a virus or hacker, or some other malicious agent. The goal is to minimize the resources (that is, number of cops) required to capture the intruder.

The game of *Cops and Robber*, introduced independently by Nowakowski and Winkler [9] and Quilliot [10] over twenty years ago, is played on a fixed graph G. We will assume in this paper that G is undirected, simple, and finite. There are two players, a set of k *cops* (or *searchers*), where $k > 0$ is a fixed integer, and the *robber*. The cops begin the game by occupying a set of k vertices. The robber then chooses a vertex, and the cops and robber move in alternate rounds. The players use edges to move from vertex to vertex. More than one cop is allowed to occupy a vertex, and the players may remain on their current vertex. The players know each others current locations and can remember all the previous moves. The cops win and the game ends if at least one of the cops can eventually

* The authors gratefully acknowledge support from NSERC and MITACS.

J. Janssen and P. Prałat (Eds.): CAAN 2007, LNCS 4852, pp. 46–56, 2007.
© Springer-Verlag Berlin Heidelberg 2007

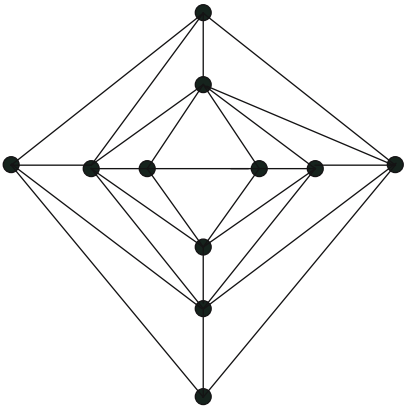


Fig. 1. A cop-win graph

occupy the same vertex as the robber; otherwise, the robber wins. As placing a cop on each vertex guarantees that the cops win, we may define the *cop number*, written $c(G)$, which is the minimum number of cops needed to win on G. The cop number was introduced by Aigner and Fromme [1] who proved that if G is planar, then $c(G) \leq 3$.

So-called *cop-win* graphs (that is, graphs G with $c(G) = 1$) were structurally characterized in [9,10]. See Figure 1 for a cop-win graph. If x is a vertex, then define $N[x]$ to be x along with the vertices joined to x. The cop-win graphs are exactly those graphs which are *dismantlable*: there exists a linear ordering $(x_j : 1 \leq j \leq n)$ of the vertices so that for all $2 \leq j \leq n$, there is a $i < j$ such that $N[x_j] \subseteq N[x_i]$. No analogous structural characterization of graphs with cop number k, where $k > 1$ is a fixed integer, is known; this is a central open problem in the subject. For a survey of results on the cop number and related search parameters for graphs, see [2].

In the last few years there was an explosion of mathematical research related to stochastic models of real-world networks, especially for models of the web graph. Many technological, social, biological networks have properties similar to those present in the web, such as power law degree distributions and the small world property. We refer to these networks as *self-organizing*. For example, power laws have been observed in protein-protein interaction networks, and networks formed by scientific collaborators. While much of the earlier mathematical work on self-organizing networks focused on designing models satisfying certain properties such as power law degree distributions, new approaches are constantly emerging.

We study vertex pursuit games in models for stochastic network models in self-organizing networks. To our best knowledge, our work is the first to consider such games in these network models. We consider Erdős, Rényi $G(n, p)$ random graphs and their generalizations used to model self-organizing networks. Define a probability space on graphs of a given order $n \geq 1$ as follows. Fix a vertex set V consisting of n distinct elements, usually taken as $[n] = \{1, 2, \ldots, n\}$, and fix $p \in [0, 1]$. Note that p can be a function of n. Define the space of *random graphs*

of order n with edge probability p, written $G(n,p)$, with sample space equalling the set of all $2^{\binom{n}{2}}$ (labelled) graphs with vertex set V, and

$$\mathbb{P}(G) = p^{|E(G)|}(1-p)^{\binom{n}{2}-|E(G)|}.$$

Informally, we may view $G(n,p)$ as of graphs with vertex set V, so that two distinct vertices are joined independently with probability p.

The cop number of $G(n,p)$ was studied in [3], where the following result was proved. In this paper, all asymptotics are as $n \to \infty$. We say that an event holds *asymptotically almost surely* (*a.a.s.*) if the probability that it holds tends to 1 as n goes to infinity.

Theorem 1. *Let $0 < p < 1$ be fixed. For every real $\varepsilon > 0$ a.a.s. for $G \in G(n,p)$*

$$(1-\varepsilon)\log_{\frac{1}{1-p}} n \le c(G) \le (1+\varepsilon)\log_{\frac{1}{1-p}} n.$$

Recent work of Chung and Lu [5,6] supplies an extension of the $G(n,p)$ random graphs to random graphs with given expected degree sequence $\mathbf{w}$. The corresponding probability space is referred to as $G(\mathbf{w})$. For example, if $\mathbf{w}$ follows a power law distribution, then $G(\mathbf{w})$ supplies a model for self-organizing networks. We will define $G(\mathbf{w})$ precisely in the next section.

The results in this paper are divided into two parts: bounding the cop number of random graphs with given expected degree and random power law graphs (Section 2), and the cop number of $G(n,p)$ random graphs where p is a function of n (Section 3). Our approach in both sections is to exploit dominating sets to give upper bounds for the cop number, while lower bounds usually follow by considering certain adjacency properties. In random power law graphs, we prove in Theorem 3 that the cop number is $\Theta(n)$.

2 The Cop Number in Random Graphs with Given Expected Degree Sequence

Let

$$\mathbf{w} = (w_1, \ldots, w_n)$$

be a sequence of n real nonnegative real numbers. We define a random graph model, written $G(\mathbf{w})$, as follows. Vertices are integers in $[n]$. Each potential edge between i and j is chosen independently with probability $p_{ij} = w_i w_j \rho$, where

$$\rho = \frac{1}{\sum_{i=1}^{n} w_i}.$$

We will always assume that

$$\max_i w_i^2 < \sum_{i=1}^{n} w_i,$$

which implies that $p_{ij} \in [0,1)$. The model $G(\mathbf{w})$ is referred to as *random graphs with given expected degree sequence* $\mathbf{w}$. Observe that $G(n,p)$ may be viewed as a special case of $G(\mathbf{w})$ by taking $\mathbf{w}$ to be equal the constant n-sequence $(pn, pn, \ldots, pn)$.

In our main result of this section, we supply an asymptotic lower bound for the cop number of graphs $G \in G(\mathbf{w})$ which generalizes the lower bound from Theorem 1. Our results demonstrate that a logarithmic lower bound is ubiquitous in random graphs with given expected degrees satisfying our conditions. Let $M = \max_i w_i$ and $m = \min_i w_i$.

Theorem 2. *Suppose that* $\mathbf{w}$ *be a sequence satisfying*

$$0 < q_0 \leq m^2\rho \leq M^2\rho \leq p_0 < 1,$$

where p_0 *and* q_0 *are fixed real numbers in* $(0,1)$. *Then for all* $\varepsilon \in (0,1)$ *with probability at least* $1 - \exp(-\Theta(n^\varepsilon))$, $G \in G(\mathbf{w})$ *satisfies*

$$c(G) \geq (1-\varepsilon)\log_{\frac{1}{1-p_0}} n.$$

One interpretation of Theorem 2 is that as the network order doubles, on average $\Theta(1)$ more cops are needed to guard the network. For the proof of Theorem 2, we use the following lemma.

Lemma 1. *Let* $0 < p < 1$, $r > 0$, *and* $\varepsilon \in (0,1)$ *be fixed. If*

$$d = \left(\log \frac{1}{1-p}\right)^{-1} (1-\varepsilon),$$

then

$$n^{\lfloor d\log n\rfloor+1}\left(1 - r(1-p)^{\lfloor d\log n\rfloor}\right)^{n-\lfloor d\log n\rfloor-1} \leq \exp(-\Theta(n^\varepsilon)). \tag{1}$$

Proof. It is enough to prove that

$$n^{d\log n+1}\left(1 - r(1-p)^{d\log n}\right)^{n-d\log n-1} \leq \exp(-\Theta(n^\varepsilon)).$$

Now

$$\begin{aligned} n^{d\log n+1}(1 - r(1-p)^{d\log n})^{n-d\log n-1} &= n^{d\log n+1}\left(1 - \frac{r}{n^{1-\varepsilon}}\right)^{n-d\log n-1} \\ &= \exp\left(f(n)\right), \end{aligned}$$

where

$$f(n) = (d\log n + 1)\log n + (n - d\log n - 1)\log\left(1 - \frac{r}{n^{1-\varepsilon}}\right).$$

However, $\exp\left(f(n)\right) \leq \exp(-\Theta(n^\varepsilon))$.

Proof of Theorem 2. We employ the following adjacency property. For a fixed $k > 0$ an integer, we say that G is $(1,k)$-*e.c.* if for each k-set S of vertices of G and vertex $u \notin S$, there is a vertex $z \notin S$ not joined to a vertex in S and joined to u. It is easy to see that if G is $(1,k)$-e.c., then $c(G) \geq k$ (the robber may use the property to escape to a vertex not joined to any vertex occupied by a cop). Let $k = \left\lfloor (1-\varepsilon) \log_{\frac{1}{1-p_0}} n \right\rfloor$. For any graph $G \in G(\mathbf{w})$ we claim that a.a.s. G is $(1,k)$-e.c. Once this is proved, the desired lower bound for the cop number will follow.

Fix S a k-subset of vertices of G and a vertex u not in S. For a vertex

$$x \in U = V(G) \backslash (S \cup \{u\}),$$

the probability that a vertex x is joined to u and to no vertex of S is

$$p_{xu} \prod_{v \in S} (1 - p_{xv}).$$

Since for $x, y \in U, x \neq y$ and for $v \in S$, the edges xv are chosen independently of the edges yv, the probability that no suitable vertex can be found for this particular S and u is

$$\prod_{x \in U} \left(1 - p_{xu} \prod_{v \in S} (1 - p_{xv}) \right) \leq \left(1 - p'(1-p_0)^k\right)^{n-k-1},$$

where

$$p' = \min_{x \in U} p_{xu}.$$

By hypothesis, $p' \geq q_0 > 0$.

The probability that there exists S and u for which no suitable x can be found is at most

$$n^{k+1}(1 - q_0(1-p_0)^k)^{n-k-1}.$$

By Lemma 1 with $q_0 = r$, $p_0 = p$, we have that

$$n^{k+1}(1 - q_0(1-p_0)^k)^{n-k-1} \leq \exp(-\Theta(n^{\varepsilon})),$$

and the theorem follows. □

In general power law graphs, there may exist an abundance of isolated vertices, even as much as $\Theta(n)$ many. Since the cop number is bounded from below by the number of isolated vertices, we expect the cop number of $G(\mathbf{w})$ to be around cn, for a constant $c \in (0,1)$. We show rigorously that this is indeed the case for random power law graphs, which we now introduce.

Given $\beta > 2$, $d > 0$, and a function $M = M(n)$ (with M tending to infinity with n), we consider the random graph with given expected degrees $w_i > 0$, where

$$w_i = ci^{-\frac{1}{\beta-1}} \tag{2}$$

for i satisfying $i_0 \le i < n + i_0$. The term c depends on β and d, and i_0 depends also on M; namely,

$$c = \left(\frac{\beta-2}{\beta-1}\right) d n^{\frac{1}{\beta-1}}, \qquad i_0 = n\left(\frac{d}{M}\left(\frac{\beta-2}{\beta-1}\right)\right)^{\beta-1}. \tag{3}$$

It is not hard to show (see [5,6]) that a.a.s. the random graphs with the expected degrees satisfying (2) and (3) follow a power law degree distribution with exponent β, average degree $d(1+o(1))$, and maximum degree $M(1+o(1))$.

We prove the following result for the cop number of a random power law graph, showing the cop number is a.a.s. equal to $\Theta(n)$.

Theorem 3. *For a random power law graph $G(\mathbf{w})$ with exponent $\beta > 2$ and average degree d, for all $\varepsilon > 0$, a.a.s. the following hold.*

1. *If X is the random variable denoting the number of isolated vertices in $G(\mathbf{w})$, then*
$$X \quad = (1+o(1))n \int_0^1 \exp\left(-d\frac{\beta-2}{\beta-1}x^{-1/(\beta-1)}\right) dx.$$
2. *For $a \in (0,1)$, define*
$$f(a) = a + \int_a^1 \exp\left(-d\frac{\beta-2}{\beta-1}a^{(\beta-2)/(\beta-1)}x^{-1/(\beta-1)}\right) dx.$$
Then
$$c(G) \quad \le \quad (1+o(1))n \min_{0<a<1} f(a).$$

The theorem demonstrates that the cop number of random power law graphs is a.a.s. $\Theta(n)$, and so is of much larger order than the logarithmic cop number of $G(n,p)$ random graphs. Hence, we should expect in real-world power law graphs such as the web graph that the cop number of order is large, and it would be interesting to conduct experiments which corroborate this claim.

The integrals in the statement of Theorem 3 do not possess closed-form solutions in general. For the integral in item 1, we have that

$$\int_0^1 \exp\left(-tx^{-1/(\beta-1)}\right) dx = e^{-t}\sum_{j=0}^{\infty}\frac{\Gamma(2-\beta)}{\Gamma(2-\beta+j)}t^j + \frac{\pi\csc(\pi\beta)t^{\beta-1}}{\Gamma(\beta-1)},$$

where $t = \frac{d(\beta-2)}{\beta-1}$. The integral in item 2 may be evaluated in cases depending on β. For example, if $2 < \beta < 3$, then the integral

$$\int_a^1 \exp\left(-t\alpha_2 x^{-1/(\beta-1)}\right) dx$$

equals

$$e^{-\alpha_2 t}\sum_{j=0}^{\infty}\frac{\Gamma(2-\beta)}{\Gamma(2-\beta+j)}t^j{\alpha_2}^j - e^{-\alpha_1 t}\sum_{j=0}^{\infty}\frac{\Gamma(2-\beta)}{\Gamma(2-\beta+j)}t^j{\alpha_1}^j + \frac{\pi\csc(\pi\beta)t^{\beta-1}}{\Gamma(\beta-1)}t^{\beta-1}a^{\beta-3}(a-1),$$

where $\alpha_1 = a^{(\beta-3)/(\beta-1)}$ and $\alpha_2 = a^{(b-2)/(b-1)}$.

We supply numerical values for lower/upper bounds of the cop number of $G(\mathbf{w})$ when $d = 10, 20$ and $\beta = 2.1, 2.7$.

	10	20
2.1	0.1806/0.2940	$0.5112 \cdot 10^{-1}$/0.1265
2.7	$0.4270 \cdot 10^{-2}$/0.1895	$0.4205 \cdot 10^{-4}$/$0.8261 \cdot 10^{-1}$

The proof of Theorem 3 requires some background on the domination number of a graph. A set of vertices S is a *dominating set* in G if each vertex not in S is joined to some vertex of S. The *domination number* of G, written $\gamma(G)$, is the minimum cardinality of a dominating set in G. An easy observation is that

$$c(G) \leq \gamma(G), \tag{4}$$

(place a cop on each vertex of dominating set with minimum cardinality). However, if $n \geq 2$, then $c(P_n) = 1$ (where P_n is a path with n vertices) and $\gamma(P_n) = \lceil \frac{n}{3} \rceil$. The bound of (4) while useful, is far from tight in general. Domination in models for self-organizing networks were considered in Cooper et al. [7].

Proof of Theorem 3. The probability that the vertex i for $i_0 \leq i < n + i_0$ (that is, the vertex i corresponds to the weight w_i) is isolated is equal to

$$\begin{aligned} p_i &= \prod_{j,j\neq i} (1 - w_i w_j \rho) \\ &= \prod_{j,j\neq i} \exp\left(-(1+o(1))w_i w_j \rho\right) \\ &= \exp\left(-(1+o(1))w_i \rho \sum_{j,j\neq i} w_j\right) \\ &= \exp\left(-(1+o(1))w_i\right) . \end{aligned} \tag{5}$$

Let X_i be an indicator random variable for the event that the vertex i is isolated. Then

$$\mathbb{P}(X_i = 1) = 1 - \mathbb{P}(X_i = 0) = p_i$$

for $i_0 \leq i < n + i_0$.

As $X = \sum_{i_0 \leq i < n+i_0} X_i$, it follows from (5) that the expected value of X is

$$\begin{aligned} \sum_{i_0 \leq i < n+i_0} p_i &= (1+o(1))n \int_0^1 \exp\left(-(1+o(1))c(xn)^{-1/(\beta-1)}\right) dx \\ &= (1+o(1))n \int_0^1 \exp\left(-d\frac{\beta-2}{\beta-1}x^{-1/(\beta-1)}\right) dx\,. \end{aligned}$$

A sum of independent random variables with large enough expected value is not too far from its mean (see, for example, Theorem 2.8 in [8]). Thus, the number of isolated vertices in $G(\mathbf{w})$ is a.a.s. equal to

$$X = (1+o(1))n \int_0^1 \exp\left(-d\frac{\beta-2}{\beta-1}x^{-1/(\beta-1)}\right) dx\,.$$

Item (1) now follows.

For item (2), we apply (4). Consider $A \subset V$ of first $\lfloor an \rfloor$ vertices from $i_0, \ldots, n+i_0$. Let $B \subset V \setminus A$ denote the set of vertices that do not have a neighbour in A. Then $D = A \cup B$ is a dominating set, and we now estimate the cardinality of D.

Consider the vertex i, $an < i < n + i_0$. Since $i_0 = o(n)$, there is $b \in (0,1]$ such that $i = (1+o(1))bn$. The probability that i does not have a neighbour in A is equal to

$$\begin{aligned}
q_i &= \prod_{j<an+i_0} (1 - w_i w_j \rho) \\
&= \exp\left(-(1+o(1))w_i\rho \sum_{j<an+i_0} w_j\right) \\
&= \exp\left(-(1+o(1))c(bn)^{-1/(\beta-1)}(dn)^{-1}n\int_0^a c(xn)^{-1/(\beta-1)}dx\right) \\
&= \exp\left(-(1+o(1))d\Big(\frac{\beta-2}{\beta-1}\Big)^2 b^{-1/(\beta-1)}\int_0^a x^{-1/(\beta-1)}dx\right) \\
&= (1+o(1))\exp\left(-d\frac{\beta-2}{\beta-1}b^{-1/(\beta-1)}a^{(\beta-2)/(\beta-1)}\right)\,.
\end{aligned}$$

Thus, using Chernoff's bound, we obtain that a.a.s.

$$|B| = (1+o(1))n\int_a^1 \exp\left(-d\frac{\beta-2}{\beta-1}a^{(\beta-2)/(\beta-1)}x^{-1/(\beta-1)}\right) dx\,,$$

and that a.a.s.

$$|D| = |A\cup B| = an + (1+o(1))n\int_a^1 \exp\left(-d\frac{\beta-2}{\beta-1}a^{(\beta-2)/(\beta-1)}x^{-1/(\beta-1)}\right) dx\,.$$

Item (2) follows as the above estimate of $|D|$ holds for every $a \in (0,1)$. □

As the number of isolated nodes is a lower bound for the domination number of a graph, the proof of Theorem 3 shows that a.a.s. the domination number of random power law graphs is $\Theta(n)$. An analogous result was found in [7] for graphs generated by the preferential attachment model.

3 The Cop Number in $G(n,p)$ Random Graphs

The cop number of random graphs $G(n,p)$ for a constant $p \in (0,1)$ was first studied in [3], who proved Theorem 1. We now consider the cop number of

$G(n,p(n))$ when $p(n)$ is a function of n. We will abuse notation and refer to p rather than $p(n)$.

Wieland and Godbole [11] proved the following two-point concentration for the domination number of random graphs $G(n,p)$ for p approaching zero sufficiently slowly as $n \to \infty$. Let $\mathbb{L}n = \log_{\frac{1}{1-p}} n$, and define

$$f(p,n) = \lfloor \mathbb{L}n - \mathbb{L}((\mathbb{L}n)(\log n)) \rfloor + 2.$$

Theorem 4. *Let p_0 be the smallest p for which*

$$p^2/40 \geq [\log((\log^2 n)/p)]/\log n \tag{6}$$

holds. A.a.s. $G \in G(n,p)$ and $p \geq p_0(n)$ satisfies

$$f(p,n) - 1 \leq \gamma(G) \leq f(p,n).$$

In particular,

$$\gamma(G) = f(p,n)(1+o(1)).$$

We obtain a concentration result for the cop number of the random graphs $G(n,p)$ where p satisfies (6). Define

$$g(p,n) = \lfloor \mathbb{L}n - 2\mathbb{L}((\mathbb{L}n)(\log n)) \rfloor + 1.$$

Note that $g(p,n) \leq f(p,n)$, and $g(p,n) = f(p,n)(1+o(1))$.

Theorem 5. *For $G \in G(n,p)$ and $p \geq p_0$, where p_0 is the smallest p for which (6) holds, a.a.s.*

$$g(p,n) \leq c(G) \leq f(p,n).$$

In particular,

$$c(G) = f(s,n)(1+o(1)).$$

The proof will follow from Theorem 4 if we can establish the lower bound for cop number of $G(n,p)$. We need the following lemma.

Lemma 2. *Let $k = \mathbb{L}n - 2\mathbb{L}((\mathbb{L}n)(\log n))$. If*

$$p \geq d \log^2 n/\sqrt{n} \tag{7}$$

where $d > 1$ is a fixed constant not depending on n, then

$$\lim_{n\to\infty} (k+1)\log n + (n-k-1)\log(1-p(1-p)^k) = -\infty. \tag{8}$$

Proof. By an elementary but tedious analysis we have by (7) that

$$(n-k-1)\log(1-p)\log\left(1-p(1-p)^k\right) = \Omega\left(\log^4 n\right), \tag{9}$$

and

$$-(k+1)\log(1-p)\log n = O\left(\log^2 n\right). \tag{10}$$

By (9) and (10), we obtain that

$$\begin{aligned}&\lim_{n\to\infty}((k+1)\log n+(n-k-1)\log(1-p(1-p)^k))\\ &=\lim_{n\to\infty}\frac{(k+1)\log(1-p)\log n+(n-k-1)\log(1-p)\log(1-p(1-p)^k)}{\log(1-p)}\\ &=-\infty,\end{aligned}$$

as desired.

Proof of Theorem 5. Let $k=\mathbb{L}n-2\mathbb{L}((\mathbb{L}n)(\log n))$. Note that the probability that G is not $(1,\lfloor k\rfloor)$-e.c. is at most

$$f(n,k,p)=n^{\lfloor k\rfloor+1}(1-p(1-p)^{\lfloor k\rfloor})^{n-\lfloor k\rfloor-1}.$$

To show that

$$n^{\lfloor k\rfloor+1}(1-p(1-p)^{\lfloor k\rfloor})^{n-\lfloor k\rfloor-1}=o(1),$$

it suffices to show that

$$n^{k+1}(1-p(1-p)^k)^{n-k-1}=o(1). \tag{11}$$

Note that (6) implies (7). As (11) is equivalent to (8), the result follows by Lemma 2. □

We last consider the cop number of the random graphs $G(n,p)$ for p approaching zero very fast. For example, if $p=o(1/n^2)$, a.a.s. $G\in G(n,p)$ is empty. So in this range of p, a.a.s. the cop number of G is n. We now consider the case when $p=d/n$ for constant $d\in(0,1)$. Bollobás [4] proved the following result.

Theorem 6. *Let $0<d<1$, $p=d/n$, and let X be the number of tree connected components of $G(n,p)$. Then the expectation of X is*

$$\mathbb{E}(X)=u(d)n+O(1),$$

where

$$u(d)=\frac{1}{d}\sum_{k=1}^{\infty}\frac{k^{k-2}}{k!}(de^{-d})^k.$$

A.a.s. $G(n,p)$ satisfies

$$|X|=u(d)n(1+o(1)).$$

We note that $u(d)\in(0,1)$. A graph is *unicyclic* if it contains exactly one cycle.

Theorem 7. *Let $0<d<1$ and $p=d/n$. Then a.a.s. $G\in G(n,p)$ is such that every connected component is a tree or a unicyclic graph, and there are at most $\log\log n$ vertices in the unicyclic components.*

Trees are cop-win graphs, while unicyclic graphs have cop number at most 2. Each tree component requires exactly one cop, while there are at most $2 \log \log n$ many cops needed for all the unicyclic components. Hence, the number of cops on the unicyclic components becomes negligible in contrast to the number of cops on tree components. Therefore, from Theorems 6 and 7 we have the following concentration result.

Corollary 1. *Let $0 < d < 1$, $p = d/n$. Then for the graph $G \in G(n,p)$,*

$$\mathbb{E}(c(G)) = u(d)n + O(\log \log n).$$

A.a.s. $G \in G(n,p)$ satisfies

$$c(G) = u(d)n(1 + o(1)).$$

Concentration results for the cop number of $G(n,p)$ with p in other ranges (such as just after the phase transition $p \sim c/n$ with $c > 1$) remain open.

Acknowledgements

We would like to thank David Vaughan for discussions on the integrals in Theorem 3.

References

1. Aigner, M., Fromme, M.: A game of cops and robbers. Discrete Applied Mathematics 8, 1–12 (1984)
2. Alspach, B.: Sweeping and searching in graphs: a brief survey. Matematiche 59, 5–37 (2006)
3. Bonato, A., Hahn, G., Wang, C.: The cop density of a graph. Contributions to Discrete Mathematics (accepted)
4. Bollobás, B.: Random Graphs. Cambridge University Press, Cambridge (2001)
5. Chung, F.R.K., Lu, L.: The average distance in a random graph with given expected degrees. Internet Mathematics 1, 91–114 (2006)
6. Chung, F.R.K., Lu, L.: Complex graphs and networks. American Mathematical Society, U.S.A. (2006)
7. Cooper, C., Klasing, R., Zito, M.: Lower bounds and algorithms for dominating sets in web graphs. Internet Mathematics 2, 275–300 (2005)
8. Janson, S., Łuczak, T., Ruciński, A.: Random Graphs. Wiley, New York (2000)
9. Nowakowski, R., Winkler, P.: Vertex to vertex pursuit in a graph. Discrete Mathematics 43, 230–239 (1983)
10. Quilliot, A.: Jeux et pointes fixes sur les graphes, Ph.D. Dissertation, Université de Paris VI (1978)
11. Wieland, B., Godbole, A.P.: On the domination number of a random graph. The Electronic Journal of Combinatorics 8 #R37 (2001)

Preemptive Scheduling on Selfish Machines

Leah Epstein[1] and Rob van Stee[2]

[1] Department of Mathematics, University of Haifa, 31905 Haifa, Israel
lea@math.haifa.ac.il
[2] Department of Computer Science, University of Karlsruhe, D-76128 Karlsruhe, Germany
vanstee@ira.uka.de

Abstract. We consider the problem of scheduling on parallel uniformly related machines, where preemptions are allowed and the machines are controlled by selfish agents. Our goal is to minimize the makespan, whereas the goal of the agents is to maximize their profit. We show that a known algorithm is monotone and can therefore be used to create a truthful mechanism for this problem which achieves the optimal makespan. We extend this result for additional common goal functions.

1 Introduction

Internet users and service providers act selfishly and spontaneously, without an authority that monitors and regulates network operation in order to achieve some social optimum such as minimum total delay. Selfish behavior may affect the performance, and it is interesting to identify the problems in which this happens, and to find how much performance can be lost as a result of lack of coordination. Many algorithmic problems, in which we investigate the cost of the lack of coordination arise. The study of lack of coordination can be compared to the lack of information (that is assumed in online algorithms) or the lack of unbounded computational resources (assumed in polynomial time approximation algorithms).

There has been a large amount of previous research into approximation and online algorithms for a wide variety of computational problems, but most of this research has focused on developing good algorithms for problems under the implicit assumption that the algorithm can make definitive decisions which are always carried out. On the internet, this assumption is no longer valid, since there is no central controlling agency.

To solve problems which occur, e.g., to utilize bandwidth efficiently (according to some measure), we now not only need to deal with an allocation problem which might be hard enough to solve in itself, but also with the fact that the entities that we are dealing with (e.g. agents that wish to move traffic from one point to the other) do not necessarily follow our orders but instead are much more likely to act selfishly in an attempt to optimize their private return (e.g. minimize their latency).

J. Janssen and P. Prałat (Eds.): CAAN 2007, LNCS 4852, pp. 57–70, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Mechanism design is a classical area of research with many results. Typically, the fundamental idea of mechanism design is to design a game in such a way that truth telling is a dominant strategy for the agents: it maximizes the profit for each agent individually. That is, each agent has some private data that we have no way of finding out, but by designing our game properly we can induce them to tell us what that is (out of well-understood self-interest), thus allowing us to optimize some objective while relying on the truthfulness of the data that we have. This is done by introducing *side payments* for the agents. In a way, we reward them (at some cost to us) for telling us the truth. The role of the mechanism is to collect the claimed private data (bids), and based on these bids to provide a solution that optimizes the desired objective, and hand out payments to the agents. The agents know the mechanism and are computationally unbounded in maximizing their utility.

A seminal paper by Archer and Tardos [4] considered the general problem of one-parameter agents. The class of one-parameter agents contain problems where any agent i has a private value t_i and his valuation function has the form $w_i \cdot t_i$, where w_i is the work assigned to agent i. Each agent i makes a bid b_i depending on its private value and the mechanism, and each agent wants to maximize its own profit.

The paper [4] shows that in order to achieve a truthful mechanism for such problems, it is necessary and sufficient to design a *monotone* algorithm, and use a payment function of the form

$$P_i(b_{-i}, b_i) = h_i(b_{-i}) + b_i w_i(b_{-i}, b_i) - \int_0^{b_i} w_i(b_{-i}, u)\, du. \tag{1}$$

Here (b_{-i}, x) is the bid vector b where the element b_i has been replaced by x, the h_i are arbitrary functions, and $w_i(b_{-i}, x)$ is the work assigned to agent i if the bid vector is (b_{-i}, x).

An algorithm is monotone if for every agent, the amount of work assigned to it does not increase if its bid increases. More formally, an algorithm is monotone if given two vectors of length m, b, b' which represent a set of m bids, which differ only in one component i, i.e., $b_i > b'_i$, and for $j \neq i$, $b_j = b'_j$, then the total size of the jobs (the work) that machine i gets from the algorithm if the bid vector is b is never higher than if the bid vector is b'.

Using this result, monotone (and therefore truthful) approximation algorithms were designed for several classical problems, like scheduling on related machines to minimize the makespan, where the bid of a machine is the inverse of its speed [4,2,6,1,13], shortest path [5,9], set cover and facility location games [7], and combinatorial auctions [14,16,3].

Problem definition. In this paper, we consider the problem of scheduling jobs in a multiprocessor setting where jobs may be preempted, and where the performance measure is the makespan. The makespan of a given schedule is the time at which the last task finishes.

Preemption means that a job may be split into parts, which can be possibly assigned to distinct machines. A part of job of size p must be assigned to a

time slot on one of the machines. The length of the time slot should be $\frac{p}{s}$ for a machine of speed s. The time slots assigned to the parts of one job on the different machines must all be disjoint.

We denote the number of processors by m and the number of jobs by n. We consider the version of this problem where the machines are related: each machine has a speed at which it runs, which does not depend on the job being run.

Denote the size of job j by p_j $(j = 1, \ldots, n)$. Denote the speed of machine i by s_i $(i = 1, \ldots, m)$. In our model, each machine belongs to a selfish user. The private value (t_i) of user i is equal to $1/s_i$, that is, the cost of doing one unit of work. The load on machine i, L_i, is the total size of the jobs assigned to machine i divided by s_i. The total work assigned to a machine i, denoted by W_i, is the total size of jobs assigned to it, i.e., $W_i = s_i \cdot L_i$. The profit of user i is $P_i - L_i$, where P_i is the payment to user i by the payment scheme defined by (1).

Our goal is to minimize the makespan. The classical version of this problem can be solved in polynomial time [12,11,17,8]. As is generally the case in algorithmic mechanism design, we are not interested in maximizing the total profit of the users.

As mentioned above, in order to imply a truthful mechanism, we need to show an algorithm for which an increase in a speed of a machine does not reduce the amount of work it receives.

Our results. We show that the algorithm given by Epstein and Tassa[10] is in fact monotone and can therefore also be used in this setting. We describe the algorithm which computes *the load* of every machine. The algorithm which creates the actual assignment is omitted, since we are only interested in the loads of machines and not in the exact assignment. This algorithm can be found in [10] as well.

Note that even though in principle idle time is allowed, the algorithm does not create idle time on any machine. The algorithm actually creates a *strongly optimal* schedule, in the sense that not only the maximum load is minimized, but also every subsequent load is minimized after the larger loads have been fixed. We thus show that it is possible to achieve an optimal makespan even with selfish agents.

We extend this result to preemptive scheduling with the goal of minimizing the ℓ_p norm of the loads vector, for $1 \leq p < \infty$. This is again done by using the algorithm of [10] for the cases $1 < p < \infty$, and a simple algorithm for $p = 1$.

Throughout the paper, we assume that the jobs are sorted in order of non-increasing size $(p_1 \geq p_2 \geq \ldots \geq p_n)$, and the machines are sorted in a fixed order of non-decreasing bids (i.e. non-increasing speeds, assuming the machine agents are truthful, $s_1 \geq s_2 \geq \ldots \geq s_m$). In case of ties, i.e., machines of identical speeds, each machine also carries an identifier (a number in $\{1, \ldots, m\}$), and a set of machines with the same speed are ordered in a increasing order of identifiers. We call the order implied by the identifiers a *lexicographical ordering* of the machines.

2 Makespan Minimization (ℓ_∞)

2.1 Algorithm

The algorithm is based on the following m lower bounds on the optimal makespan, given already by Liu and Yang[15] for a special case.

- For $k = 1, \ldots, m-1$,
$$\sum_{i=1}^{k} p_i / \sum_{i=1}^{k} s_i \ ,$$
That is, the total size of the k largest jobs, divided by the sum of k largest speeds.
- The last lower bound is
$$\sum_{i=1}^{n} p_i / \sum_{i=1}^{m} s_i \ ,$$
That is, the total size of all the jobs, divided by the sum of the speeds.

It is known that the maximum of all these bounds equals the optimal makespan [12,11,17,8]. The algorithm below is presented in [10]. The algorithm repeatedly executes the following steps until all jobs are assigned. Based on the bounds above, it determines a value k (an index of a machine) which determines the smallest maximum load that can be achieved for the remaining machines. If $k = m$, it assigns the jobs to the machines so that the load on all machines is equal and halts. Otherwise, it assigns the k largest jobs to k fastest machines (this set of machines will be called a *group*).

We use the following notations.

$$P_k = \begin{cases} \sum_{j=1}^{k} p_j & 1 \le k \le m-1 \\ \\ \sum_{j=1}^{n} p_j & k = m \end{cases} \ ,$$

and

$$S[a:b] = \sum_{i=a}^{b} s_i .$$

Algorithm 1

1. *Set $t = 0$ and $k_t = 0$ (at each stage k_t equals the number of values W_j that were already determined).*
2. *For every $k_t + 1 \le k \le m$, compute*
$$q_k = \frac{P_k - P_{k_t}}{S[k_t + 1 : k]} \ ,$$
and set k_{t+1} to be the (minimal) value of k for which q_k is maximal. The set of machines $\{k_t + 1, \ldots, k_{t+1}\}$ is defined to be the $t+1$-th group.

3. *For all* $k_t + 1 \leq j \leq k_{t+1}$, *set*

$$W_j = s_j \cdot \frac{P_{k_{t+1}} - P_{k_t}}{S[k_t + 1 : k_{t+1}]} .$$

4. *If* $k_{t+1} < m$ *set* $t = t + 1$ *and go to Step* 2.

This algorithm is optimal. In our analysis, we will use the following property.

Lemma 1. *[10] The loads are monotonically non-increasing as a function of the machine indices. Within a group, loads are identical.*

The following corollary holds since machines are sorted by speed.

Corollary 1. *The work assigned to machines is a monotonically non-increasing function of the machine indices.*

Proof. Consider machines i and $i+1$ in the sorted list, for some $1 \leq i \leq m-1$. By Lemma 1, $L_i \geq L_{i+1}$. We have $s_i \geq s_{i+1}$ and so

$$W_i = s_i \cdot L_i \geq s_{i+1} L_{i+1} = W_{i+1}.$$

□

2.2 Monotonicity

In this section, we prove the following theorem.

Theorem 2. *Algorithm 1 is monotone.*

We number the groups in the order of creation by the algorithm. We use additional notations. Let $s_a(b)$ be the speed of the a-th machine in group b, we use (a, b) to denote this machine. Let $S_a(b)$ be the sum of the a largest speeds among machines in groups $b, b+1, \ldots$. If group b consists of at least a machines, then this is actually the sum of a largest speeds of machines in this group. Let $p_a(b)$ be the size of the a-th largest job remaining after the first $b-1$ steps of the algorithm, and let $P_a(b)$ be the total size of the a largest such jobs.

We consider the situation where one machine (the j-th machine of group g) becomes faster, that is, decreases its bid. We denote the new speed of this machine by $s'_j(g)$ and we let

$$\varepsilon = s'_j(g) - s_j(g) > 0.$$

We use $S'_a(b)$ to denote the sum of the a largest speeds in groups $b, b+1, \ldots$, after the speed change. All other bids remain unchanged. We call the instance with the original speed the *original instance*, and the instance with the changed speed the *new instance* or the *modified instance*.

The following lemmas reduces the number of cases to be considered.

Lemma 2. *Consider machine (j, g) of the original instance that changes its speed. If the new location of this machine is later than the machines of groups $1, \ldots, h$ of the original instance (for some $h \leq g-1$), the groups $1, \ldots, h$ created by the algorithm for the new instance consist of the same machines as created for the original instance.*

Proof. Assume by contradiction that the algorithm does not act in the same way on the first h groups, and let $1 \leq c \leq h$ be the first group that is different for the new instance compared to the original instance. Let k be the number of machines in group c for the original instance and let k' be the number of machines for the new instance, where $k \neq k'$. Since the algorithm chose a group with k machines for the original instance, we have

$$\frac{P_k(c)}{S_k(c)} > \frac{P_{k'}(c)}{S_{k'}(c)}$$

(since the algorithm chooses a group of minimal number of machines in case of ties). If $k' < k$, by the assumption above, the machines of groups $1, \ldots, c$ of the original instance are earlier in the ordering than the machines which changes its speed. Therefore, the k machines of group c of the original instance remain in the same location in the ordering and there is no change in the speeds of any machines of group c for both instances, and so $S_{k'}(c) = S'_{k'}(c)$. This derives an immediate contradiction since we get

$$\frac{P_k(c)}{S'_k(c)} = \frac{P_k(c)}{S_k(c)} > \frac{P_{k'}(c)}{S_{k'}(c)} \geq \frac{P_{k'}(c)}{S'_{k'}(c)},$$

which would imply our algorithm makes a group of size k instead of k'. Otherwise, if $k' > k$, we have $S_k(c) = S'_k(c)$ and $S_{k'}(c) \leq S'_{k'}(c)$, and the contradiction is derived similarly. □

Below we consider the cases where a machine that increases its speed either does not change its location in the sorted list of machines, or changes places with its predecessor in the sorted list. The following lemma shows that these cases are sufficient to prove that the algorithm is monotone.

Lemma 3. *If there exists an instance for which a machine increases its speed and is allocated less work by the algorithm as a result, then there exists such an instance where as a result of the speed change the machine does not change its location in the sorted list, or appears just one place earlier.*

Proof. Assume that there exists an instance which disproves monotonicity. We may assume that the machine which changes its speed moves to a location which is at least two places earlier in the sorted list, as a result.

We split the process of change in speed into several phases. Let z be the index of the machine which changes its speed in the ordering for the original instance and $z' < z$ its location for the new instance. A single phase consists of an increase of speed for a machine until it changes places with the machine before it. There

are $z - z'$ such phases, and thus we consider $z - z' + 1$ instances, starting with the original instance, and considering also every instance that results from each additional phase.

Since the machine that changed its speed gets a smaller amount of work, there must exist at least one phase in which its work decreases. The instances which is defined just before this phase, with the speed change that results in the instance just after this phase prove the claim. □

We therefore need to consider three cases. In the first two cases we assume that (j, g) is located in the ordering of the new instance later than all machines of groups $1, \ldots, g-1$ in the original instance.

Case 1.1. Machine (j, g) remains in the same group. By Lemma 2 this means that groups $1, \ldots, g-1$ remain unchanged as a result of the change in speed. We consider the case that group g contains machine j in both instances.

Let k denote the number of machines in group g for the original instance, and let k' be the number of machines in this group after the increase of speed. Note that all three cases $k < k'$, $k = k'$ and $k > k'$ are possible in principle.

The original work assigned to machine (j, g) is

$$s_j(g)\frac{P_k(g)}{S_k(g)} .$$

The work assigned to this machine after the speed change is

$$s'_j(g)\frac{P_{k'}(g)}{S'_{k'}(g)} .$$

We show that

$$\frac{P_{k'}(g)}{S'_{k'}(g)} \geq \frac{P_k(g)}{S'_k(g)}$$

holds. This statement is trivial in case $k = k'$. For $k \neq k'$ it follows from the choice of the algorithm to create a group of k' machines and not of k machines. Therefore,

$$s'_j(g)\frac{P_{k'}(g)}{S'_{k'}(g)} \geq s'_j(g)\frac{P_k(g)}{S'_k(g)} .$$

We have that

$$s'_j(g)\frac{P_k(g)}{S'_k(g)} \geq s_j(g)\frac{P_k(g)}{S_k(g)},$$

since

$$\frac{s_j(g)+\varepsilon}{(S_k(g)+\varepsilon} \geq \frac{s_j(g)}{S_k(g)} \text{ for } s_j(g) \leq S_k(g),$$

which obviously holds because $S_k(g) = \sum_{i=1}^{k} s_i(g)$ and j belongs to the g-th group, i.e., $1 \leq j \leq k$.

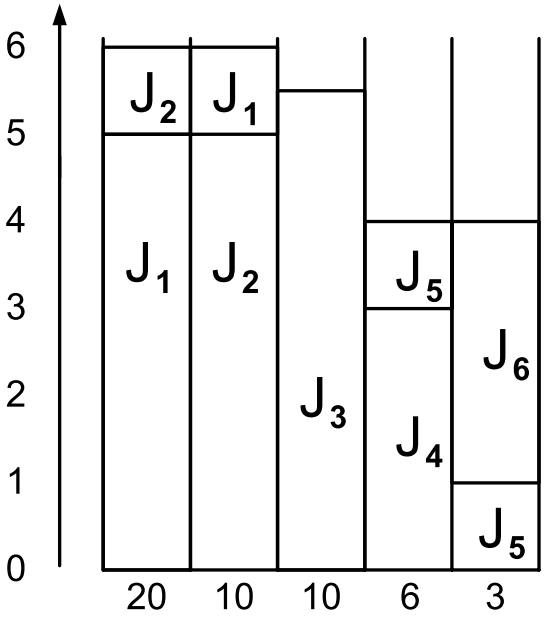


Fig. 1. The output for the original input

Case 1.2. We now consider the case where machine (j, g) does not remain in the same group. By Lemma 2 groups $1, \ldots, g-1$ still remain unchanged as a result of the change in speed, but group g changes in a way that it does not contain machine (j, g) of the original instance. This machine becomes a part of a later group $c > g$.

This situation is possible. The increase of a speed of a machine in the group has the following effect. The lower bounds for the maximum load decrease starting from this machine. The bound that determined the last machine (k) of the group g may no longer be a maximum. In this case, one or more new groups are formed before the one that contains j. Since we assume that the machine which changed its speed is located in the ordering of the new instance later than all machines of groups $1, \ldots, g-1$ in the original instance, this situation can only mean that the length of the g-th group has decreased, since in the new ordering, machine (j, g) cannot appear later than in the original ordering (by the definition of the sorting, where ties are broken in a consistent way).

For an example, see Figures 1, 2 and 3. The set of jobs is $\{J_1, \ldots, J_6\}$, and their sizes are 110, 70, 55, 18, 9 and 9, respectively. The output of the algorithm on the original set of speeds (20, 10, 10, 6 and 3) is given in Figure 1. Figure 2 shows the change in the output after the second machine increases its speed to 15. Figure 3 shows the change in the output after an additional change of speed, in which the fifth machine increases its speed to 3.

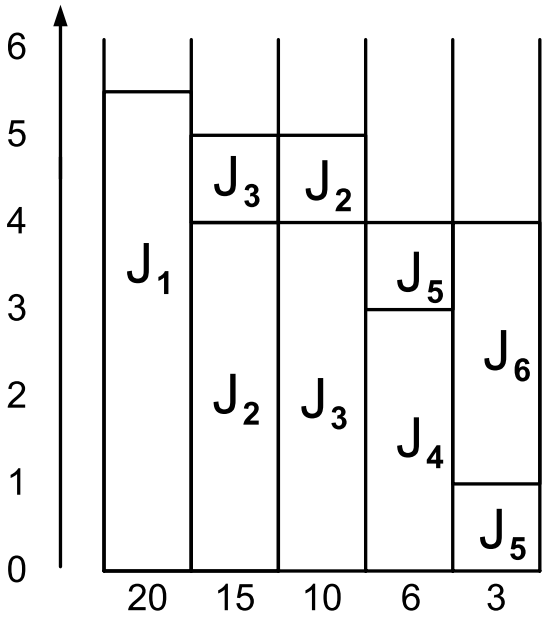


Fig. 2. The output after the second machine changes its speed

Note that we may assume that group g is the first group. Since for both instances the algorithm creates the same groups $1, \ldots, g-1$, running the same jobs, we can omit these machines and jobs from the instance. Thus we assume that machine 1 is the first machine of group g, and number the machines starting from the first machine of group g before the speed change, and the jobs excluding the jobs that are scheduled on the machines of groups $1, \ldots, g-1$.

Let k' be the last machine of the last group before (j, g) (in its new location) after the speed change. Let k'' be the last machine of the group which contains (j, g) after the speed change.

Case 1.2.a: Machine (j, g) remains in the same location in the ordering.
Given the original instance (without the jobs we omitted as described above), denote the total size of the k' largest jobs by L, and the remaining total size of jobs assigned to machines $1 \ldots, k$ by M (that is the difference between the size of all jobs assigned to these machines, and L). Denote the total size assigned to machines $k'+1, \ldots, k''$ after the speed change by N.

We define similarly the total speeds of these three groups of machines as S, T, and U, where S is the total speed of machines $1, \ldots, k'$, T is the total speed of machines $k'+1, \ldots, k$, and N is the total speed of machines $k'+1, \ldots, k''$. We denote the speed of the machine which changes its speed by s and the new speed by $s' = s + \varepsilon$.

We clearly have $T \geq s$, since the machine of this speed is in the set $\{k'+1, \ldots, k\}$. We also let $T' = T + \varepsilon$ denote the total speed of machines in

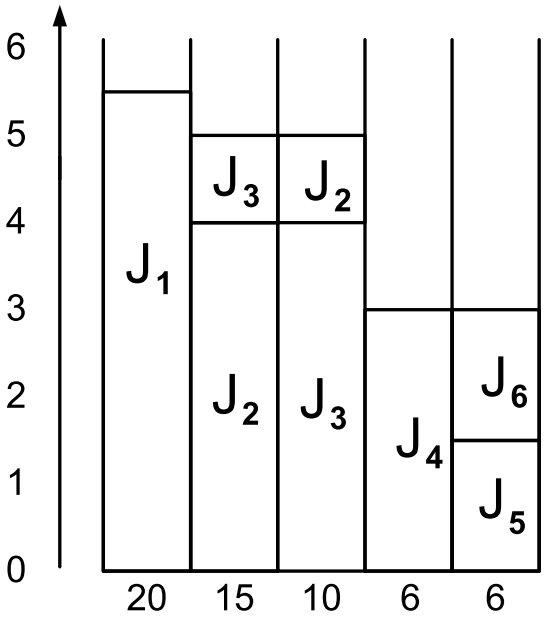


Fig. 3. The output after the fifth machine changes its speed as well

$\{k'+1,\ldots,k\}$ after the speed change, and by $U' = U + \varepsilon$ the total speed of machines in $\{k'+1,\ldots,k''\}$.

Since the algorithm chose a group g with machines $1,\ldots,k$ for the original instance, and not $1,\ldots,k'$, we have

$$\frac{L}{S} \leq \frac{L+M}{S+T}\,,$$

implying

$$LT \leq MS. \tag{2}$$

After the speed change, when the algorithm examines the set of machines $k'+1,\ldots,m$, it chooses the index k'' rather than k (although it may be that $k''=k$), so we have

$$\frac{N}{U+\varepsilon} \geq \frac{M}{T+\varepsilon}\,. \tag{3}$$

The work assigned to the machine which changes its speed, before the change, is $\frac{s(L+M)}{S+T}$, and after the change, it is $\frac{s'N}{U'}$, thus we would like to show

$$\frac{s(L+M)}{S+T} \leq \frac{s'N}{U'}\,.$$

Using (3), it is enough to show that

$$\frac{(s+\varepsilon)M}{T+\varepsilon} \geq \frac{s(L+M)}{S+T}\,.$$

This holds when

$$\begin{aligned} Ms(S+T)+M\varepsilon(S+T) &\geq s(L+M)(T+\varepsilon) \\ &= (L+M)sT+(L+M)s\varepsilon, \end{aligned}$$

or

$$s(MS-LT)+\varepsilon(M(S+T)-(L+M)s)\geq 0$$

which holds because $MS \geq LT$ and

$$M(S+T)-(L+M)s \geq M(S+T)-(L+M)T,$$

so

$$M(S+T)-(L+M)s \geq 0$$

holds when

$$M(S+T)\geq T(L+M),$$

or

$$MS \geq LT,$$

which is true by (2).

Case 1.2.b: The location of the machine that changed it speed (for the new instance) is one place before its location for the original instance. In this case we again assume that the speed changes gradually, and split the change in speed into three parts, the increase before the change of location, the swap, and an additional increase. We only need to consider the swap, which happens when the pair of machines have the same speed, or just after that. By Corollary 1 we have that the swap can only increase the work of the machine that becomes faster.

Case 2. Machine j is now a part of the previous group. By Lemma 2, this can only happen if the speed of machine j becomes larger or equal to the speed of the slowest machine in the group $g-1$ of the original instance, and j changes its location in the sorted list of machines. By Lemma 3 we need to consider the case where it moves to a location that is just before its previous location.

We consider a process in which the speed of the machine increases gradually, and partition the speed increase into before the swap, the swap, and after the swap. Denote the two locations that we consider by $i, i+1$.

The proofs of cases 1.1 and 1.2 covers the speed changes before and after the swap, thus we only need to consider the swap, which takes place when the two machines have the same speed, or just after that. This means that the machines changed roles, and thus the machine that used to be in location $i+1$ now gets the work $W_i \geq W_{i+1}$, by Corollary 1.

3 Other Norms

We start with the simple case of the ℓ_1 norm. In this case it is noted in [10] that the jobs are assigned to the set of fastest machines, that is, let b be a

maximal index such that $s_1 = \ldots = s_b$, then all the jobs are assigned to the first b machines. We use a specific variant of this algorithm that assigns all jobs to machine 1. It is straightforward to see the following. We call this algorithm *Algorithm 2*.

Proposition 1. *Algorithm 2 is monotone.*

We next consider the other cases ($1 < p < \infty$). We use the terminology of [10] and define

$$S_p[a:b] = \sum_{i=a}^{b} s_i^{\frac{p}{p-1}}.$$

The algorithm works as follows.

Algorithm 3

1. *Set $t = 0$ and $k_t = 0$ (at each stage k_t equals the number of values W_j that were already determined).*
2. *For every $k_t + 1 \leq k \leq m$, compute*

$$q_k = \frac{P_k - P_{k_t}}{S_p[k_t + 1 : k]},$$

   and set k_{t+1} to be the (minimal) value of k for which q_k is maximal. The set of machines $\{k_t + 1, \ldots, k_{t+1}\}$ is defined to be the $t+1$-th group.
3. *For all $k_t + 1 \leq j \leq k_{t+1}$, set*

$$W_j = s_j^{p/(p-1)} \cdot \frac{P_{k_{t+1}} - P_{k_t}}{S_p[k_t + 1 : k_{t+1}]}.$$

4. *If $k_{t+1} < m$ set $t = t + 1$ and go to Step 2.*

It can be seen that the algorithm acts in the same way that Algorithm 1 would work if the set of speeds were $s_1^{\frac{p}{p-1}}, \ldots, s_m^{\frac{p}{p-1}}$. Since we are not interested in the exact schedule but only in the work that each machine receives, in order to reduce to the proof in the previous section, we only need the following fact, that follows from $p > 1$.

Fact 1. *For a pair of speeds $s, s' > 0$, and $1 < p < \infty$ we always have $s' > s$ if and only if $s'^{\frac{p}{p-1}} > s^{\frac{p}{p-1}}$.*

We can thus prove the following.

Theorem 4. *Algorithm 3 is monotone.*

Proof. As mentioned above, Algorithm 3 simply runs algorithm 1 with adapted speeds. Thus we need to show that an increase of a speed s occurs if and only if an increase in "speed" $s^{\frac{p}{p-1}}$ occurs. This follows from Fact 1. □

4 Conclusion

We have shown that for a class of preemptive scheduling problems, it is possible to obtain truthful mechanisms simply by applying previously known algorithms. This is usually not the case for non-preemptive problems, which are typically NP-hard, whereas polynomial time approximation schemes lack the structure of optimal solutions. Note also that for makespan minimization in non-preemptive scheduling, if running time is not limited, it is possible to find an optimal algorithm which is monotone as follows. The machines are ordered by their lexicographical ordering. Given this ordering, the algorithm chooses an optimal schedule which has a smallest load vector (lexicographically) [4].

References

1. Andelman, N., Azar, Y., Sorani, M.: Truthful approximation mechanisms for scheduling selfish related machines. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 69–82. Springer, Heidelberg (2005)
2. Archer, A.: Mechanisms for Discrete Optimization with Rational Agents. PhD thesis, Cornell University (2004)
3. Archer, A., Papadimitriou, C., Talwar, K., Tardos, É.: An approximate truthful mechanism for combinatorial auctions with single parameter agents. In: Proc. of 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 205–214 (2003)
4. Archer, A., Tardos, É.: Truthful mechanisms for one-parameter agents. In: Proc. 42nd Annual Symposium on Foundations of Computer Science, pp. 482–491 (2001)
5. Archer, A., Tardos, É.: Frugal path mechanisms. In: Proc. of 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 991–999 (2002)
6. Auletta, V., De Prisco, R., Penna, P., Persiano, G.: Deterministic truthful approximation mechanisms for scheduling related machines. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 608–619. Springer, Heidelberg (2004)
7. Devanur, N.R., Mihail, M., Vazirani, V.V.: Strategyproof cost-sharing mechanisms for set cover and facility location games. In: ACM Conference on E-commerce, pp. 108–114 (2003)
8. Ebenlendr, T., Sgall, J.: Optimal and online preemptive scheduling on uniformly related machines. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 199–210. Springer, Heidelberg (2004)
9. Elkind, E., Sahai, A., Steiglitz, K.: Frugality in path auctions. In: Proc. of 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 701–709 (2004)
10. Epstein, L., Tassa, T.: Optimal preemptive scheduling for general target functions. Journal of Computer and System Sciences 72(1), 132–162 (2006)
11. Gonzalez, T., Sahni, S.: Preemptive scheduling of uniform processor systems. Journal of the ACM 25(1), 92–101 (1978)
12. Horvath, E.C., Lam, S., Sethi, R.: A level algorithm for preemptive scheduling. Journal of the ACM 24(1), 32–43 (1977)
13. Kovács, A.: Fast monotone 3-approximation algorithm for scheduling related machines. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 616–627. Springer, Heidelberg (2005)

14. Lehmann, D.J., O'Callaghan, L., Shoham, Y.: Truth revelation in rapid, approximately efficient combinatorial auctions. In: ACM Conference on Electronic Commerce, pp. 96–102 (1999)
15. Liu, J.W.S., Yang, A.-T.: Optimal scheduling of independent tasks on heterogeneous computing systems. In: Proceedings of the ACM National Conference, vol. 1, pp. 38–45 (1974)
16. Mu'alem, A., Nisan, N.: Truthful approximation mechanisms for restricted combinatorial auctions. In: Proc. of the 18th National Conference on Artificial Intelligence and 14th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI), pp. 379–384 (2002)
17. Shachnai, H., Tamir, T., Woeginger, G.J.: Minimizing makespan and preemption costs on a system of uniform machines. Algorithmica 42(3-4), 309–334 (2005)

Selfish Routing and Path Coloring in All-Optical Networks*

Ioannis Milis[1], Aris Pagourtzis[2], and Katerina Potika[2]

[1] Department of Computer Science
Athens University of Economics and Business, Greece
milis@aueb.gr
[2] School of Electrical and Computer Engineering
National Technical University of Athens, Greece
{pagour,epotik}@cs.ntua.gr

Abstract. We study routing and path coloring problems in all-optical networks as non-cooperative games. We especially focus on oblivious payment functions, that is, functions that charge a player according to her own strategy only.

We first strengthen a known relation between such games and online routing and path coloring. In particular, we show that the price of anarchy of such games is lower-bounded by, and in several cases precisely equal to, the competitive ratio of appropriate modifications of the First Fit algorithm.

Based on this framework we provide results for two classes of games in ring networks: in Selfish Routing and Path Coloring a player must determine both a routing and a coloring for her request, while in Selfish Path Coloring the routing is predetermined and only a coloring of requests needs to be specified. We prove specific upper and lower bounds on the price of anarchy of these games under various payment functions.

1 Introduction

In all-optical networks, communication requests are carried out by assigning to them a path in the network (routing) as well as a transmission wavelength. By using wavelength division multiplexing (WDM) it is possible to route several requests through the same link(s) of the network, and carry them out simultaneously by assigning a different wavelength to each request.

In this context, given a network topology and a set of communication requests, several interesting questions arise. If the routing of the requests is also given, the Path Coloring (PC) problem asks for the minimum number of colors (wavelengths) required such that requests sharing a common link are assigned different colors. If the routing of the requests is not given, the Routing and Path Coloring (RPC) problem asks for both a routing and a color assignment minimizing

* Research supported by "Pythagoras" grant of the Greek Ministry of Education, co-funded by the European Social Fund (75%) and National Resources (25%) — Operational Program for Educational and Vocational Training II (ΕΠΕΑΕΚ II).

J. Janssen and P. Prałat (Eds.): CAAN 2007, LNCS 4852, pp. 71–84, 2007.
© Springer-Verlag Berlin Heidelberg 2007

the number of colors under the same constraint. More optimization questions can be stated by introducing additional parameters and constraints. During the last decades a large body of work has been concentrated on the complexity and approximability questions for these optimization problems [1,2,3,4] (for a nice survey of early results see [5] and references therein).

A recent research direction concerns network optimization under game-theoretic criteria [6,7,8]. In such a context, an optimization problem can be modeled as a non-cooperative game of independent entities (players). These entities have their own objectives; they do not necessarily have to obey to a centralized protocol or they can manipulate this protocol (e.g. by providing false information) in order to achieve their own goals. The algorithmic game theory approach is used to optimize global objective functions taking into account the selfish behavior of the participating entities.

Following this direction we study the PC and RPC problems in all-optical networks as non-cooperative games. Each communication request is considered as a player and a payment function charges each player a cost depending on the (routing and color) choices of all players (including her own choices). Given a set of choices for all players we say that the game is in an equilibrium if no player can decrease her own cost by changing her choices. This equilibrium concept was first introduced by John Nash [9] and it is known as a Nash equilibrium. Although Nash has shown that each non-cooperative game has a mixed Nash equilibrium, the existence of a pure one is an open question for many games. Moreover, due to the selfish behavior of the players, such a pure equilibrium does not necessarily optimize a global objective goal. Such a goal is also known as social cost and for our problems can be defined as the number of colors used for (routing and) coloring a given set of requests. The global performance of Nash equilibria is measured by the Price of Anarchy (PoA) or coordination ratio which is defined as the ratio of the social cost of the worst Nash equilibrium over the optimal centralized solution [6], and reflects the loss in the global performance due to lack of coordination between players.

In this paper we study selfish PC and RPC in all-optical networks of ring topology; let us mention that, as far as we know, selfish PC has not been considered before. We first prove some general properties that further clarify the relation between selfish (R)PC and online (R)PC; the most important one is that the PoA of (R)PC under any *oblivious collision-free* payment function f is not smaller than the competitive ratio of a modification of the First-Fit algorithm that uses f as a selection criterion. (Note that the notion of oblivious collision-free payment function includes all functions that guarantee that no color collisions occur, but apart from that charge a player according to the player's own strategy only.) This property allows to obtain lower bounds on the PoA from lower bounds on the competitive ratio of First-Fit and its modifications; to the best of our knowledge no such lower bounds have been presented before for games in all-optical networks. We then study selfish PC and propose a payment function with PoA between 5.4 and 9. Finally, we propose two quite natural payment functions for selfish RPC. For the first of them, which forces players

to choose the smallest possible color, we show a tight upper bound for the PoA which is half the trivial upper bound $\frac{|R|}{OPT}$, where R is the given set of requests and OPT is the value of an optimal centralized solution for the corresponding RPC instance. For the second, which forces players to choose shortest path routing, we give an upper bound for its PoA which does not depend on the number of players but only (logarithmically) on the size of the network. Although a payment function with PoA bounded by a constant was already known [10] our payment functions are more natural.

The paper is organized as follows: In the next section we describe the formal model for our problems and the notation used in the paper, while in Section 3 we give a brief review of related work. In Section 4 we examine the relation between the solutions obtained by online and offline algorithms for PC and RPC and the Nash equilibria for the corresponding non-cooperative games. In Sections 5 and 6 we study selfish PC and RPC, respectively; we define payment functions yielding Nash equilibria and we present upper and lower bounds for the Price of Anarchy in both cases. We conclude in Section 7 by giving a brief comparison to earlier techniques and results.

2 Game Theoretic Model

We are given a network (graph) $G = (V, E)$ and a set of communication requests R. Each request r is a pair of nodes of G, i.e., $r = (x, y)$. When the routing of requests in R is also given in advance (pre-determined) we simply consider that a set of paths P is given instead of R. Therefore, an instance of the RPC problem is denoted by (G, R) and an instance of the PC problem, where players only have to choose a color for their paths, is denoted by (G, P).

In selfish RPC (selfish PC) on G each player i issues a request r_i (a path resp.). For simplicity, we identify a player with a request. A strategy σ_i for player i is a pair (p_i, c_i) (just (c_i) for selfish PC), where p_i is a simple path connecting the endpoints of r_i and c_i is a color assigned to p_i. Let S_i denote all possible strategies of player i. The possible strategies for each player are implicated by the topology of graph G and the number of colors allowed. If we restrict the number of colors to be no more than $|R|$ then there is a finite number of strategies for each player (we do not need to define them explicitly). There is also a payment function for each player i, that is: $f_i : S_1 \times \ldots \times S_{|R|} \to \mathbb{N}$. From now on, we will restrict our study to games where all players have the same payment function f.

Definition 1. *By* S-RPC *we denote the class of Selfish-RPC games, and a game in* S-RPC *with input graph G, set of requests R, and payment function f, is denoted by a triple (G, R, f).*

By S-PC *we denote the class of Selfish-PC games (pre-determined routing), and a game in* S-PC *with input graph G, set of routed requests P, and payment function f, is denoted by a triple (G, P, f).*

Given a class of graphs $\mathcal{G}$ and a payment function f, we denote by S-RPC $(\mathcal{G}, f)$ *(*S-PC$(\mathcal{G}, f)$*) the subclass of* S-RPC *(*S-PC *resp.) that consists of games (G, R, f) ((G, P, f) resp.) such that $G \in \mathcal{G}$.*

For a game (G, R, f) (and similarly for a game (G, P, f)) we define the following:

- A *pure strategy profile*, or simply strategy profile, is a vector $S = \{\sigma_1, \sigma_2, \ldots, \sigma_{|R|}\}$ of strategies, one for each player.
- A (pure) strategy profile is a *pure Nash Equilibrium (NE)* if for each player i it holds that

$$f(\sigma_1, \ldots, \sigma_i, \ldots, \sigma_{|R|}) \leq f(\sigma_1, \ldots, \sigma'_i, \ldots, \sigma_{|R|})$$

  for any strategy $\sigma'_i \in S_i$.
- The *social cost* $sc(S)$ of strategy profile S is the number of colors used for (routing and) coloring, if no color collisions appear; otherwise $sc(S) = \infty$.

Let OPT denote the optimum social cost for a game, that is, $OPT = \min_{S \in \mathcal{S}} sc(S)$, where $\mathcal{S}$ is the set of all possible strategy profiles. Note that OPT coincides with the cost of an optimal solution of the corresponding RPC (PC) instance.

The *price of anarchy* (PoA) of a game is the worst-case number of colors used in a NE (social cost) divided by OPT, that is,

$$\text{Price of Anarchy} = \frac{\max_{S \text{ is NE}} sc(S)}{OPT}.$$

The *price of stability* (PoS) of a game is the best-case number of colors used in a NE (social cost) over OPT, that is,

$$\text{Price of Stability} = \frac{\min_{S \text{ is NE}} sc(S)}{OPT}.$$

The price of anarchy (stability) of the class of games S-RPC$(\mathcal{G}, f)$ (S-PC$(\mathcal{G}, f)$) is the maximum price of anarchy (resp. stability) among all games in S-RPC$(\mathcal{G}, f)$ (resp. S-PC$(\mathcal{G}, f)$).

Definition 2. *We say that a payment function for a selfish (routing and) path coloring game is* oblivious collision-free *if:*

(a) *it guarantees that in a Nash Equilibrium no color collisions occur (by charging a very large amount to players that use the same color and share links of the network) and*

(b) *it charges a player (who does not collide with other players) according to the player's own strategy only.*

Let us observe that for any instance of S-RPC (S-PC) with oblivious collision-free payment function it holds that $sc(S) \leq |R|$ ($sc(S) \leq |P|$, resp.) if S is a NE; hence, $PoA \leq \frac{|R|}{OPT}$ ($PoA \leq \frac{|P|}{OPT}$, resp.). All functions considered in this paper are oblivious collision-free. For the sake of simplicity we will omit from the descriptions of our payment functions the condition that guarantees collision-free Nash Equilibria.

3 Previous Work

Bilò and Moscardelli [11] consider the existence and performance of Nash equilibria of selfish RPC games in all-optical networks. They study four possible payment functions. They show that only two of these payment functions, namely when each player pays for her own color and when she pays for the maximum color used by any other overlapping player, guarantee convergence to a pure NE. However, they prove that the PoA is as high as $|R|$ even for rings. In [10] they refine this result to $\frac{|R|}{OPT}$ for any payment function which is a non-decreasing function of the color of the player.

Bilò et al. [10] consider different information levels of local knowledge that players may have for computing their payments in selfish RPC games and give bounds for the PoA in chains, rings and trees. In the complete level of information each player knows all other players' routing and coloring strategies. In the intermediate level of information each player only knows which colors are used on any edge of the network and in the minimal level of information each player knows which colors are used only on edges along paths that the player can choose. For the complete level they prove that the PoA is the same as the best approximation ratio for RPC, thus 1 in chains and 2 in rings, under payment functions specifically constructed according to the corresponding algorithms. For the intermediate level they give a payment function specifically constructed according to Slusarek's algorithm [12] for online PC in rings (also known as online circular arc coloring) that results in a PoA that is $3 + O(\frac{\log L}{L})$ in chains and $6 + O(\frac{\log L}{L})$ in rings, where L is the maximum load. For the minimal level they prove that for any payment function which is a non-decreasing function of the color of the player, the PoA in chains is bounded by the competitive ratio (say FF_{chain}) of the First-Fit algorithm for online PC in chains and the PoA in trees is $O(\log |R|)$; they also give a payment function for rings with PoA bounded by $2 \cdot FF_{chain}$. Pemmaraju et al. [13] have recently shown that $FF_{chain} \leq 8$, therefore the ratios obtained in [10] are in fact 8 in chains and 16 in rings (instead of 25.72 and 51.44 originally mentioned).

The existence of Nash equilibria and the complexity of recognizing and computing a Nash equilibrium for selfish RPC under several payment functions are considered by Georgakopoulos et al. [14]. Their results indicate that recognizing a Nash equilibrium can be done in polynomial time, when each player pays for her own color, when she pays for the maximum color used by any other overlapping player and when she pays for the most loaded edge that she uses. On the other hand, when the player pays for all the different colors appearing along her path, recognizing a Nash equilibrium is NP-complete.

4 Solutions to PC and RPC as Nash Equilibria

In this section we explore the relation of the solutions obtained by online and offline algorithms for PC and RPC to Nash equilibria for S-PC and S-RPC with respect to various oblivious collision-free payment functions.

In the online version of RPC problem requests arrive as an ordered sequence $\langle R\rangle = \langle r_1, r_2, \ldots, r_{|R|}\rangle$. Such an online instance of RPC is denoted by $(G, \langle R\rangle)$. Upon arrival of a request r_i, an online algorithm should decide a path and a color assignment to r_i so that no color collisions appear on any edge of paths that are already colored (that is, corresponding to requests r_j with $j < i$); the algorithm has no knowledge of requests that are going to appear later (that is, requests r_j with $j > i$). The objective is to minimize the number of colors used. As before, an instance of online PC is denoted by $(G, \langle P\rangle)$, where $\langle P\rangle$ is a sequence of paths ordered by arrival time.

Probably the simplest online algorithm for PC is First-Fit, which colors each request r_i with the smallest available color, provided that no color collisions occur. We will also make use of the following version of First-Fit, which is appropriate for online RPC: the algorithm chooses a path and color for request r_i in such a way that no color collisions occur and the color assigned to r_i is the minimum possible.

We now define a useful generalization of First-Fit for RPC. Consider a cost function f which specifies a cost for each path and color assignment (p, c) to a request r_i, taking into account the path and color assignment to requests $r_j,\ j < i$. Then, First-Fit with criterion f ($FF(f)$ for short) assigns to each request r_i the path p and color c that minimize $f(r_i, p, c)$, breaking ties arbitrarily. For example, the standard First-Fit for RPC described above can be seen as $FF(f)$, where $f(r_i, p, c) = c$ if p does not overlap with any path of color c, otherwise $f(r, p, c) = \infty$. A similar generalization of First-Fit for PC is defined analogously by using cost functions that take into account only the color assignment to requests (since paths are given; in this case the payment function has two arguments p and c). Formally, in the above description for each payment function f, the path-color (or just color) assignment to requests $r_j,\ j < i$, should also appear as argument of function f; we will omit it here for the sake of simplicity.

The following two lemmata reveal an interesting relation between selfish routing and coloring and the corresponding online (centralized) problems. The second lemma is in fact a slight reformulation of an observation from [10].

Lemma 1. *Consider a game (G, R, f) in* S-RPC *(*S-PC*) where f is an oblivious collision-free payment function. For any ordering $\langle R\rangle$ of R, an execution of $FF(f)$ algorithm on $(G, \langle R\rangle)$ gives a strategy profile for (G, R, f) which is a Nash Equilibrium.*

Proof. Consider the path-color assignment obtained by an execution of $FF(f)$ on $(G, \langle R\rangle)$. A request r_i cannot be assigned a path-color combination of lower cost unilaterally, otherwise $FF(f)$ would have chosen that path-color combination for r_i. The reason is that if such a different assignment is possible then it does not cause color collisions with respect to the path-color assignment of all other requests. Therefore, it certainly does not cause any color collision with respect to requests $r_j, j < i$; hence, upon arrival of r_i, $FF(f)$ would have chosen this lower cost assignment. □

Lemma 2 ([10]). *Consider a game (G, R, f) in* S-RPC *(*S-PC*) where f is collision-free and non-decreasing on the players' color (hence also oblivious). For every strategy profile S that is a Nash Equilibrium for (G, R, f), there is an ordering $\langle R \rangle$ of R such that there is an execution of $FF(f)$ algorithm on $(G, \langle R \rangle)$ yielding the same path-color assignment to R as S.*

We now show how to convert any (routing and) coloring solution to RPC (PC) to a Nash Equilibrium for the corresponding game in S-RPC (S-PC resp.) with at most the same number of colors.

Lemma 3. *Let k be the number of colors used in a solution to instance (G, R) of* RPC *((G, P) of* PC *respectively). We can compute a strategy profile which is a Nash Equilibrium of social cost at most k for game (G, R, f) in* S-RPC *(*S-PC *respectively) where f is oblivious collision-free and a non decreasing function of the players' color.*

Proof. We convert the solution to instance (G, R) for RPC into a strategy profile which is a Nash Equilibrium for game (G, R, f) in S-RPC by using the Nash Conversion algorithm described below.

Algorithm 1. *Nash Conversion*

for each color $c := 1$ to k **do**
  for each request r colored with c **do**
    for each color $c' := 1$ to $c - 1$ **do**
      if there exists a path (including the current one) for request r that does not overlap with any other path colored with c'
      then { use that path to route r and color it with c'; **exit for** }

For PC the above algorithm works by modifying the "if" statement as follows: "if the path of r does not overlap with any path colored with c' then assign color c' to r".

Note that no request can move to a smaller color, because Algorithm 1 assigns the smallest available color, say c', to r and does not affect afterwards the path-color assignment of requests that have color smaller than c'. □

Combining the above lemmata we obtain the following theorem:

Theorem 1. *Let $\mathcal{G}$ be a class of graphs.*

1. *The price of anarchy for the class of games* S-RPC$(\mathcal{G}, f)$ *(*S-PC$(\mathcal{G}, f)$*), where f is oblivious collision-free, is at least as large as the competitive ratio of $FF(f)$ for* RPC *(*PC*, resp.) in graphs that belong to $\mathcal{G}$.*
2. *The price of anarchy for the class of games* S-RPC$(\mathcal{G}, f)$ *(*S-PC$(\mathcal{G}, f)$*), where f is oblivious collision-free and is a non-decreasing function of the players' color, is equal to the competitive ratio of First-Fit for* RPC *(*PC*, resp.) in graphs that belong to $\mathcal{G}$.*

3. *The price of stability for any game* (G, R, f) *in* S-RPC *(*S-PC*), where* f *is oblivious collision-free and is a non-decreasing function of the players' color, is equal to* 1.

Proof. 1: By Lemma 1, each execution of $FF(f)$ leads to a path-color assignment which is a NE for a game in S-RPC$(\mathcal{G}, f)$ (S-PC$(\mathcal{G}, f)$); the social cost of that NE is equal to the number of colors used by $FF(f)$. Dividing by OPT we get the claim.

2: Let S be a NE of the highest social cost. By Lemma 2, it turns out that there is an execution of $FF(f)$ on the corresponding RPC (PC) instance that requires the same number of colors as S. Dividing by OPT we get that the competitive ratio of $FF(f)$ is at least as large as the price of anarchy for S-RPC$(\mathcal{G}, f)$ (S-PC$(\mathcal{G}, f)$ resp.). Combining with 1 we get the claim.

3: It suffices to consider the optimal coloring and convert it to a NE by using the Nash Conversion algorithm. □

Using Theorem 1,2 and the fact that the competitive ratio of the First Fit algorithm for online PC in chains is between 4.4 and 8 [13], we have that:

Corollary 1. *The payment function* $f(p, c) = c$ *induces* S-PC *games in chains with a price of anarchy between* 4.4 *and* 8.

5 S-PC in Rings

In this section we propose and study an oblivious collision-free payment function that results in a relatively low PoA for S-PC in rings.

We first observe that the natural choice of taking as payment function the one that charges the color value gives $2.53 \log n + 5$ (n is the number of nodes) on the PoA for S-PC in rings. This is obtained by Theorem 1,2 and the competitive ratio of the First Fit algorithm for online PC in rings shown in [15].

Let L_e be the load on edge e, i.e. number of paths that use e. Let E' be a set of edges then we denote by $L_{E'}$ the maximum load over all edges in E'. Let L be the maximum load of G. Consider an arbitrary edge e of ring G. Payment function f_e is defined as follows: if a player p (recall that players can be seen as paths in this case) uses edge e then she is encouraged to use the smallest available color, since she pays the value of the color she uses; otherwise she is encouraged to use the smallest available color which is greater than L_e (for which she pays the color value) instead of using any color in $\{1, \ldots, L_e\}$ (for which she must pay a much higher price). Formally,

$$f_e(p, c) = x_{p,e} \times \lfloor \frac{L_e}{c} \rfloor \times |P| + c$$

where $x_{p,e} = \begin{cases} 0 \text{ , if } p \text{ traverses edge } e \\ 1 \text{ , otherwise} \end{cases}$

Recall that FF_{chain} denotes the competitive ratio of First-Fit for online PC in chains (online interval coloring).

Theorem 2. *The payment function f_e induces* S-PC *games in rings with a price of anarchy equal to $FF_{chain} + 1$.*

Proof. Observe that the graph $G-e$ is a chain. Payment function f_e implies that all players that have their path in path set $P_{G-e} = P \setminus P_e$ have no gain by using a color from the set $\{1, \ldots, L_e\}$, because it results in high cost (at least $|P|$). Therefore, a Nash equilibrium S can be seen as the result of two independent executions of First-Fit on players: (a) the first execution is on players (paths) that use e, with available colors $\{1, \ldots, L_e\}$, and (b) the second execution is on players that do not use e, with available colors $\{L_e + 1, \ldots, |P|\}$. Both subsets of players are ordered according to their color in S (increasingly).

For the first group of players L_e colors will be used, while for the second, of load L_{G-e}, First Fit will need at most $FF_{chain} \cdot L_{G-e}$ colors. Hence, the total number of colors in S will be

$$\begin{aligned} sc(S) &\leq L_e + FF_{chain} \cdot L_{G-e} \leq (FF_{chain} + 1) \max\{L_e, L_{G-e}\} \\ &\leq (FF_{chain} + 1)OPT \end{aligned}$$

because any algorithm will need at least as many colors as the maximum load of requests. Since no specific assumption was made for S, the above inequality holds for all NE implying that $PoA \leq FF_{chain} + 1$.

It is also possible to bound PoA from below by considering an instance where P_e consists of L requests and P_{G-e} consists of a worst-performance chain instance for First-Fit of load L. Assume that paths in P_e do not overlap paths in P_{G-e}. Then $OPT = L$. On the other hand, if we give this instance as input to $FF(f_e)$ algorithm, with the requests in P_{G-e} ordered as in the worst-performance instance of First-Fit, then $FF(f_e)$ will need $(FF_{chain} + 1)L$ colors. By Theorem 1.1 this implies that $PoA \geq FF_{chain} + 1$. □

Corollary 2. *The payment function f_e induces* S-PC *games in rings with a price of anarchy between* 5.4 *and* 9.

6 S-RPC in Rings

In this section we consider S-RPC in rings induced by two different oblivious collision-free payment functions. The first one forces players to choose the smallest possible color while the second one forces them to choose shortest path routing.

6.1 The Color-Length Payment Function

We consider the payment function $f(r, p, c) = c \cdot n + length(p)$, where n is the number of nodes in the ring. It is clear that under this function a player r always selects the smallest possible color even if it requires to follow the longest one of her two possible alternative paths.

Theorem 3. *The payment function function $f(r, p, c) = c \cdot n + length(p)$ induces* S-RPC *games in rings with a price of anarchy equal to $\frac{|R|}{2\ OPT} + 1$, where R is the given set of requests.*

Proof. We first prove that $PoA \leq \frac{|R|}{2\ OPT} + 1$.

Let S be a NE for a $(Ring, R, f)$ game. Let R_1 be the subset of requests assigned an exclusive color (assigned only to one of these requests) and R_2 be the subset of requests that share a color with at least one other request. It follows that

$$sc(S) \leq \frac{|R_2|}{2} + |R_1| = \frac{|R|}{2} + \frac{|R_1|}{2}.$$

We shall prove that $\frac{|R_1|}{2} \leq OPT$ and therefore

$$PoA = \frac{sc(S)}{OPT} \leq \frac{|R|}{2\ OPT} + 1.$$

Clearly the requests in R_1 are routed via paths that overlap each other, for otherwise at least two of them can take the same color. Moreover, each of them is routed via its shortest path for otherwise S would not be a NE for $(Ring, R, f)$. This is because such a request, routed via its longest path, can improve its own cost by choosing its shortest path and keeping its (unique) color. Hence, the requests in R_1 are routed via paths of length at most $n/2$.

Consider the optimal solution; it uses OPT colors. According to this coloring the set of requests, R, can be partitioned into OPT disjoint subsets $C_1, C_2, \ldots, C_{OPT}$, each one containing the requests assigned the same color. The requests in each $C_i,\ 1 \leq i \leq OPT$, are routed via non overlapping paths and hence they are consecutive in a clockwise traversal of the ring i.e., no request starts or ends between the start and the end point of any other. Therefore, they, but at most one, are routed via their shortest paths; that is, at most one of them is routed via a path of length greater than $n/2$.

Consider now the routing of the requests in R_1 in S and in the optimal solution. In both cases the routing of these requests coincides (shortest path routing) except for at most OPT requests i.e., the single requests that are possibly routed via longest paths in each set $C_i,\ 1 \leq i \leq OPT$. Since the requests in R_1 are routed via paths overlapping each other, it follows that at most two requests from each set C_i can be in R_1: the one that is routed via its longest path, now routed via its shortest path, and one of the rest. Therefore, $|R_1| \leq 2\ OPT$.

We prove next, by a counterexample, that $PoA \geq \frac{|R|}{2\ OPT} + 1$.

Consider the following instance: A ring of $2k+6t$ nodes and a set R of requests consisting of $k+2$ subsets $R_0, \ldots, R_{k+1}$, each containing t requests:

- R_0 consists of t 'crossing' requests $\{k+2t+(j-1), (2k+5t+j)\}, 1 \leq j \leq t$ (the last request is between node $k+3t-1$ and node $2k+6t$).
- $R_i, 1 \leq i \leq k$, consists of t identical requests $\{i, i+1\}$ (from node i to node $i+1$).
- R_{k+1} consists of t 'crossing' requests $\{k+j, k+j+t\}, 1 \leq j \leq t$.

The optimal solution routes R_1 to R_{k+1} with shortest paths and R_0 with longest paths, and assigns to each R_i colors $\{1, \ldots, t\}$ (see Figure 1). Thus $OPT = t$.

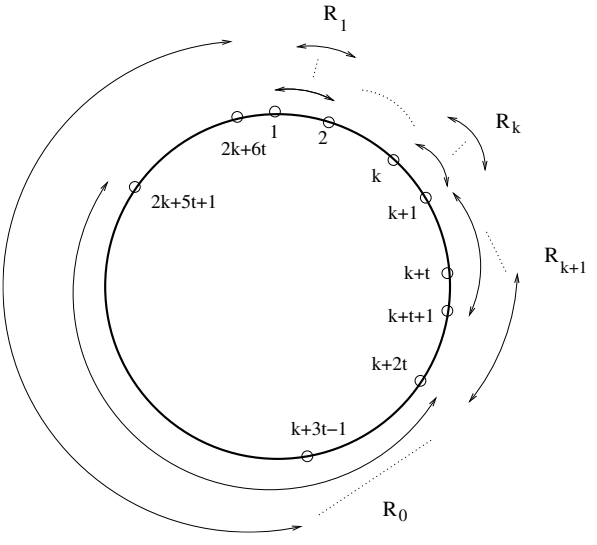


Fig. 1. Example of a S-RPC game and its optimal centralized solution

Consider now the execution of $FF(f)$ algorithm on instance $(Ring, \langle R \rangle)$ of online RPC, assuming that requests in R_i appear in $\langle R \rangle$ before requests in R_j for $i < j$. Then, $FF(f)$ first routes requests in R_0 via their shortest paths, that is, via $\{1, k + 2t\}$, and assigns them colors $\{1, \ldots, t\}$. Then, for each $R_i, 1 \leq i \leq k$, every two requests are routed via complementary paths and receive the same color; thus $t/2$ new colors are needed for each $R_i, 1 \leq i \leq k$ (see Figure 2). Finally, requests in R_{k+1} would overlap each other and every other previously considered request, no matter which of the two possible paths is used. Therefore t new colors are needed for requests R_{k+1} and the shortest path is chosen for each of them. Altogether, $FF(f)$ uses $\frac{kt}{2} + 2t$ colors. By Theorem 1, the number of colors used by $FF(f)$ is a lower bound of the social cost of the worst Nash equilibrium, that is

$$PoA \geq \frac{\frac{kt}{2} + 2t}{t} = \frac{(k+2)t}{2t} + 1 = \frac{|R|}{2OPT} + 1.$$

□

6.2 The Length-Color Payment Function

We consider the payment function $f(r, p, c) = length(p) \cdot |R| + c$, where R is the given set of requests. It is clear that under this function a player r always selects the shortest one of its two possible alternative paths even if it requires to take a larger color.

Theorem 4. *The payment function* $f(r, p, c) = length(p) \cdot |R| + c$ *induces* S-RPC *games in rings with a price of anarchy such that* $FF_{chain} + 1 \leq PoA \leq 5.06 \log n + 10$*, where* n *is the number of nodes in the ring.*

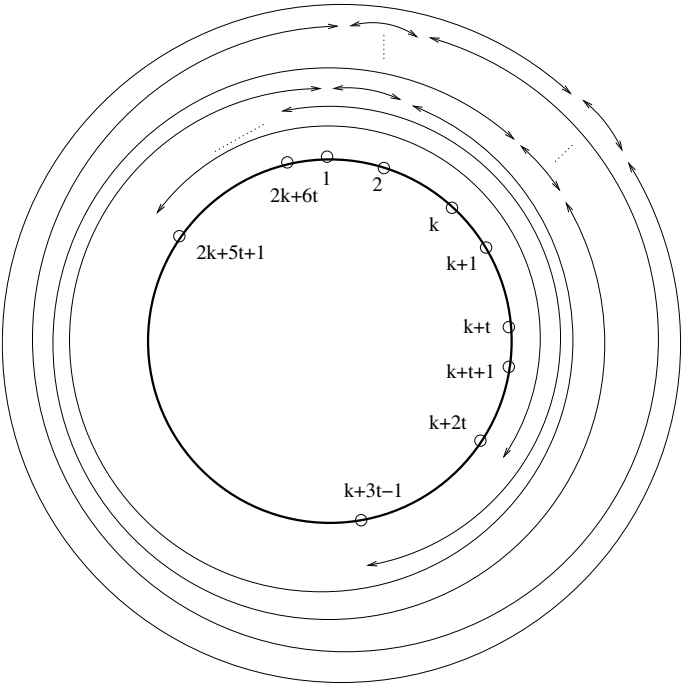


Fig. 2. Solution obtained by applying algorithm FF(color-length) on the instance of Figure 1 (only subsets R_i, $0 \leq i \leq k$, are shown, for the sake of clarity)

Proof. The proof of the upper bound is based on a result for dynamic wavelength assignment on rings in [15]. This result states that the First Fit algorithm needs at most $2.53L \log n + 5L$ wavelengths, where L is the maximum load on the ring. Combining this result with the observation that any shortest path routing produces a maximum load that is at most twice the one produced by an optimal routing we get the upper bound.

For the proof of the lower bound we consider the following S-RPC game: A ring of $2k$ nodes and a set R of requests consisting of 2 subsets R_1 and R_2 such that:

- R_1 consists of an arbitrary number of requests which, when routed via their shortest paths, yield a maximum load of L in the ring links. Moreover, for each request $(i, j) \in R_1$ it holds that $1 \leq i \neq j \leq a < k - 1$.
- R_2 consists of L identical requests $(1, i),\ k > i > a$.

The optimal solution routes R_1 via shortest and R_2 via longest paths. This way no request in R_1 overlaps with any request in R_2. Requests in R_1 require L colors, since they are on the chain $1, 2, \ldots, a$, and requests in R_2 can be colored by the same L colors. Therefore, $OPT = L$.

The $FF(f)$ online algorithm on instance $(Ring, \langle R \rangle)$ of RPC, if requests in R_1 appear in $\langle R \rangle$ before requests in R_2 (or vice versa), routes all requests via

shortest paths and therefore it uses $L \cdot FF_{chain}$ colors for R_1 and L new colors for R_2. Using Theorem 1.1 it follows that $PoA \geq FF_{chain} + 1$. □

7 Conclusions

In this paper we studied selfish (routing and) path coloring games in all-optical networks. We proposed a payment function for S-PC in rings with a PoA between 5.4 and 9. For S-RPC in rings we studied two natural payment functions: one called 'color-length' which favors smallest colors and one called 'length-color' which favors shortest paths. We have shown that the color-length function fails to achieve a low PoA; however, its PoA is half the PoA of any payment function that charges according to the value of the color only [10,11]. On the other hand, the length-color function achieves a PoA which does not depend on the number of requests but only on the number of nodes of the ring (logarithmically). It is still open whether the upper bound for the length-color function can be further improved taking into account that the lower bound we have shown is as low as 5.4. Note that all our functions require only local color information, namely to know which colors are used along edges that can be used by a player (minimal level of information using the classification in [10]).

Comparing to earlier work we observe that, as far as we know, S-PC has not been considered before. For S-RPC in rings, a payment function with $PoA \leq 16$ has been proposed in [10]; however, that payment function forces players to avoid routing through a particular edge of the graph, which may increase the total traffic of the network. Therefore, our length-color function might be more appropriate in cases where reducing the total traffic is important (e.g. if the social cost takes into account the sum of the loads over all edges).

In order to obtain our results we established a connection of the PoA of selfish (routing and) path coloring games to the competitive ratio of First-Fit-like algorithms for the corresponding online (routing and) path coloring problems. This connection is a generalized and strengthened form of an observation from [10]. In particular, the observation in [10] was used in order to obtain an upper bound on PoA from the competitive ratio of First-Fit. Our strengthening allows to obtain lower bounds as well.

References

1. Garey, M.R., Johnson, D.S., Miller, G.L., Papadimitriou, C.H.: The complexity of coloring circular arcs and chords. SIAM Journal on Algebraic and Discrete Methods 1(2), 216–227 (1980)
2. Karapetian, I.A.: Coloring of arc graphs (in russian). Akad. Nauk Armyan. SSR Dokl. 70(1), 306–311 (1980)
3. Raghavan, P., Upfal, E.: Efficient routing in all-optical networks. In: STOC. Proc. of the 26th Annual ACM Symposium on Theory of Computing, pp. 134–143 (1994)
4. Erlebach, T., Jansen, K., Kaklamanis, C., Mihail, M., Persiano, P.: Optimal wavelength routing on directed fiber trees. Theor. Comput. Sci. 221(1-2), 119–137 (1999)

5. Gargano, L., Vaccaro, U.: Routing in all-optical networks: Algorithmic and graph theoretic problems. Numbers, Information and Complexity, pp. 555–578. Kluwer Academic Publishers, Dordrecht (2000)
6. Koutsoupias, E., Papadimitriou, C.H.: Worst-case equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
7. Mavronicolas, M., Spirakis, P.G.: The price of selfish routing. In: STOC. Proc. of the 33rd Annual ACM Symposium on Theory of Computing, pp. 510–519 (2001)
8. Roughgarden, T., Tardos, É.: How bad is selfish routing? J. ACM 49(2), 236–259 (2002)
9. Nash, J.F.: Equilibrium points in n-person games. Proc. of the National Academy of Sciences of the United States of America 36(1), 48–49 (1950)
10. Bilò, V., Flammini, M., Moscardelli, L.: On Nash equilibria in non-cooperative all-optical networks. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 448–459. Springer, Heidelberg (2005)
11. Bilò, V., Moscardelli, L.: The price of anarchy in all-optical networks. In: Kralovic, R., Sýkora, O. (eds.) SIROCCO 2004. LNCS, vol. 3104, pp. 13–22. Springer, Heidelberg (2004)
12. Slusarek, M.: Optimal online coloring of circular arc graphs. Informatique Theoretique et Applications 29(5), 423–429 (1995)
13. Pemmaraju, S.V., Raman, R., Varadarajan, K.R.: Max-coloring and online coloring with bandwidths on interval graphs (manuscript, 2006)
14. Georgakopoulos, G.F., Kavvadias, D.J., Sioutis, L.G.: Nash equilibria in all-optical networks. In: Deng, X., Ye, Y. (eds.) WINE 2005. LNCS, vol. 3828, pp. 1033–1045. Springer, Heidelberg (2005)
15. Gerstel, O., Sasaki, G., Kutten, S., Ramaswami, R.: Worst-case analysis of dynamic wavelength allocation in optical networks. IEEE/ACM Transactions on Networking 7(6), 833–846 (1999)

A Worst-Case Time Upper Bound for Counting the Number of Independent Sets

Guillermo De Ita[1] and Aurelio López-López[2]

[1] Universidad Autónoma de Puebla
deita@cs.buap.mx
[2] INAOE - Tonantzintla, Pue. México
allopez@inaoep.mx

Abstract. The problem of counting the number of independent sets of a graph G (denoted as $NI(G)$) is a classic #P-complete problem for graphs of degree 3 or higher. Exploiting the strong relation between $NI(G)$ and Fibonacci numbers, we show that if the depth-first graph of G does not contain a pair of basic cycles with common edges, then $NI(G)$ can be computed in linear time (in the size of the graph). This determines new classes of instances of graphs without restrictions on their degrees and where the number of independent sets is computed in polynomial time.

We design an exact deterministic algorithm for computing $NI(G)$ based on the topological structure of the graph G, applying the well-known splitting rule from Davis and Putnam (D&P) procedure. D&P is a familiar method for solving the Satisfiability Boolean Problem. Our algorithm for computing $NI(G)$ establishes a leading Worst-Case Upper Bound of $O(poly(n,m) * 1.220744^n)$, n and m being the number of nodes and edges of the graph G, respectively. The exact technique reported here can be used to compute the redundancy of a line in a communication network.

Keywords: Counting the Number of Independent Sets, Exact Counting, Graph Theory.

1 Introduction

Counting problems are not only mathematically interesting, but they arise in many applications. For example, if we want to know the probability that a formula in propositional calculus is true, or the probability that a graph remains connected given a probability of failure of an edge, we have to count to approximate such probabilities.

Regarding hard counting problems, the computation of the number of independent sets of a graph has been a key for determining the frontier between efficient counting and intractable counting procedures. Vadhan [14] showed that counting the number of independent sets in graphs of maximum degree 4 is #P-complete. Greenhill [8] refined the previous result showing that counting the number of independent sets on graphs of degree 3 or on 3-regular graphs is also #P-complete.

J. Janssen and P. Prałat (Eds.): CAAN 2007, LNCS 4852, pp. 85–98, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Different techniques have been used for counting the number of independent sets of a graph G. These techniques can be classified in two kinds of methods: exact and approximate algorithms.

The Markov chain described by Luby and Vigoda [11] is one of the first approximate counting algorithms for the independence set problem. The application of Markov chain Monte Carlo algorithms has achieved some success to approximate, in polynomial time, the number of independent sets of a graph G, specially for graphs of maximum degree of at most four [6]. Although more Monte Carlo algorithms have been developed further in [5,6,7,8,13], this approximation technique is likely to fail for graphs of maximum degree six or higher, leaving open for now, the case for maximum degree 5 [6].

One of the important trends of research for enumerative combinatorics in general, and for counting the number of independent sets in particular, has been the application of the integer polynomial theory. An excellent example of this line of research is the construction of the independence polynomial of a graph G, defined as: $I(G;x) = \sum_{k=0}^{\alpha(G)} s_k x^k = s_0 + s_1 x + s_2 x^2 + \ldots + s_{\alpha(G)} x^{\alpha(G)}$ where $\alpha(G)$ is the size of the maximum independence set in G and the coefficients $s_k, k = 0, ..., \alpha(G)$ represent the number of independent sets of cardinality k in G. This polynomial is a good representative of the enumerative structure of $NI(G)$ [2,9,10]. In fact, various aspects of combinatorial information related to a graph are kept in the set of coefficients.

In our case, we address the computation of $NI(G)$ using the topological structure of the graph G. We start by traversing G in a depth-first search. We show that if the resulting depth-first graph of G does not contain intersected basic cycles (no pair of basic cycles shares edges) then $NI(G)$ is computed in linear time in the size of the graph G. We have called this polynomial class *Topologically Ordered Graphs* and it generalizes the polynomial classes showed in [4,12,13] for counting the number of independent sets. The Topologically Ordered class establishes a finer border between the class of graphs where the computation of $NI(G)$ is done efficiently versus those which require an exponential time, at least up to now.

In the last section, we present an exact algorithm for computing $NI(G)$ based on the topologically ordered graphs and applying the well-known splitting rule from Davis and Putnam procedure's. Our proposal establishes a leading worst-case upper bound of $O(poly(n,m) * 1.220744^n)$ where $poly(n,m)$ is a polynomial in n and m, the number of nodes and edges in the graph, respectively.

In some cases, we present the relationship between our algorithm and the equivalent property established based on the independence polynomial, with the advantage that the analysis of the time complexity is clearer with our algorithmic point of view. Furthermore, our algorithms can be adapted for solving other counting problems and then, to impact on the time complexity for solving those problems.

The determination of the number of independent sets has several applications in statistical physics [5,7,13,14], e.g. computation in the Potts and hard-core lattice gas model and the problem of counting q-particle Widom-Rowlinson

configurations in graphs, where $q > 2$. Nevertheless, other important application of counting independent sets is for estimating the degree of reliability in communication networks [14].

For example, if we assume that the communication lines (edges) in a network G have the same 'failure probability' and those failures are independent of each other, we can measure different classes of reliability of the network, given that an edge $c \in G$ fails, according to the network component under consideration. A way to estimate the 'redundancy' of a line c in the network G is by applying the conditional probability $P_{c/G}$ which can be approximated by the fraction of the number of independent sets which are added, when the edge c is removed (fails), that is, $P_{c/G} = \frac{NI(G)}{NI(G-c)}$. Thus, $P_{c/G}$ expresses the strategic value of an edge c in a network G by estimating the redundancy of the line. As c is any edge of G, $P_{c/G}$ can be used for estimating the redundancy for any edge of G.

For dynamic networks where their lines are always being reconfigured, as happens when the networks are modelling the Web, for instance, computing $P_{c/G}$ is a growing challenge. Given the intractability nature of the reliability problems in a network, the design of efficient algorithms for computing $P_{c/G}$, or at least exponential algorithms with low growth rate, has been an important area of research.

In this article, we show a leading algorithm for computing the number of independent sets in a graph, so that it can be used to compute the redundancy of the communication lines in a network.

The paper is organized as follows: Section 2 introduces notation, and Section 3 analyzes the basic cases for counting independent sets. The algorithm to count independent sets when the graph does not have intersected cycles is detailed in Section 4, followed by the description of the computation for the general case in Section 5. Before concluding, the complexity of the latter algorithm is discussed in Section 6.

2 Notation

Let $G = (V, E)$ be an undirected graph with vertex set (or node set) V and set of edges E. Two vertices v and w are called *adjacent* if there is an edge $\{v, w\} \in E$, connecting them. Sometimes, the shorthand notation of $u\ v$ is used for denoting the edge $\{u, v\} \in E$.

The *neighborhood* for $x \in V$ is $N(x) = \{y \in V : \{x, y\} \in E\}$ and its *closed neighborhood* is $N(x) \cup \{x\}$ which is denoted by $N[x]$. We denote the cardinality of a set A, by $|A|$. The degree of a vertex x, denoted by $\delta(x)$, is $|N(x)|$, and the degree of G is $\Delta(G) = max\{\delta(x) : x \in V\}$. The size of the neighborhood of x, $\delta(N(x))$, is $\delta(N(x)) = \sum_{y \in N(x)} \delta(y)$.

A path from v to w is a sequence of edges: $v_0v_1, v_1v_2, \ldots, v_{n-1}v_n$ such that $v = v_0$ and $v_n = w$ and v_k is adjacent to v_{k+1}, for $0 \leq k < n$. The length of the path is n. A simple path is a path where $v_0, v_1, \ldots, v_{n-1}, v_n$ are all distinct. A cycle is just a nonempty path such that the first and last vertices are identical,

and a simple cycle is a cycle in which no vertex is repeated, except that the first and last vertices are identical. A graph G is acyclic if it has no cycles.

Given a graph $G = (V, E)$, let $G' = (V', E')$ be a subgraph of G if $V' \subseteq V$ and E' contains edges $v, w \in E$ such that $v \in V'$ and $w \in V'$. If E' contains every edge $v, w \in E$ where $v \in V'$ and $w \in V'$ then G' is called the *induced graph* of G. A *connected component* of G is a maximal induced subgraph of G, that is, a connected component is not a proper subgraph of any other connected subgraph of G. Note that, in a connected component, for every pair of its vertices x, y, there is a path from x to y. If an acyclic graph is also connected, then it is called a *free tree*.

Given a graph $G = (V, E)$, $S \subseteq V$ is an independent set in G if for every two vertices v_1, v_2 in S, $\{v_1, v_2\} \notin E$. Let $I(G)$ denote the set of all independent sets of G. An independent set $S \in I(G)$ is *maximal* if it is not a subset of any larger independent set and, it is *maximum* if it has the largest size among all independent sets in $I(G)$. The determination of the maximum independent set has received much attention since it is a NP-complete problem.

The corresponding counting problem on independent sets, denoted by $NI(G)$, consists of counting the number of independent sets of a graph G. $NI(G)$ is a #P-complete problem for graphs G where $\Delta(G) \geq 3$. $NI(G)$ remains #P-complete when it is restricted to 3-regular graphs [8].

There are different polynomial procedures for computing $NI(G)$ when $\Delta(G) \leq 2$ [4,12,13]. In fact, all of them have linear-time complexity. In the following sections, we present exact combinatorial procedures for computing $NI(G)$ according to the topology of the graph G.

3 Base Cases for Counting Independent Sets

Since $NI(G) = \prod_{i=1}^{k} NI(G_i)$ where $G_i, i = 1, \ldots, k$ are the connected components of G [12,3], then the total time complexity for computing $NI(G)$ is given by the maximum rule as $T(|G|) = max\{T(|G_i|) : G_i$ is a connected component of $G\}$. Thus, from here on, we consider as an input graph only a connected component. We start analyzing the most simple cases for such components.

Case A:
Let $G = (V, E)$ be a graph consisting of a single sequence of nodes (path), i.e. $V = \{1, 2, ..., n\}$ and there exists an edge $e_i = \{i, i+1\}, i = 1, \ldots, n-1$, for each pair of sequential nodes. We build the family $f_i = \{G_i\}, i = 1, \ldots, n$ where each $G_i = (V_i, E_i)$ is the induced graph of G formed by just the first i nodes of V. We associate to each node $v_i \in V$ a pair (α_i, β_i) where α_i expresses the number of sets in $I(G_i)$ where the node v_i does not appear, while β_i conveys the number of sets in $I(G_i)$ where the node v_i appears, thus $NI(G_i) = \alpha_i + \beta_i$.

The first pair (α_1, β_1) is $(1, 1)$ since for the induced subgraph $G_1 = \{v_1\}$, $I(G_1) = \{\emptyset, \{v_1\}\}$. If we know the value for (α_i, β_i) for any $i < n$, and as the next induced subgraph G_{i+1} is built from G_i adding the node v_{i+1} and the edge $\{v_i, v_{i+1}\}$, it is not hard to see that the pair $(\alpha_{i+1}, \beta_{i+1})$ is built from (α_i, β_i) applying the recurrence equation:

$$\alpha_{i+1} = \alpha_i + \beta_i \quad ; \quad \beta_{i+1} = \alpha_i \tag{1}$$

The series (α_i,β_i), i=1,...,n, built from recurrence (1), lead to $NI(G_i)$=$\alpha_i + \beta_i$ for $i = 1, ..., n$. Thus, the computation of $NI(G)$ is based on the incremental calculation of $NI(G_i), i = 1, \ldots, n$. If we perform a linear search on the sequential graph G starting at an extreme, e.g. beginning at v_1 and moving to its incident nodes while applying recurrence (1), in linear time on the number of nodes n, we obtain $NI(G) = NI(G_n) = \alpha_n + \beta_n = F_{n+2}$, where F_n is the $nth-$Fibonacci number [4].

Case B:
Other basic case is when $G = (V, E)$ $|V| = n = |E| = m$ is a simple cycle, i.e. every node in V has degree two. In this case, the cycle can be decomposed as: $G = G' \cup \{c_m\}$, where $G' = (V, E')$, $E' = \{c_1, ..., c_{m-1}\}$. G' constitutes so a sequential graph, and $c_m = \{v_m, v_1\}$ is the edge which, if added to G', forms the cycle G.

Observe that every independent set of G is an independent set of G', that is, $I(G) \subseteq I(G')$ since G has one edge more than G'. Thus, if $S \in I(G')$ and $v_1 \in S$ and $v_m \in S$ then S is not an independent set of G. Then, $I(G)$ can be built from $I(G')$ by eliminating those independent sets containing the nodes: v_1 and v_m. Hence, $NI(G) = NI(G') - |\{S \in I(G') : v_1 \in S \wedge v_m \in S\}|$.

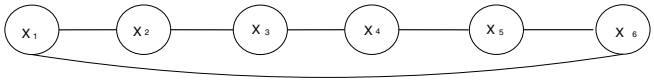
We can apply the case (A) for computing $NI(G')$ since G' is a single sequence of nodes. And, in order to count $|\{S \in I(G') : v_1 \in S \wedge v_m \in S\}|$, we can fix on $I(G')$ the independent sets where v_1 is involved, which is done by computing a new series (α'_i,β'_i), i=1,...,m starting with the pair $(\alpha'_1, \beta'_1) = (0, 1)$, considering in this way only the independent sets of $I(G')$ where v_1 appears. We apply (1) for computing the new series: (α'_i,β'_i), i=2, ...,m and also, in order to consider only the independent sets where v_m appears, the final pair (α'_m, β'_m) is taken only as $(0, \beta'_m)$.

In the following examples, we denote with $\rightarrow$ the application of recurrence (1) on (α_i, β_i) in order to obtain $(\alpha_{i+1}, \beta_{i+1})$. And, if we express the new series in terms of Fibonacci numbers, we have that $(\alpha'_1,\beta'_1) = (0, 1) = (F_0, F_1) \rightarrow (\alpha'_2, \beta'_2) = (1, 0) = (F_1, F_0) \rightarrow (\alpha'_3, \beta'_3) = (1, 1) = (F_2, F_1), \ldots, (\alpha'_m, \beta'_m) = (F_{m-1}, F_{m-2})$, and the value for the final pair $(\alpha'_m, \beta'_m) = (0, \beta'_m)$ is $(0, F_{m-2})$, then $|\{S \in I(G') : v_1 \in S \wedge v_m \in S\}| = 0 + \beta_m = F_{m-2}$.

Then, $NI(G) = NI(G')$ - $|\{S \in I(G') : v_1 \in S \wedge v_m \in S\}| = \alpha_m + \beta_m - \beta'_m = F_{m+2} - F_{m-2}$. Thus, we can formulate the following theorem.

Theorem 1. *If G is a simple cycle with n nodes then the number of independent sets of G, expressed in terms of the Fibonacci numbers, is: $NI(G) = F_{n+2} - F_{n-2}$.*

The formulas obtained in the cases A and B are equivalent to the formulas obtained by Arocha [2] using Fibonacci polynomials. The polynomial defined by $F_0(x) = 1, F_1(x) = 1$, and recursively $F_n(x) = F_{n-1}(x) + x \cdot F_{n-2}(x)$, for $n \geq 2$ is the so-called Fibonacci polynomial. The independence polynomial for a path (sequence of nodes) P_n and a chordless cycle C_n with n nodes can be expressed, as: $I(P_n; x) = F_{n+1}(x)$ and $I(C_n; x) = F_{n-1}(x) + 2x \cdot F_{n-2}(x)$, respectively.



$(\alpha_i, \beta_i) : (1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,3) \rightarrow (8,5) \rightarrow (13,8)$
$(\alpha'_i, \beta'_1) : (0,1) \rightarrow (1,0) \rightarrow (1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,3) \Rightarrow (0,3)$

Fig. 1. Computing $NI(G)$ when G is a simple cycle

Example 1. *Let* $E = \{c_i\}_{i=1}^{6} = \{\{x_1, x_2\}, \{x_2, x_3\}, \{x_3, x_4\}, \{x_4, x_5\}, \{x_5, x_6\}, \{x_6, x_1\}\}$ *be the set of edges of a simple cycle* $G = (V, E)$. *Let* $G' = (V, E')$ *where* $E = E' \cup \{c_6\}$, *so* G' *is* G *without edge* c_6. *As* G' *is a sequence of 6 nodes then* $NI(G') = F_{6+2} = 21$. *While the value for* $|\{S \in I(G') : x_1 \in S \wedge x_6 \in S\}|$ *is* $F_{6-2} = 3$. *Then,* $NI(G) = 21 - 3 = 18$.

We call *Linear_NI* to the linear procedure that consists of the above two cases (A and B). *Linear_NI* will be applied to process any single sequence of nodes or simple cycle that we find as part of a more complex graph.

4 Computing $NI(G)$ When G Has Non-intersected Cycles

Let $G = (V, E)$ be a connected graph with $|V| = n$, $|E| = m$ and such that $\Delta(G) \geq 2$. Let v_r be the node of minimum degree in G, that is chosen to start the depth-first search. Then we obtain a spanning tree T_G where v_r is the root node and a set of fundamental cycles $C = \{C_1, C_2, ..., C_k\}$. Each back edge marks the beginning and the end of a fundamental cycle. Let A_G be the depth-first search graph of G formed by the spanning tree T_G and the set of fundamental cycles C.

We refer as CM to the base cycle matrix of the graph. Each fundamental cycle of A_G is represented in a row of CM. Given any pair of cycles C_i and C_j from C, $i \neq j, i, j = 1, \ldots k$, if C_i and C_j share edges, we call them *intersected* cycles, otherwise they are called *independent* cycles.

Theorem 2. *If the depth-first search graph from a given graph* G *does not contain intersected fundamental cycles, then the number of independent sets of* G *is computed in linear time.*

Proof: We present as proof a linear time algorithm that computes $NI(G)$ and where the main idea is to compute $NI(G)$ by calculating $NI(C)$ for every basic component, i.e. single sequences of nodes or non-intersected cycles, $C \in G$, guiding such computation by a topological sorting on a directed version of G.

Algorithm. *Count_Ind_Sets*()
Input. A_G is the graph built by the depth-first search over G, and CM is the base cycle matrix.
Output. $NI(G)$, the number of independent sets of G.

Procedure:

1. Translate A_G to a Directed Acyclic Graph (DAG), denoted by D_G, assigning an orientation to each edge $\{u, v\}$ in A_G by directing: $u \rightarrow v$ when v is an ancestor node of u in T_G.

   There are two basic structures in D_G: *Branches* and *Rings*.

   A branch B_s of D_G is a directed sequence of nodes which starts in a node v_s where $\delta_{in}(v_s)$=0 and $\delta_{out}(v_s)$=1, the internal nodes of the sequence are reached from v_s through a linear path formed by nodes v_i such that $\delta_{in}(v_i) = 1 = \delta_{out}(v_i)$. The branch ends in the first node v_o such that $\delta_{in}(v_o) > 1$ or $\delta_{out}(v_o) > 1$, and v_o is reached through the linear path started in v_s.

   A ring R_s of D_G is the subgraph formed by the edges and nodes that are part of a fundamental cycle in T_G, so, for each cycle $C_i \in C$, there is a ring R_i, i=1, ...k. The start-node of R_s is the node v_s that is part of the back edge $c_i = \{v_s, v_o\}$ and $\delta_{out}(v_s)$=2, while the second node v_o of the back edge is called the end-node of the ring.
2. Apply a topological sorting procedure on D_G, obtaining an ordered number $'o'$ associated to each node in D_G, such that $o(u) < o(v)$ whenever $u \rightarrow v$.
3. Extend the topological sorting to the substructures of D_G according to the topological order number associated to the end-node of each substructure, such number indicates the order for processing the substructure in step 4.

   When a branch and a ring have the same topological number, the branch is evaluated before the ring. Starting the evaluation of substructures, each pair (α_v, β_v) is set to (0,0), for any node $v \in D_G$.
4. $NI(G)$ is computed traversing each substructure in D_G, according to the topological order of each substructure. When a node v is visited, the pair (α_v, β_v) is computed depending on the substructure in which this node appears, as follows:
   (a) When a branch B_s is evaluated, the basic procedure of case (A) is applied. The evaluation of the branch starts in its initial node v_s, associating the pair (α_s,β_s)=(1,1) when v_s does not have a stored value yet. After evaluating the branch, its initial node and its internal nodes are removed from D_G, as well as its associated edges. The end-node v_o is the only node of the branch preserved, maintaining in (α_o, β_o) the final value obtained after processing the whole branch.
   (b) When a ring R_s is evaluated, the basic case (B) is applied since R_s is evaluated as an undirected cycle. The processing of R_s starts with the start-node v_s associating the values (α_s, β_s)=(1,1) or taking the values already stored in v_s. After evaluating the ring R_s, all of its nodes as well as its associated edges are removed from D_G, preserving only the end-node v_o of R_s where the resulting pair (α_o, β_o) is stored.

   When a node v is considered, this could have been visited before and then a pair $(\alpha_{v_1}, \beta_{v_1})$ has been already associated with v. When v is visited again, a new pair $(\alpha_{v_2}, \beta_{v_2})$ is obtained. The final pair (α_v, β_v) associated to v is computed as: $\alpha_v = \alpha_{v_1} * \alpha_{v_2}$ and $\beta_v = \beta_{v_1} * \beta_{v_2}$, since two different processing lines have met in a common node.

5. The steps (4a) and (4b) are applied repeatedly, since the internal nodes of rings and branches are removed from D_G and new branches could come out. The topological order number assigned to the end-node of the emergent branches determines the order for processing such new branches.

   This iterative process finishes when the root node v_r of T_G is reached and then, $NI(G) = \alpha_r + \beta_r$.

Example 2. *Consider the depth-first search graph G illustrated in figure 2. Taking x_1 as the root node for the depth-first search, the DAG D_G showed in figure 3 is generated by the step 1 of the procedure. The evaluation order in D_G, according to the topological order (step 2), is: $x_7, x_8, x_9, x_3, x_6, x_4, x_5, x_2, x_1$. And the substructures of D_G are: the branches: $x_3 \rightarrow x_6$, and the rings: $x_9 \rightarrow x_6 \rightarrow x_4$, $x_4 \rightarrow x_5 \rightarrow x_2$ and $x_7 \rightarrow x_8 \rightarrow x_2 \rightarrow x_1$ that have to be evaluated in the same order as was previously listed.*

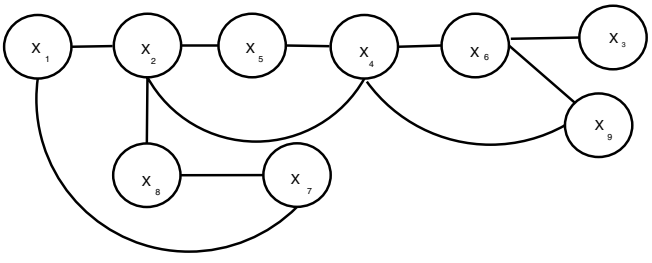


Fig. 2. A depth-first graph A_G

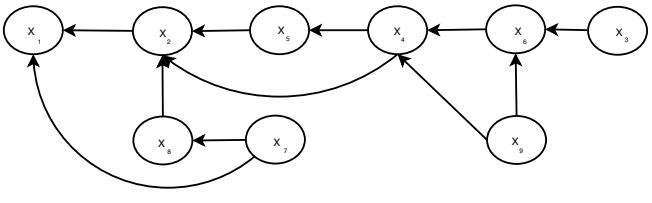


Fig. 3. The DAG associated with A_G

Note that all the sub-procedures involved in *Count_Ind_Sets*, such as depth-first search, topological sorting, processing rings and branches, and so on, are all linear procedures in the size of the graph. Thus, *Count_Ind_Sets* has a time complexity of $O(n + m)$, n and m being the number of nodes and edges of G.

The graphs that satisfy the conditions of theorem 2, constitute a new polynomial class of graphs for counting the number of independent sets. We called this class of graphs, the class of *Topologically Ordered Graphs*. This new class is a superclass of graphs of degree two and it has no restrictions on the degree of the graphs, but it depends on the topological structure of the graph.

5 Computing $NI(G)$ in the General Case

Let $G = (V, E)$ be a graph where $|V| = n$, $|E| = m$, and $\Delta(G) >= 2$. Let T_G be the depth-first graph of G. We assume that T_G is connected and has intersected cycles since otherwise, it can be processed by the procedure presented in the previous section. Let $C = \{C_1, ..., C_{nc}\}$ be the set of fundamental cycles in T_G which are already set apart in the cycle matrix CM. The basic idea is to choose a node v and reduce the problem to count separately the number of independent sets with v and without v.

Algorithm. $NI\_for\_Max\_Degree(T_G)$
Input. $T_G = (V, E)$: the depth-first search graph of G containing intersected cycles.
Output. $NI(G)$: the number of independent sets of G.
Procedure:

1. Apply $Linear\_NI$ to T_G in order to process every single sequence of nodes or simple cycle that T_G could have. This step processes any node u of degree 1 in T_G.
2. Take $S = \{v \in V : v$ is part of an intersected cycle in $T_G\}$. We search for the node $v \in S$ such that v is part of a back edge e_1 and $\delta(v) = max\{\delta(u) : u \in S\}$. If $\delta(v) = \delta(u)$ and v and u have maximum degree in S, we select the node $v \in S$ such that $\delta(N(v))$ has maximum value. Note that the edge e_1 determines a base cycle C_1 and there is at least other cycle C_2 determined by other back edge e_2 such that C_1 and C_2 are intersected in T_G.
3. We apply a splitting reduction rule, as is used in the Davis and Putnam procedure for counting models in Boolean formulas [1,3], being v the selected node for performing the splitting. The rule generates two new graphs from T_G: G_1 and G_2, these are proper subgraphs of T_G and they are formed as follows:
   (a) Build the independent sets without v (Case v set to false):
   Let G_1=(V_1,E_1) be the resulting graph of T_G when v is removed as well as its incident edges, but G_1 includes all the remaining nodes of T_G. $I(G_1)$ comprises the independent sets of T_G that do not contain v.
   Since v is part of a back edge e_1, v is not an articulation point in T_G given that every path crossing by v in T_G, goes now by the other back edge e_2 in G_1. Thus, G_1 is still a connected graph. $|V_1|$=n_1=$n-1$, $|E_1| = m_1 \leq m-3$, and the number of base cycles in G_1 is $nc_1 = m_1 - n_1 + 1 \leq m - 3 - (n-1) + 1 = m - n - 1$, this is, $nc_1 \leq nc - 2$. Then, G_1 has at least two cycles less than T_G.
   (b) Build the independent sets containing v (Case v set to true):
   Let G_2=(V_2,E_2) be the graph obtained from T_G when the closed neighborhood $N[v]$ is removed from T_G as well as any incident edge of $N[v]$. $I(G_2)$ adds up the independent sets where v appears, thus, v and $N(v)$ do not have to be considered further to build $I(G)$.

If G_2 remains a connected graph, then at least the two cycles C_1 and C_2 are no longer in G_2. If G_2 is not a connected graph, this implies that the cycles C_1 and C_2 were decomposed in breaking paths and they are no longer part of G_2, in any case, G_2 has at least two cycles less than T_G.

In any of the two previous cases (a) and (b), at least two intersected cycles of T_G are decomposed and they are no longer part of G_1 and G_2. The application of the splitting rule reduces the number of intersected cycles by at least two, and builds also an enumerative binary tree E_G, where each of its nodes has associated a subgraph of T_G.

4. Once the splitting rule is applied on T_G, the linear procedure *Linear_NI* is employed on the subgraphs G_1 and G_2 to process every new sequence of nodes (branch) and simple cycle (ring) that could appear.
5. The splitting rule is applied repeatedly on each subgraph associated with each node of E_G, whenever such subgraph has intersected cycles.
6. When the associated graph G_h of a node of E_G does not have intersected cycles, then $NI(G_h)$ is computed applying the procedure *Count_Ind_Sets*. In such case, the node is a leaf node of E_G and does not require the application of the splitting rule. We now have $H(E_G) = \{G_h : G_h$ is the graph associated to a leaf node of $E_G\}$.
7. After E_G has been built, we have that $NI(G) = \Sigma_{G' \in H(E_G)} NI(G')$.

The correctness of the algorithm *NI_for_Max_Degree* follows from the following lemma.

Lemma 1. *The result of recursively splitting on the variable v, computes $NI(G)$.*

Proof. Let $G = (V, E)$ be the input graph to the algorithm *NI_for_Max_Degree*. If we can find all the independent sets of G, $I(G)$, then for any node $v \in V$ we can consider the subsets $I_v(G) \subset I(G)$ where v appears and the subset $I_{\neg v}(G) \subset I(G)$ where v does not appear. Note that $I_v(G) \cap I_{\neg v}(G) = \emptyset$ and $NI(G) = |I(G)| = |I_v(G)| + |I_{\neg v}(G)|$. So, when applying the splitting rule (step 3) the branch which considers v set to 'true', computes $|I_v(G)|$ and the other branch considers v set to 'false' and computes $|I_{\neg v}(G)|$. And, at the end of the splitting procedure, the sum of values obtained in both branches is computed, that is, $|I_v(G)| + |I_{\neg v}(G)|$, leading in this way to $NI(G)$.

The splitting rule can be seen as the algorithmic implementation of the following independence polynomial property, described by Hoede and Li in [9]:
$I(G; x) = I(G - v; x) + x \cdot I(G - N[v]; x)$, being v the node selected for applying the splitting rule (step 2, in the previously described algorithm).

Example 3. *Let $T_G = (V, E)$ be the graph illustrated in figure 4, $\Delta(T_G) = 3$, and it has intersected cycles. Applying the procedure NI_for_Max_Degree(T_G), we obtain the enumerative tree E_G where the subgraph G_1 generated when considering x_7 as false is showed in figure 5. Note that the splitting rule has to be applied again in G_1, since it still has intersected cycles. For the case x_7 set to true, the graph G_2 is obtained (showed in figure 6), G_2 is associated with a leaf node of*

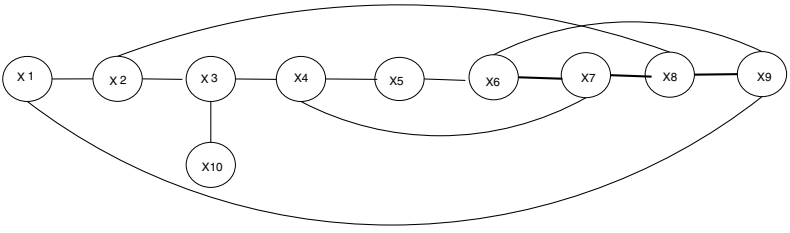


Fig. 4. Applying the splitting rule on T_G

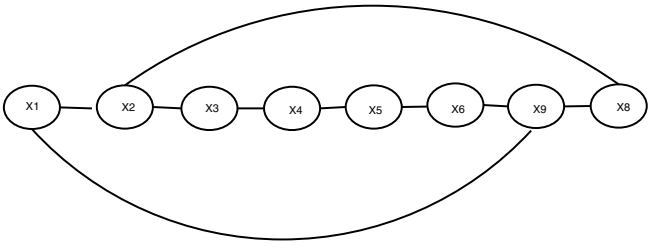


Fig. 5. The splitting rule has to be applied again on G_1

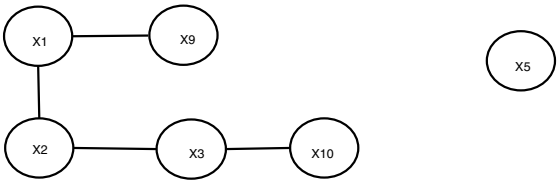


Fig. 6. G_2 is associated with a leaf node of the enumerative tree

E_G since it has no intersected cycles and G_2 can be processed by Count_Ind_Sets *in linear time.*

6 Time Complexity of the Algorithm

Let $G = (V, E)$ be the input graph of the algorithm $NI\_for\_Max\_Degree$, $|E| = m$, $|V| = n$. The steps 1, 2, 4, 6, and 7 have linear time complexity, in fact they are $O(m + n)$. The recursive application of the splitting rule (step 5) generates an enumerative tree E_G. The splitting rule (step 3) is applied while a graph H associated with a node of E_G has intersected cycles. This reduction generates two new graphs $H_1 = (V_1, E_1)$ and $H_2 = (V_2, E_2)$ from H that are proper subgraphs of H.

Notice that the splitting rule always removes $N[v]$ and its incident edges from H to form H_2, and that, for each node $u \in N(v)$ at least one of its incident edges

has been removed from H to form H_1. H_1 and H_2 have at least two intersected cycles less than H.

The time behavior of the algorithm resides in step 5 and corresponds to the number of intersected cycles on the graph associated with each node of E_G. Let use the variable nc to denote the number of cycles in a graph associated with a node of E_G. Then, the time complexity of the algorithm can be expressed by the recurrence: $T(nc) = 2 * T(nc-2) = 2^k * T(nc-2*k)$.

Such recurrence ends when $nc - 2 * k = 1$, that is, when $k = (nc-1)/2$. In consequence, the time complexity is $T(nc) \in O(2^k * (m+n)) = O(2^{(nc-1)/2} * (m+n))$. For the worst case, we can consider that every pair of cycles appearing in any subgraph of the nodes of E_G are intersected, then $nc = m - n + 1$ and so:

$$O(2^{(m-n)/2} * (m+n)) \tag{2}$$

is an upper bound for the time complexity of the algorithm, n and m being the number of nodes and number of edges of the input graph, respectively.

If $\Delta(G) = 3$:
If we assume that the input graph G has maximum degree three then $m \leq (3n)/2$, and the upper bound in (2) is expressed as $O(2^{n/4} * (m+n)) \approx O(1.1892^n * (m+n))$. Thus, the above procedure computes $NI(G)$ for graphs of degree three, including 3-regular graphs, with a worst-case upper bound of $O(1.1892^n * (m+n))$.

When a node v is selected for splitting reduction, for any $u \in N(v)$ such that $\delta(u) = 2$, u will be processed (and then removed) from H_1 by the procedure *Linear_NI*, since at least one edge incident to u is removed during the application of the splitting rule and after, every node with degree 1 is processed by *Linear_NI*, removing so the node and its incident edge.

Then, if each $u \in N(v)$ has $\delta(u) = 2$ then $N[v]$ will not appear either in H_1 or H_2, and the recurrence relation for the time complexity of the algorithm is expressed as: $T(n) = 2T(n - |N[v]|) \leq 2T(n-4)$. It follows that $T(n) \in O((1.1892)^n * (m+n))$.

When some nodes of $N(v)$ remain in H_1 then we need to analyze the relation between the number of edges m and number of nodes n in the graph H, and how this relation is kept in the subgraphs H_1 and H_2. Let $rel = m - n$ be the variable used to indicate the relation between the number of edges and nodes on a graph G. Note that if $rel \leq 0$ for some graph H, then the procedure *Count_Ind_Sets* computes $NI(H)$ in linear time.

We deepen the analysis of the time complexity, considering the different cases of the degree of the selected node v where the splitting rule is applied.

Case $\delta(v) = 4$:
When $\delta(v) = 4$, $|N[v]| = |\{v, u_1, u_2, u_3, u_4\}| = 5$. The only way that $N(v)$ is not removed at all from H_1, after the use of the splitting rule and the application of *Linear_NI*, is that some nodes of $N(v)$ are part of a cycle in H_1. This happens only if there exist at least two nodes $u_1, u_2 \in N(v)$ with degree higher than two. Thus, considering the possible edges among nodes of $N[v]$, we have that: $\delta(N[v]) = \sum_{u \in N[v]} \delta(u) \geq 4 + 2 + 1 + 1 + 1 = 9$.

Therefore, the splitting rule builds the subgraph H_1, where $m_1 \leq |E_1| = m-4$ and $n_1 = |V_1| = n-1$. The relation between the number of edges and number of nodes in H_1 is: $rel_1 = m_1 - n_1 \leq m-4-(n-1) = m-n-3 = rel-3$.

By the other branch, we have that $V_2 = V - N[v]$ and any incident edge to a node of $N[v]$ does no longer appear in H_2. Then, $m_2 = |E_2| \leq m-9$ and $n_2 = |V_2| = n-5$. And, $rel_2 = m_2 - n_2 \leq m-9-(n-5) = m-n-4 = rel-4$. So, the time complexity of the algorithm can be expressed by the recurrence:

$$T(rel) \leq T(rel-3) + T(rel-4) \tag{3}$$

Such recurrence has the characteristic polynomial $p(r) = r^4 - r - 1$ which has the maximum real root $r \approx 1.220744$. This leads to a worst-case upper bound $O(r^n * poly(m,n)) \approx O(1.220744^n * poly(m,n))$.

Case $\delta(v) > 4$:
Notice that if $\delta(v) > 4$, e.g. $\delta(v) = 5$ then $\delta(N[v]) = 6$. And if $N(v)$ remains in H_1 this means that $\delta(N(v)) \geq 5+2+1+1+1+1 = 11$, $m_2 = |E_2| \leq m-11$ and $n_2 = |V_2| = n-6$. Thus, the relation between the number of edges and number of nodes in H_2 is: $rel_2 = m_2 - n_2 \leq m-11-(n-6) = m-n-5 = rel-5$.

By the other branch, we have that $m_1 = |E_1| \leq m-5$ and $n_1 = |V_1| = n-1$, obtaining $rel_1 = m_1 - n_1 \leq m-5-(n-1) = m-n-4 = rel-4$. Then, if $\delta(v) > 4$, this leads to a faster decomposition of the parent graph in the enumerative tree, since the recurrence equation under this circumstance is: $T(rel) \leq T(rel-5) + T(rel-4)$.

Finally, the highest order of growth for the time complexity of our algorithm $NI\_for\_Max\_Degree$ is given by the recurrence (3). Therefore, the worst-case upper bound is $O(r^n * poly(m,n)) \approx O(1.220744^n * poly(m,n))$.

7 Conclusions

Computing the number of independent sets of a graph G, denoted as $NI(G)$, is a classic #P-complete problem for graphs of degree 3 or higher [8]. We establish that if the depth-first graph of a given graph G has no intersected cycles, then the computation of $NI(G)$ is a tractable problem. The new polynomial class for $NI(G)$ does not impose restrictions on the degree of the graph, but rather, it depends on the topological structure of the graph. This polynomial class for $NI(G)$ allows to establish a finer border between the classes FP and #P for the problem of counting independent sets.

Regarding graphs of degree 3, we establish, based also on the topological structure of the graph G, a worst-case upper bound of $O((n+m) * 1.1892^n)$ for computing $NI(G)$, where n and m are the number of nodes and edges, respectively. For the general case, considering graphs regardless of their degree, we establish for the time complexity, a leading Worst-Case Upper Bound of $O(1.220744^n * poly(m,n))$.

One application of the described algorithm is the estimation of the 'redundancy' of an edge c in a network G. Assuming independence and the same 'failure

probability' for each edge in G, the redundancy of c is computed as the conditional probability $P_{c/G} = \frac{NI(G)}{NI(G-c)}$, which allows to measure the strategic value for any edge c in the network G. Furthermore, our proposal can be applied to other counting problems, and it can impact directly on the time complexity of the algorithms for those problems.

References

1. Angelsmark, O., Jonsson, P.: Improved Algorithms for Counting Solutions in Constraint Satisfaction Problems. In: Int. Conf. on Constraint Programming, pp. 81–95 (2003)
2. Arocha, J.L.: Propiedades del polinomio independiente de un grafo. Revista Ciencias Matemáticas V, 103–110 (1984)
3. Dahllöf, V., Jonsson, P., Wahlström, M.: Counting models for 2SAT and 3SAT formulae. Theoretical Computer Sciences 332(1-3), 265–291 (2005)
4. De Ita, G., Tovar, M.: Applying Counting Models of Boolean Formulas to Propositional Inference. In: Advances in Computer Science and Engineering, vol. 19 (2006)
5. Dyer, M., Greenhill, C.: Some #P-completeness Proofs for Colourings and Independent Sets, Research Report Series, University of Leeds (1997)
6. Dyer, M., Frieze, A., Jerrum, M.: On Counting Independent Sets in Sparse Graphs. SIAM J. Comput. 31(5), 1527–1541 (2002)
7. Dyer, M., Greenhill, C.: Corrigendum: The complexity of counting graph homomorphism. RSA: Random Structures and Algorithms 25, 346–352 (2004)
8. Greenhill, C.: The complexity of counting colourings and independent sets in sparse graphs and hypergraphs. Computational Complexity 9(1), 52–72 (2000)
9. Hoede, C., Li, X.: Clique polynomials and independent set polynomials of graphs. Discrete Mathematics 125, 219–228 (1994)
10. Levit, V.E., Mandrescu, E.: The independence polynomial of a graph - a survey, Holon Academic Inst. of Technology (to appear)
11. Luby, M., Vigoda, E.: Approximately counting up to four. In: Twenty-Ninth Annual Symp. on Theory of Computing, pp. 682–687. ACM, New York (1997)
12. Roth, D.: On the hardness of approximate reasoning. Artificial Intelligence 82, 273–302 (1996)
13. Russ, B.: Randomized Algorithms: Approximation, Generation, and Counting, Distinguished dissertations. Springer, Heidelberg (2001)
14. Vadhan Salil, P.: The Complexity of Counting in Sparse, Regular, and Planar Graphs. SIAM Journal on Computing 31(2), 398–427 (2001)

Improving the Efficiency of Helsgaun's Lin-Kernighan Heuristic for the Symmetric TSP

Dirk Richter[1], Boris Goldengorin[2,3,4], Gerold Jäger[5], and Paul Molitor[1]

[1] Computer Science Institute, University of Halle-Wittenberg,
D-06099 Halle (Saale), Germany
richterd@informatik.uni-halle.de, paul.molitor@informatik.uni-halle.de
[2] Faculty of Economic Sciences, University of Groningen,
9700 AV Groningen, The Netherlands
B.Goldengorin@rug.nl
[3] University of Economics and Business, Lviv Highway 51/2,
29016 Khmelnitsky, Ukraine
[4] Department of Applied Mathematics,
Khmelnitsky National University, Ukraine
[5] Department of Computer Science, Washington University
Campus Box 1045, One Brookings Drive
St. Louis, Missouri 63130-4899, USA
jaegerg@cse.wustl.edu

Abstract. Helsgaun has introduced and implemented the lower tolerances (α-values) for an approximation of Held-Karp's 1-tree with the purpose to improve the Lin-Kernighan Heuristic (LKH) for the Symmetric TSP (STSP). The LKH appears to exceed the performance of all STSP heuristic algorithms proposed to date.

In this paper we improve Helsgaun's LKH based on an approximation of Zhang and Looks' backbones and an extension of double bridges further combined with implementation details by all of which we guide the search process instead of Helsgaun's α-values. Our computational results are competitive and lead to improved solutions for some of the VLSI instances announced at the TSP homepage.

Keywords: Traveling Salesman Problem, Lin-Kernighan Heuristic, Tolerances, Backbones, Double Bridge Technique.

1 Introduction

The traveling salesman problem (TSP) is the problem of finding a Hamiltonian cycle with minimum costs of a graph. If the graph has n nodes, a tour T is a permutation $T = (x_1, x_2, \ldots, x_n)$ of the vector $(1, 2, \ldots, n)$ with corresponding costs $c(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n-1} c(x_i, x_{i+1}) + c(x_n, x_1)$. This paper focus on the symmetric case where all costs satisfy $c(x_i, x_j) = c(x_j, x_i)$.

Lin and Kernighan introduced a heuristic which is based on the exchange of k tour edges, called k-swap or k-opt [20]. This local search algorithm still remains at the heart of the most successful approaches. In fact, Johnson and

J. Janssen and P. Prałat (Eds.): CAAN 2007, LNCS 4852, pp. 99–111, 2007.
© Springer-Verlag Berlin Heidelberg 2007

McGeoch [17] describe the Lin-Kernighan (LK) algorithm as the world champion heuristic for the TSP from 1973 to 1989. Further, this was only conclusively superseded by chained or iterated versions of LK, originally proposed by Martin et al. [21,22]. For the TSP, multiple-run heuristics have long been the method of choice when very high-quality solutions are required. Lin and Kernighan [20] have suggested to use pseudo-random starting tours to permit repeated application of their local search procedure. Besides just taking the best of the tours that are produced, Lin and Kernighan propose to use the intersection of the edge sets of the tours as a means to guide further runs of their algorithm. Their idea is to modify the basic procedure so that it will not delete any edge that has appeared in each of the tours that have been found up to that point. They start this restricted search after a small number of tours have been found (they use between two and five tours in their tests). Variations of this idea have been explored recently by Helsgaun [16], Schilham [26], and Tamaki [28]. Considering STSP heuristics, Helsgaun's LKH [16] appears to exceed all further algorithms including the multiple runs of Chained Lin-Kernighan and some other high-end STSP heuristics introduced by Applegate et al. [1], Balas and Simonetti [2], Cook and Seymour [5], Gamboa et al. [6,7], Kahng and Reda [18], Schilham [26], Tamaki [28], and Walshaw [30].

Zhang and Looks [32] made an interpretation of a *backbone* for the STSP as an edge between two cities that appears in all optimal STSP tours. In fact they have measured edge appearance frequencies to estimate the probabilities of backbone variables since to find the backbones is not possible without solving the problem exactly. A theoretical study of backbones is started in Chrobak and Poljak [3] by proving that the intersection of edges from the optimal STSP and Minimum Spanning Tree (MST) solution has at least two common edges. Goldengorin et al. [10,11] have shown that all common edges in all optimal tours have strictly positive upper tolerances, but Libura [19] has indicated that it is $\mathcal{NP}$-hard to find out an upper tolerance for an edge in an optimal tour.

Van der Poort [24] has used the upper tolerance of an edge in MST for an approximation of the upper tolerance of the same edge in an optimal tour and Helsgaun [16] has used the lower tolerance (α-value) for the same purpose. Goldengorin et al. [10,11] and Turkensteen et al. [29] have shown that the arcs with strictly positive upper tolerance in an optimal Assignment Problem (AP) solution are common arcs for all optimal AP solutions. Ghosh et al. [8], Goldengorin and Jäger [9], Goldengorin et al. [12,13], and Turkensteen et al. [29] have applied the largest (bottleneck) upper tolerance for an arc in an optimal solution of the relaxed AP to guide a search of either a high quality heuristic or an exact algorithm for the Asymmetric TSP. Experimentally Helsgaun has used α-values (lower tolerances) for indicating the most likely edges in an optimal STSP solution.

We have used Helsgaun's implementation as a basis and incorporated backbone approximations and tolerances to guide the search process and k-swap-kicks to speed up the search.

In Section 2 we define the notion of tolerances [10,29]. The following sections discuss special aspects of TSP optimization that are suitable to enhance Helsgaun's TSP heuristic: the application of k-swap-kicks in Section 3, backbones in Section 4 and further implementation aspects in Section 5. In Section 6 we give experimental results which show the efficiency of the proposed methods. In particular, they allowed us to set world records for two well-known TSP instances [37]. The paper closes in Section 7 with a summary and suggestions for future work.

2 Tolerances

Tolerances are successfully used to guide the search process within different frameworks of heuristics for the Asymmetric [8,9,12], and Symmetric TSP [16]. A theoretical background of the tolerance based approach for solving different classes of combinatorial optimization problems is outlined in [10,11]. We distinguish between two types of tolerances: upper and lower tolerances. We introduce the concept of tolerances for an "optimal" tour having in our mind that the optimality will be further used with respect to either one of the TSP relaxations (for example, 1-Tree [16]) or a polynomially searchable neighborhood (for example, k-opt [14,15,23,25]), since finding an exact tolerance for a $\mathcal{NP}$-hard problem is also a $\mathcal{NP}$-hard problem.

Given an optimal tour T, we define for each edge $x \in T$ ($x \notin T$) the upper (lower) tolerance as the maximum increase $u_T(x)$ (decrease $l_T(x)$) of the edge length $c(x)$ preserving the optimality of T under the assumption that the lengths of all other edges remain unchanged. Formally, for the edges x, y and $\alpha \in \mathbb{R}$ let

$$c_{\alpha,x}(y) := \begin{cases} c(x) + \alpha, & \text{if } x = y \\ c(y), & \text{otherwise} \end{cases}$$

be a modification of the cost function which changes the costs for edge x to $c(x) + \alpha$. Further let $\mathcal{T}_c$ be the set of all optimal tours. The tolerances with respect to an optimal tour T are defined as follows:

$$u_T(x) := \sup\{\alpha \in \mathbb{R} \mid T \in \mathcal{T}_{c_{+\alpha,x}}\}, \quad \text{if } x \in T$$
$$l_T(x) := \sup\{\alpha \in \mathbb{R} \mid T \in \mathcal{T}_{c_{-\alpha,x}}\}, \quad \text{if } x \notin T$$

Let $T^+(x)$ be an optimal tour under the condition that it contains x, and $T^-(x)$ be an optimal tour under the condition that it does not contain x. Then the upper and lower tolerance of x with respect to the optimal tour T can be computed as follows (see [10]):

$$u_T(x) = c(T^-(x)) - c(T), \quad \text{if } x \in T \tag{1}$$

$$l_T(x) = c(T^+(x)) - c(T), \quad \text{if } x \notin T \tag{2}$$

3 *k*-Swap-Kicks

In Helsgaun's heuristic a greedy initial tour is constructed in each trial, where a trial is a repeated phase in which an initial tour is permanently improved by doing k-swaps until no more improving k-swaps can be found (Helsgaun considers $k \leq 5$). The resulting tours are called k-optimal. As the search space for k-swaps is restricted by a candidate system of all edges, in fact, Helsgaun's code only computes approximations of k-optimal tours. Constructing a new initial tour in each trial leads to the loss of k-optimality. Our idea is to rescue a part of k-optimality in the next trial instead of constructing a new initial tour. We modify the k-optimal tour from the last trial by one or more special l-swaps ($l > k$) and we choose the resulting tour as the new initial tour. In the literature for $k = 4$ this technique is known as *double bridge technique* [17,27]. Figure 1 shows an example of a double bridge move. If we would use only a simple double bridge move (a special 4-swap), the 5-swap search would end in the same local minimum as before. Thus we adopt this technique for special l-swaps with $l \geq 6$.

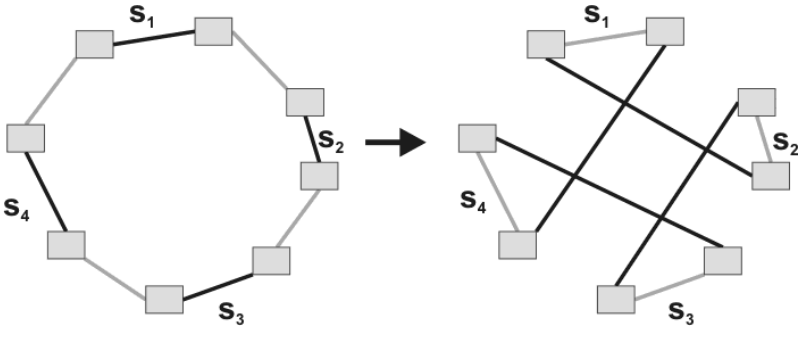


Fig. 1. Double bridge move

In our approach, the edges s_1, s_2, s_3, s_4 shown in Figure 1 are replaced by paths that are segments of the tour, where a *segment* is an ordered list of nodes. So a k-segmentation is a split of a tour T into k segments s_i, so that the concatenation in the order $s_1, s_2, \ldots, s_k$ gives the original tour T, i.e, $concat(s_1, s_2, \ldots, s_k) = T$. Given a k-segmentation $(s_1, s_2, \ldots, s_k)$ of a tour T, we call the k-swap which transforms T into the tour $T' = concat(s_1, s_k, s_{k-1}, s_{k-2}, \ldots, s_2)$ a *k-swap-kick*. Clearly this is a natural extension of a double bridge.

4 Backbones

Helsgaun [16] uses α-values to guide the search process of his algorithm which are lower tolerances to the minimum 1-tree. He shows that using his α-values instead of costs leads to tours with much better quality.

In [25] we have introduced and experimented with tolerances for many problems related to the TSP and different from α-values (e.g. relaxed assignment, assignment, 2-opt) with the purpose to improve Helsgaun's heuristic. Most of these tolerances give worse results in comparison to α-values. In this section, we introduce our most promising approach, the *backbone* tolerance.

It might be possible that there is an edge x which is contained in each optimal tour ($x \in \bigcap \mathcal{T}_c$). Edges occurring in each optimal tour are called *backbones* [4,31,32]. Identifying edges to be a backbone would therefore reduce the problem size and thus speed up a heuristic solving the TSP. Note that backbones are exactly the edges with a strictly positive upper tolerance w.r.t. an arbitrary chosen optimal tour [10].

In [4,31,32] the probability of being a backbone of an edge x is approximated by the relative frequency of occurring in approximated k-optimal tours found during an initialization phase. In our context approximated k-optimal means k-optimal for the restricted search space, i.e. only for the edges in the candidate system.

We measure by means of this relative frequency the probability of being a backbone of an edge x and call it a *backbone approximation*. In other words, the relative frequency of an edge will play the opposite role of its cost.

The main distinction between an exact and a heuristic algorithm is that an exact algorithm proves the optimality of an outputted solution on the whole set of feasible solutions and a heuristic makes a choice of the best solution among a small subset of feasible solutions. If this small subset contains an optimal solution, then the heuristic outputs an optimal solution, otherwise it outputs the best within that small subset. If we replace the optimal solution by the best solution in a small subset and treat it as an optimal one, then we are able to introduce the upper and lower tolerances w.r.t. the best solution for all edges of this small subset. If the small subset is defined for the set of all approximated tours found during an initialization phase, then we have arrived to the notions of *approximated backbone tolerances*. Using (1) and (2), these *approximated backbone tolerances* can be computed as follows:

For an edge e which is contained in any best approximated tour found during the initialization phase, the *upper approximated backbone tolerance* of e is defined as the difference of the optimum value of all approximated tours not containing e minus the optimum value of *all* approximated tours.

For an edge e which is not contained in a best approximated tour (but in at least one approximated tour), the *lower approximated backbone tolerance* of an edge e is defined as the difference of the optimum value of all approximated tours containing e minus the optimum value of *all* approximated tours.

Note that the approximated backbone tolerance is a measure of how likely an edge is in an optimal tour. Whereas backbone approximations use an average value over all tours, approximated backbone tolerances are dominated only by the best tours found during the initialization phase.

5 Implementation Aspects

Based on the ideas of k-swap-kicks and backbones, we have developed a new version of Helsgaun's heuristic.

In all experiments we use the same standard parameters for Helsgaun's heuristic, with two exceptions: we use 5 independent runs instead of 10 independent runs and the internal constant $maxdim = 15,000$ instead of $2,000$ (where $maxdim$ denotes the maximum dimension for which the costs of the edges are fully cached into a matrix in memory), as we have observed that increasing this internal constant considerably improves the general heuristic speed.

At the beginning of the algorithm a set of independent greedy initial tours is chosen, which are improved by one or more trials of Helsgaun's original heuristic, where a trial ends, when 5-swaps can find no further improvement. After this initialization phase we determine a new candidate system depending on backbone approximations. In this way backbone approximations are used to guide the search process instead of Helsgaun's α-values. The decision to apply backbone approximations instead of approximated backbone tolerances is traced back to the fact that the experiments made so far show that approximated backbone tolerances give worse results than backbone approximations in average. Nevertheless, we believe that approximated backbone tolerances are the better approach, thus more sophisticated heuristics have to be found.

Furthermore, in the main phase of the algorithm an initial tour for the next trial is constructed by applying multiple l-swap-kicks ($l > k$) randomly to the approximated k-optimal tour from the last trial. Thus a local optimum can be left with rescuing a lot of k-optimality. Each such start with a new initial tour is called *step*. We choose – like in Helsgaun's original implementation – the number of trials as the number of nodes n.

We use two different implementations: one is tuned for speed, the other for tour quality. In the first implementation which we tuned for speed, at the end of the initialization phase the tour edges are sorted using a randomized quicksort to identify duplicates and to count the occurrence for computing the backbone approximations. In contrast, in the second implementation which we tuned for quality we use (double) hashing instead of quicksort as it saves memory and thus enables to handle larger problems and longer initialization phases (hashing behaved slower than quicksort in our experimental runs). Additionally in the second implementation, the independent initial tours are chosen randomly instead of greedily (this is more effort but leads to slightly better tours) and k-swap-kicks are used with tuned parameters, e.g. we use a better distribution function for the segments.

6 Experimental Results

The following experiments were executed on Intel Xeons 2.4 GHz with 1G RAM. In total we investigated about 4.5 years of running time for all these experiments. All times are given in the format "hours:minutes:seconds".

We tested the algorithms BB... T1 (the first implementation tuned for speed), BB... T2 (the second implementation tuned for quality), and LKH (the original version of Helsgaun). For example BB5P2T1 means that a backbone approximation is used after an initialization of cardinality 5% of the dimension (i.e. $\lceil 0.05 * n \rceil$ initial tours are constructed independently) and each step consists of 2 trials.

6.1 Comparison of Quality for the First Trials

First we compare two variants BB3P2T1 and BB5P2T1 with LKH considering tour quality, more exactly considering the following measure. As the main differences appear in the first trials, we consider only this area.

Let P be the set of analyzed problems, $c^X_{j,p}(i)$ the costs of the tour found by heuristic or tolerance X at Trial i in Run j for a problem $p \in P$. Further let $c_{best}(p)$ be the costs of the currently best known tour for problem $p \in P$ with dimension n_p and R the number of runs. Then we define:

$$avg.excess^X(i) = \frac{1}{|P|} \sum_{p \in P} \frac{1}{R} \sum_{j=1}^{R} \frac{c^X_{j,p}(\frac{i \cdot n_p}{100}) - c_{best}(p)}{c_{best}(p)} \tag{3}$$

As test instances we use the 33 smallest unsolved problems of the national and VLSI instances [34,37].

In Figure 2 the results are shown. We consider at the x-axis the number of trials in percentage up to 20 % of all trials.

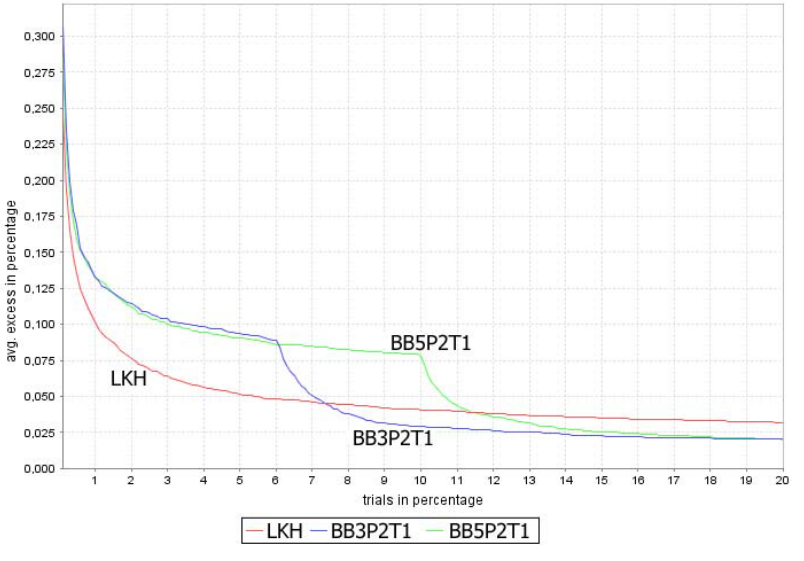


Fig. 2. Average quality of the national and VLSI instances for the first trials

We observe that the average difference to an optimal solution or to the best known lower bound is reduced by 21.73 % for BB3P2T1 and by 24.09 % for BB5P2T1 after *all* trials.

6.2 Improved Instances

During our experiments we have either improved or confirmed the same quality for many TSP instances from the TSP homepage [36]. Two of them xsc6880 and frh19289 were placed at the website [37] as currently best solutions. Note that despite the efforts of many researchers during recent three years we have not only found better tours, but also much faster in terms of normalized times [33]. In Table 1 the detailed results can be found. In the last column you find the normalized running times according to the DIMACS Implementation Challenge. The current overview of this competition can be found in [33].

Table 1. Results for the improved instances

Problem	lb	Found by	Old ub	Algorithm	New ub	Time	Normalized
xsc6880	21,507	Nguyen	21,537	BB3P1T1	21,535	1:28:47	6:08:52
frh19289	55,163	Helsgaun	55,801	BB5P1T1	55,799	49:02:58	125:03:37

6.3 Comparison of Time and Quality

For these experiments again we use the 33 smallest unsolved problems of the national and VLSI instances (because of too large times, some larger problems are only tested by the first implementation tuned for speed). Table 2 and Figure 3 show the results of these experiments. The exact values of average time and average excess can be found in the second and third column of Table 2, respectively. In Figure 3, the average computation time for 5 independent runs is plotted. Thus the more left a point is, the faster the corresponding algorithm is. Smaller excess means better tours in average (see (3)).

We observe that six parameter settings of our versions found faster *and* better tours in average than Helsgaun's version. The version BB1P3T1 is the fastest algorithm which gives nearly the same tour quality as LKH. The backbone version BB3P2T2 finds in average the best tours, but needs more time than LKH.

The k-swap-kicks were the main reason for the speed-up, as for each following trial less k-swaps are needed to find an approximated k-optimal tour. Also we do not need to construct a new initial tour, which additionally saves some time. The shorter the initialization phase is, the worse is the backbone approximation.

7 Summary and Future Research Directions

In this paper we have improved Helsgaun's version of the Lin-Kernighan Heuristic (LKH) which is the world champion heuristic for the Symmetric TSP (STSP) from 1998 to the current date applied to large instances including the World TSP

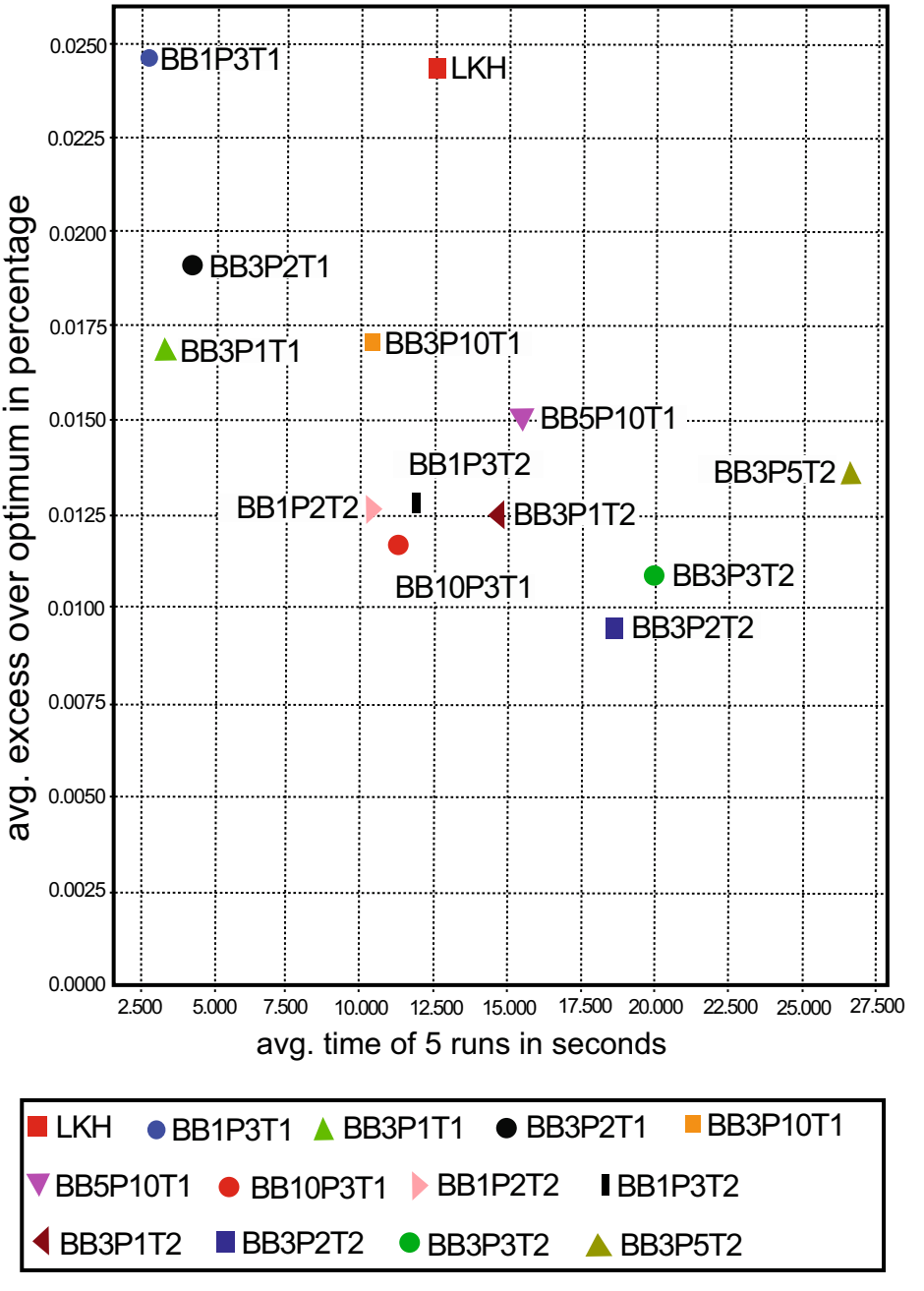


Fig. 3. Average time and average quality for the national and VLSI instances

Table 2. Average time and average quality for the national and VLSI instances

Version	Avg. time in sec.	Avg. excess in %
LKH	03:27:47	0.024493
BB1P3T1	**00:45:32**	0.024716
BB3P1T1	00:55:16	0.016972
BB3P2T1	01:12:29	0.019171
BB3P10T1	02:51:05	0.017121
BB5P10T1	04:13:03	0.015167
BB10P3T1	03:06:33	0.011782
BB1P2T2	02:51:27	0.012631
BB1P3T2	03:16:58	0.012869
BB3P1T2	04:03:36	0.012502
BB3P2T2	05:10:07	**0.009563**
BB3P3T2	05:32:54	0.010899
BB3P5T2	07:22:21	0.013636

[38] with 1,904,711 cities. Our improvements are based on a fundamental notion of a backbone edge, coined by Zhang and Looks [32]. Unfortunately to find a backbone edge has the same computational complexity as to find an optimal tour (see e.g., [3,10]). We have avoided this difficulty by using the notion of backbone approximation which can be efficiently computed, compared to the exact tolerance the computation of which for an optimal tour is also $\mathcal{NP}$-hard. We have used the backbone approximation to guide the search process instead of Helsgaun's α-values (or exactly lower tolerances) computed for the corresponding 1-Tree relaxation of the STSP. Furthermore we have introduced *approximated backbone tolerances* which lead to slightly worse experimental results than backbone approximations. Nevertheless, we will investigate approximated backbone tolerances in more detail, as we believe that we can obtain even better results when applying this approach.

Another improvement is based on a generalization of double bridge move (a special 4-swap) which can be considered as a k-swap-kick for $k \geq 6$ and allows us to speed up the LKH. The above mentioned improvements are incorporated into two different implementations of LKH the first of which is tuned for speed at the initialization phase by a randomized quicksort for computing the backbone approximations. The second implementation is tuned for quality by using the double hashing which reduces the necessary memory and allows us to handle larger instances. Our computational experiments show that, for example, the first implementation leads to an essential quality improvement of outputted tours w.r.t. either the known lower bounds (for instances with unknown optimal tours) or optimal solutions by at least 21% compared to the LKH. Despite the efforts of many researchers during recent three years we have found not only better tours but also much faster in terms of normalized times.

An interesting direction of research is to apply the k-swap-kicks not randomly but guide them by using tolerances for some other promising data structures like stem and cycle including ejection chains for solving large scale STSP instances.

We believe that our notions of backbone approximations and approximated backbone tolerances can be applied for construction improvement type heuristics for other computationally difficult combinatorial optimization problems the first of which is the Capacitated Vehicle Routing Problem and its variations induced by distance-capacitated, time windows, pickup and delivery constraints.

Our source code is available at [35].

Acknowledgement

The research of all authors was supported by a DFG grant MO 645/7-3, Germany. The research of the third author was additionally funded by the United States National Science Foundation grant IIS-053525.

References

1. Applegate, D., Cook, W., Rohe, A.: Chained Lin-Kernighan for Large Traveling Salesman Problems. INFORMS J. Comput. 15(1), 82–92 (2003)
2. Balas, E., Simonetti, N.: Linear Time Dynamic Programming Algorithms for New Classes of Restricted TSPs: A Computational Study. INFORMS J. Comp. 13, 56–75 (2001)
3. Chrobak, M., Poljak, S.: On Common Edges in Optimal Solutions to the Traveling Salesman and Other Optimization Problems. Discrete Appl. Math. 20(2), 101–111 (1988)
4. Climer, S., Zhang, W.: Searching for Backbones and Fat: A Limit-Crossing Approach with Applications. In: AAAI-02, American Association for Artificial Intelligence. Proceedings of the 18th National Conference on Artificial Intelligence (2002), www.aaai.org
5. Cook, W., Seymour, P.: Tour Merging via Branch-Decomposition. INFORMS J. Comput. 15(3), 233–248 (2003)
6. Gamboa, D., Rego, C., Glover, F.: Data Structures and Ejection Chains for Solving Large Scale Traveling Salesman Problems. European Journal Oper. Res. 160(1), 154–171 (2005)
7. Gamboa, D., Rego, C., Glover, F.: Implementation Analysis of Efficient Heuristic Algorithms for the Traveling Salesman Problem. Comput. Oper. Res. 33(4), 1154–1172 (2006)
8. Ghosh, D., Goldengorin, B., Gutin, G., Jäger, G.: Improving the Performance of Greedy Heuristics for TSPs Using Tolerances. Communications in Dependability and Quality Management 10(1), 52–70 (2007)
9. Goldengorin, B., Jäger, G.: How to Make a Greedy Heuristic for the Asymmetric Traveling Salesman Problem Competitive. SOM (Systems, Organisations and Management) Research Report 05A11, University Groningen, The Netherlands (2005)
10. Goldengorin, B., Jäger, G., Molitor, P.: Some Basics on Tolerances. In: Cheng, S.-W., Poon, C.K. (eds.) AAIM 2006. LNCS, vol. 4041, pp. 194–206. Springer, Heidelberg (2006)
11. Goldengorin, B., Jäger, G., Molitor, P.: Tolerances Applied in Combinatorial Optimization. J. Comput. Sci. 2(9), 716–734 (2006)

12. Goldengorin, B., Jäger, G., Molitor, P.: Tolerance Based Contract-or-Patch Heuristic for the Asymmetric TSP. In: Erlebach, T. (ed.) CAAN 2006. LNCS, vol. 4235, pp. 86–97. Springer, Heidelberg (2006)
13. Goldengorin, B., Sierksma, G., Turkensteen, M.: Tolerance Based Algorithms for the ATSP. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 222–234. Springer, Heidelberg (2004)
14. Gutin, G.: Exponential Neighborhood Local Search for the Traveling Salesman Problem. Comput. Oper. Res. 26, 313–320 (1999)
15. Gutin, G., Glover, F.: Further Extension of the TSP Assign Neighborhood. Journal of Heuristics 11(5-6), 501–505 (2005)
16. Helsgaun, K.: An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. European Journal Oper. Res. 126(1), 106–130 (2000)
17. Johnson, D., McGeoch, L.: The Traveling Salesman Problem: A Case Study in Local Optimization. In: Aarts, E., Lenstra, J.K. (eds.) Local Search in Combinatorial Optimization, pp. 215–310. John Wiley and Sons, Chichester (1997)
18. Kahng, A.B., Reda, S.: Match Twice and Stitch: A New TSP Tour Construction Heuristic. Oper. Res. Lett. 32(6), 499–509 (2004)
19. Libura, M.: Sensitivity Analysis for Minimum Hamiltonian Path and Traveling Salesman Problems. Discrete Appl. Math. 30, 197–211 (1991)
20. Lin, S., Kernighan, B.W.: An Effective Heuristic Algorithm for the Traveling Salesman Problem. Oper. Res. 21, 498–516 (1973)
21. Martin, O., Otto, S.W., Felten, E.W.: Large-Step Markov Chains for the Traveling Salesman Problem. Complex Systems 5(3), 299–326 (1991)
22. Martin, O., Otto, S.W., Felten, E.W.: Large-Step Markov Chains for the TSP Incorporating Local Search Heuristics. Oper. Res. Lett. 11, 219–224 (1992)
23. Orlin, J.B., Sharma, D.: Extended Neighborhood: Definition and Characterization. Math. Program., Ser. A 101(3), 537–559 (2004)
24. Van der Poort, E.S.: Aspects of Sensitivity Analysis for the Traveling Salesman Problem. PhD Thesis, Department of Econometrics and Operations Research, University of Groningen, The Netherlands (1997)
25. Richter, D.: Toleranzen in Helsgauns Lin-Kernighan-Heuristik für das TSP. Diploma Thesis, Martin-Luther-University Halle-Wittenberg, Germany (2006)
26. Schilham, R.M.F.: Commonalities in Local Search. PhD Thesis, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, The Netherlands (2001)
27. Stützle, T., Grün, A., Linke, S., Rüttger, M.: A Comparison of Nature Inspired Heuristics on the Traveling Salesman Problem. In: Deb, K., Rudolph, G., Lutton, E., Merelo, J.J., Schoenauer, M., Schwefel, H.-P., Yao, X. (eds.) Parallel Problem Solving from Nature-PPSN VI. LNCS, vol. 1917, pp. 661–670. Springer, Heidelberg (2000)
28. Tamaki, H.: Alternating Cycles Contribution: A Tour Merging Strategy for the Traveling Salesman Problem. Research Report MPI-I-2003-1-007, Max-Planck-Institut für Informatik, Saarbrücken, Germany (2003)
29. Turkensteen, M., Ghosh, D., Goldengorin, B., Sierksma, G.: Tolerance-Based Branch and Bound Algorithms for the ATSP. European Journal Oper. Res., 1–14 (to appear, 2007)
30. Walshaw, C.: A Multilevel Approach to the Traveling Salesman Problem. Oper. Res. 50(5), 862–877 (2002)
31. Zhang, W.: Configuration Landscape Analysis and Backbone Guided Local Search: Part I: Satisfiability and Maximum Satisfiability. Artificial Intelligence 158(1), 1–26 (2004)

32. Zhang, W., Looks, M.: A Novel Local Search Algorithm for the Traveling Salesman Problem that Exploits Backbones. In: Kaelbling, L.P., Saffiotti, A. (eds.) IJCAI 2005. Proceedings of the 19th International Joint Conference on Artificial Intelligence, pp. 343–350 (2005)
33. DIMACS Implementation Challenge: www.research.att.com/~dsj/chtsp/
34. National Instances from the TSP Homepage: www.tsp.gatech.edu/world/summary.html
35. Source code of this paper. http://www.informatik.uni-halle.de/ti/forschung/toleranzen/quelltexte/index.en.php
36. TSP Homepage: www.tsp.gatech.edu/
37. VLSI Instances from the TSP Homepage: www.tsp.gatech.edu/vlsi/summary.html
38. World TSP from the TSP Homepage: www.tsp.gatech.edu/world/

Combinatorial Algorithms for Listing Paths in Minimal Change Order

Zareen Alamgir[1] and Sarmad Abbasi[2]

[1] Department of Computer Science
National University of Computer and Emerging Sciences
Block B, Faisal Town
Lahore, Pakistan
zareen.alamgir@nu.edu.pk
[2] Center for Advanced Studies in Mathematics
Lahore University of Management Sciences
Opposite Sector "U", DHA
Lahore, Pakistan
sarmadabasi@lums.edu.pk

Abstract. Combinatorial algorithms that list combinatorial objects in minimal change order are of fundamental interest in computer science and mathematics. In minimal change ordering, successive elements differ in some pre-specified small way. In this paper, we deal with the generation of paths in a special type of minimal change ordering, the revolving door ordering. We propose a simple algorithm to list all paths in a complete graph, K_n, with n vertices in revolving door order such that each path is generated exactly once. The algorithm is built using space and time efficient schemes that list all spanning paths and "path sets" in revolving door order. Our algorithm is optimal in the sense that it operates in constant amortized time (CAT) and uses linear space.

Keywords: Combinatorial algorithms, Minimal change order, Revolving door order, Complete graph, Generation of paths.

1 Introduction

Generation of the combinatorial objects is of fundamental interest in computer science. In the last few decades, a tremendous amount of research has been carried out in this area as the emergence of high speed computers has made it possible to construct exhaustive lists of combinatorial objects.

We can speed up combinatorial generation by listing objects in minimal change order; an order in which successive elements differ in a small way. Combinatorial algorithms based on minimal change ordering provide new insights into the structure of combinatorial families as they usually involve elegant recursive constructions [8]. The minimal change order in which two consecutive objects have distance two (that is they differ in exactly two positions) is named revolving door order by Nijenhuis and Wilf [1,5]. Much work has been done in listing trees

J. Janssen and P. Prałat (Eds.): CAAN 2007, LNCS 4852, pp. 112–130, 2007.
© Springer-Verlag Berlin Heidelberg 2007

in revolving door order so that successive trees differ only by an edge. Malcolm Simth [5] proposes a revolving door algorithm for listing all spanning trees in a graph and recently Korsh [6] gives a gray code scheme for generating rooted and free trees. However, we know of no published algorithm that generates all paths in revolving door order in a complete graph, K_n, with n vertices.

Finding all paths in a graph is a fundamental aspect in solving scheduling problems, measuring accessibility and traffic flows. In some cases, we can map paths and circuits to some specific class of permutation. This is helpful, since, generation of all permutations is one of the most studied areas in combinatorial generation. Recently, Knuth has compiled comprehensive material on permutation generation in his new volume on combinatorial generation [5]. In [4], Harada enumerates all hamiltonian circuits in a complete graph, K_n, with n vertices. His scheme is based on the Johnson-Trotter method [2,9] for listing permutations. Harada also shows that the hamiltonian paths can be obtained from the hamiltonian circuits but his hamiltonian paths are not in revolving door order.

A brief outline of our contributions is as follows: we propose efficient algorithms for generating all spanning paths and all paths in K_n in revolving door order. Our algorithms are optimal in the sense that they use linear space and operate in CAT, constant amortized time. Thus, they take constant time on average to generate a path. Secondly, our algorithms are based on a recursive structure. Generally, recursive generation algorithms are preferred because they are elegant, flexible and provide insight into combinatorial structure.

The rest of the document is organized as follows: Section 2 gives basic definitions. Section 3 presents a revolving door CAT algorithm for generating all spanning paths in K_n. Section 4 introduces "path set" and gives a recursive scheme for listing path set for a given n in CAT. In Section 5, we develop an algorithm to enumerate all paths in K_n building on the work of previous sections. Finally, Section 6 contains concluding remarks.

2 Basic Definitions

This paper is concerned with enumerating paths in a complete graph K_n. We will assume that the vertex set of K_n is $V_n = \{1, \ldots, n\}$. A path P in K_n is a set of edges

$$\{\{v_0, v_1\}, \ldots, \{v_{k-2}, v_{k-1}\}\}.$$

An enumeration algorithm will output paths as lists of vertices. Two paths P and Q are said to be in minimal change order or revolving door order if

$$|(P \setminus Q) \cup (Q \setminus P)| \leq 2.$$

A path

$$\{\{v_0, v_1\}, \ldots, \{v_{k-2}, v_{k-1}\}\}$$

of length k corresponds to the list

$$L = (v_0, \ldots, v_{k-1})$$

of length k. Let us define

$$\text{rev}(L) = (v_k, \ldots, v_0).$$

Note that L and $\text{rev}(L)$ are two lists representing the same path. We will view the output of all enumeration algorithm as lists of lists. Thus, in order to show that two lists, L_1 and L_2, correspond to different paths we must show that $L_1 \neq L_2$ and $L_1 \neq \text{rev}(L_2)$.

3 Revolving Door Algorithms for Generating Spanning Paths

In this section, we outline a recursive technique for enumerating all $\frac{n!}{2}$ spanning paths in K_n in revolving door order.

3.1 Naive Idea: List Spanning Path Algorithm

Let $P = v_0, \ldots, v_{k-1}$ be a path of length k and $0 \leq i \leq k$. Define

$$\begin{aligned} \text{insert}(P, w, i) &= v_0, v_1, \ldots, v_{i-1}, w, v_i \ldots, v_{k-1} \\ \text{append}(P, w) &= \text{insert}(P, w, k) \\ \text{prepend}(P, w) &= \text{insert}(P, w, 0) \end{aligned}$$

We also define:

$$\begin{aligned} \text{rot}(P) &= v_1, v_2, \ldots, v_{k-1}, v_0 \\ \text{rev}(P) &= v_{k-1}, v_{k-2} \ldots, v_1, v_0 \end{aligned}$$

The following facts are easy to verify:

Fact 1. *For any path P,*

$$\text{rev}(\text{rot}(\text{ rev}(P))) = \text{rot}^{-1}(P).$$

□

Fact 2. *Let P and Q be two paths. $P = Q$ if and only if*

$$\text{rot}^j(P) = \text{rot}^j(Q)$$

and $P = \text{rev}(Q)$ if and only if

$$\text{rot}^j(P) = \text{rev}(\text{rot}^{-j}(Q)).$$

Given a list $\mathcal{L} = P_0, \ldots, P_{t-1}$ of paths of length $n-1$ such that v does not appear on any path in $\mathcal{L}$ we define

$$\text{EXPAND}(\mathcal{L}, v) = Q_0, Q_1, \ldots, Q_{nt-1}$$

where:

$$Q_{kn+j} = \begin{cases} \text{append}(P_k, v), & \text{if } j = 0; \\ \text{rot}(Q_{kn+j-1}) = \text{rot}^j(\text{append}(P_k, v)), & \text{otherwise.} \end{cases}$$

where $0 \leq k \leq t-1$ and $0 \leq j \leq n-1$. The following critical facts are easy to verify:

Fact 3.

$$Q_{kn+n-1} = \text{ prepend}(P_k, v).$$

□

Fact 4. *If v occurs in position i in Q_r then*

$$r \equiv n - i - 1 \bmod n.$$

□

Fact 5. *The last path of EXPAND$(\mathcal{L}, v) =$ prepend(P_{t-1}, v), where P_{t-1} is the last path in $\mathcal{L}$.*

Let $\mathcal{L}_2 = ((1, 2))$. Suppose we define

$$\mathcal{L}_3 = \text{EXPAND}(\mathcal{L}_2, 3) = ((1, 2, 3), (2, 3, 1), (3, 2, 1)),$$

we observe that $\mathcal{L}_3$ is a complete list of all the spanning paths in K_3 in revolving door order. We let

$$\mathcal{L}_n = \text{EXPAND}(\mathcal{L}_{n-1}, n) \text{ for } n > 3$$

and show that $\mathcal{L}_n$ is also a complete list of all spanning paths in K_n in revolving door order.

Theorem 1. *$\mathcal{L}_n$ satisfies the following properties:*

1. *Paths in $\mathcal{L}_n$ are in revolving door order.*
2. *The first vertex of every path, except the first one, is the last vertex of the preceding path.*
3. *For any two distinct paths P and P' in $\mathcal{L}_n$ we have $P \neq P'$.*
4. *For any two (not necessarily distinct) paths P and P' in $\mathcal{L}_n$ we have $P \neq$ rev(P').*

Proof. We proceed by induction on n. The base cases ($n = 2, 3$) are easily seen to be true by inspection. Let

$$\mathcal{L}_{n-1} = P_0, \ldots, P_{t-1}$$

and

$$\mathcal{L}_n = \text{EXPAND}(\mathcal{L}_n, n) = Q_0, \ldots, Q_{nt-1}.$$

We first show that the paths in $\mathcal{L}_n$ are in revolving door order and the last vertex of each path is the first vertex of the preceding path. The rot operation maintains these properties. We verify that claims also holds when we apply the insert operation. Consider two successive paths Q_{kn+n-1} and $Q_{(k+1)n} = \text{append}(P_{k+1}, n)$. We have $Q_{kn+n-1} = \text{prepend}(P_k, n)$ by Fact 3. Hence, n is the last vertex of $Q_{(k+1)n}$ and the first vertex of Q_{kn+n-1}. To see that Q_{kn+n-1} and $Q_{k(n+1)}$ are in revolving door order, observe that by induction the last vertex, w of P_{k+1} is the first vertex of P_k. Hence, $Q_{(k+1)n}$ and Q_{kn+n-1} have the edge $\{n, w\}$ in common, therefore, the only edges that $Q_{(k+1)n}$ and Q_{kn+n-1} differ on are the ones that P_k and P_{k+1} differ on. By induction P_k and P_{k+1} are in revolving door order therefore, Q_{kn+n-1} and $Q_{n(k+1)}$ are also in revolving door order.

Next we show that $Q_r = Q_s$ implies that $r = s$. If $Q_r = Q_s$ then these two paths have the vertex n at the same position, let say at i. By Fact 4 $r = kn + (n - i - 1)$ and $s = k'n + (n - i - 1)$. Furthermore,

$$\begin{aligned} Q_r &= \text{rot}^{n-i-1}(\text{append}(P_k, n)) \text{ and} \\ Q_s &= \text{rot}^{n-i-1}(\text{append}(P_{k'}, n)) \end{aligned}$$

Since $Q_r = Q_s$ by Fact 2 $\text{append}(P_{k'}, n) = \text{append}(P_k, n)$ which implies that $P_k = P'_k$. Therefore by induction $k = k'$ and hence $r = s$. Lastly we show that for all r, s we have

$$Q_r \neq \text{ rev}(Q_s).$$

In this case, if n occurs in position i in Q_r then it occurs in position $n - i - 1$ in Q_s. Hence by Fact 4 we have $r = kn + n - i - 1$ and $s = k'n + i$ for some k and k'. Furthermore,

$$\begin{aligned} Q_r &= \text{rot}^{n-i-1}(\text{append}(P_k, n)) \text{ and} \\ Q_s &= \text{rot}^{i}(\text{append}(P_{k'}, n)) \end{aligned}$$

Now,

$$\text{rot}^{-n+i+1}(Q_r) = \text{append}(P_k, n)$$

On the other hand,

$$\begin{aligned} \text{rot}^{-n+i+1}(Q_r) &= \text{rot}^{-n+i+1}(\text{ rev}(Q_s)) \\ &= \text{ rev}(\text{rot}^{n-i-1}((\text{rot}^{i}(\text{append}(P_{k'}, n))))) \\ &= \text{ rev}(\text{rot}^{n-1}(\text{append}(P_{k'}, n))) \\ &= \text{ rev}(\text{ prepend}(P_{k'}, n)) \\ &= \text{append}(\text{ rev}(P_{k'}, n)) \end{aligned}$$

Thus, $P_k = \text{rev}(P_{k'})$ contradicting the induction hypothesis. Here, we critically use the fact that $P_k \neq P_{k'}$ for all k and k'. □

Corollary 1. *$\mathcal{L}_n$ is a list of all spanning paths in K_n in revolving door order. Furthermore, each spanning path in K_n appears in $\mathcal{L}_n$ exactly once.*

Proof. A simple inductive argument shows that $\mathcal{L}_n$ consist of exactly $\frac{n!}{2}$ spanning paths. Since all paths are distinct, hence, each path must occur in it exactly once. □

3.2 Block Spanning Path Algorithm

The list algorithm does not seem to have an efficient implementation. In this section, we show that the output of the list algorithm can be obtained by simple recursive algorithm that can be implemented efficiently. The reason for introducing the list algorithm is that it is much simpler to prove the correctness of the list algorithm. On the other hand, it is easy to see how to implement the block algorithm efficiently. To show that the block algorithm is correct we simply prove that it produces the same list as the list algorithm.

Given a path $P = v_0, v_1, v_2, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_{n-1}$, the operation $\text{OP}_i(P)$ removes the edge (v_i, v_{i+1}) and inserts a new edge (v_i, v_n). Thus

$$\text{OP}_i(P) = v_{i+1}, \ldots, v_{n-1}, v_i, v_{i-1}, \ldots, v_2, v_1$$

(See Figure 1).

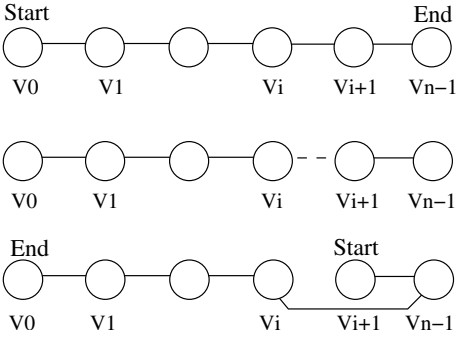


Fig. 1. Operation OP_i

The operation OP_i can be implemented so that it takes time $O(i+1)$ using a simple linked list structure. The following facts are also easily verified:

Fact 6.

$$\text{OP}_0(P) = \text{rot}(P).$$

□

Fact 7.

$$\mathrm{OP}_i(\ \mathrm{prepend}(P, v)) = \mathrm{append}(\mathrm{OP}_{i-1}(P), v).$$

□

Let $\mathcal{O} = O_1, \ldots, O_s$ be an ordered list of operations on paths of length n. Suppose P is a path of length n, then

$$\mathcal{O}(P) = \{P_0, \ldots, P_s\}$$

where

$$P_0 = P \text{ and } P_i = \mathrm{OP}(P_{i-1})$$

that is, $\mathcal{O}(P)$ is a list of paths starting from P that are obtained by successively applying the operations in the list $\mathcal{O}$. Give two lists of operations

$$\mathcal{O} = O_1, \ldots, O_s \text{ and } \mathcal{O}' = O'_1, \ldots, O'_t$$

we define, $\mathcal{O} \oplus \mathcal{O}'$ to be a concatenation of the two lists; that is,

$$\mathcal{O} \oplus \mathcal{O}' = O_1, \ldots, O_s, O'_1, \ldots, O'_t.$$

We also use $\oplus$ to denote the concatenation of lists of paths.

With these notations in mind, we define a *Block* $B_{n,i}$, of order i, as an ordered list of operations given as follows:

$$\begin{aligned}\mathcal{B}_{n,0} &= () \\ \mathcal{B}_{n,i} &= \mathcal{B}_{n,i-1} \oplus (\mathrm{OP}_i \oplus \mathcal{B}_{n,i-1})^{n-i} \\ &= \mathcal{B}_{n,i-1} \oplus \mathrm{OP}_i \oplus \mathcal{B}_{n,i-1} \oplus OP_i \cdots \oplus \mathrm{OP}_i \oplus B_{n,i-1}\end{aligned}$$

We observe the following recursion about $\mathcal{B}_{n,i}(P)$.

Lemma 1.

$$\mathcal{B}_{n,i}(P) = \mathcal{B}_{n,i-1}(P_0) \oplus \mathcal{B}_{n,i-1}(P_1) \cdots \oplus \mathcal{B}_{n,i-1}(P_{n-i})$$

where $P_0 = P$ and P_s is obtained by taking the last path in $\mathcal{B}_{n,i-1}(P_{s-1})$ and applying OP_{i-1} to it.

Correctness of Block Spanning Path Algorithm. We will prove the following recursion for the blocks.

Theorem 2. *Let $n \geq 3$ and $1 \leq i \leq n-2$. If P be a path of length $n-1$ and v be a vertex that is not on P then*

$$\mathcal{B}_{n,i}(\mathrm{append}(P, v)) = \mathrm{EXPAND}(B_{n-1,i-1}(P), v).$$

Proof. We prove this theorem by induction on n where the statement can be checked for $n = 3$ by inspection. For the inductive step we assume that the theorem is true for $n-1$ and prove it for n by induction on i. Firstly, notice that

$$\mathcal{B}_{n,1} = (\mathrm{OP}_0)^{n-1}$$

Thus

$$\begin{aligned}\mathcal{B}_{n,1}(\mathrm{append}(P,v)) &= \mathrm{append}(P,v), \mathrm{rot}(\mathrm{append}(P,v)), \ldots, \mathrm{rot}^{n-1}(\mathrm{append}(P,v)) \\ &= \mathrm{EXPAND}(P,v).\end{aligned}$$

Whereas, $B_{n-1,0}(P) = P$ and therefore,

$$\mathrm{EXPAND}(B_{n-1,0}(P), v) = \mathrm{EXPAND}(P, v).$$

Hence the statement is true for $i = 1$. Now, we proceed with the induction step. Let $i \geq 2$ and consider

$$\mathrm{EXPAND}(B_{n-1,i-1}(P), v).$$

By Lemma 1 we have:

$$\mathcal{B}_{n-1,i-1}(P) = \mathcal{B}_{n-1,i-2}(P_0) \oplus \mathcal{B}_{n-1,i-2}(P_1) \cdots \oplus \mathcal{B}_{n-1,i-2}(P_{n-i})$$

where $P_0 = P$ and P_s is obtained by taking the last path in $\mathcal{B}_{n-1,i-2}(P_{s-1})$ and applying OP_{i-2} to it. We denote the last path of $\mathcal{B}_{n-1,i-2}(P_s)$ by Q_s. Then we have

$$P_0 = P \text{ and } P_s = \mathrm{OP}_{i-2}(Q_{s-1}) \text{ for } s > 0.$$

Hence,

$$\begin{aligned}\mathrm{EXPAND}(B_{n-1,i-1}(P), v) &= \mathrm{EXPAND}(\mathcal{B}_{n-1,i-2}(P_0) \oplus \cdots \oplus \mathcal{B}_{n-1,i-2}(P_{n-i}), v) \\ &= \mathrm{EXPAND}(\mathcal{B}_{n-1,i-2}(P_0), v) \oplus \cdots \oplus \mathrm{EXPAND}(\mathcal{B}_{n-1,i-2}(P_{n-i}), v))\end{aligned}$$

On the other hand by Lemma 1 we also have

$$\mathcal{B}_{n,i}(\mathrm{append}(P,v)) = \mathcal{B}_{n,i-1}(R_0) \oplus \mathcal{B}_{n,i-1}(R_1) \cdots \oplus \mathcal{B}_{n,i-1}(R_{n-i})$$

where $R_0 = \mathrm{append}(P, v)$ and R_s is obtained by taking the last path in $\mathcal{B}_{n,i-1}$ (R_{s-1}) and applying OP_{i-1} to it. We denote the last path in $\mathcal{B}_{n,i-1}(R_s)$ by S_s. Thus

$$R_0 = \mathrm{append}(P, v) \text{ and } R_s = \mathrm{OP}_{i-1}(S_{s-1}) \text{ for} s > 0.$$

We have to show that

$$\mathcal{B}_{n,i-1}(R_s) = \mathrm{EXPAND}(\mathcal{B}_{n-1,i-2}(P_s), v) \text{ for } s = 1, \ldots, n-i.$$

By induction on i it suffices to show that

$$R_s = \mathrm{append}(P_s, v).$$

We will show this by induction on s. This is clearly true for $s = 0$ as $R_0 = \text{append}(P, v)$ and $P_0 = P$. Hence, we can proceed with the induction step. By induction (on s) we have

$$P_{s-1} = \text{append}(R_{s-1}, v)$$

and by induction on i we have,

$$\mathcal{B}_{n,i-1}(R_{s-1}) = \text{EXPAND}(\mathcal{B}_{n-1,i-2}(P_{s-1}), v).$$

This means that the last path S_{s-1} in $\mathcal{B}_{n,i-1}(R_{s-1})$ is equal to the last path in EXPAND($\mathcal{B}_{n-1,i-2}(P_{s-1}), v$). However, the last path in EXPAND($\mathcal{B}_{n-1,i-2}$ $(P_{s-1}), v$) is given by:

$$\text{prepend}(Q_{s-1}, v).$$

Hence,

$$S_{s-1} = \text{ prepend}(Q_{s-1}, v).$$

Now,

$$\begin{aligned} R_s &= \text{OP}_{i-1}(S_{s-1}) \\ &= \text{OP}_{i-1}(\text{ prepend}(Q_{s-1}, v)) \\ &= \text{append}(\text{OP}_{i-2}(Q_{s-1}), v) \\ &= \text{append}(P_s, v) \end{aligned}$$

completing the inductive step. This concludes our proof.

□

The following critical theorem is now easy to prove:

Theorem 3.

$$\mathcal{B}_{n,n-2}((1, 2, \ldots, n)) = \mathcal{L}_n.$$

Proof. The claim can be easily verified for $n = 2, 3$. We only have to observe that $\mathcal{B}_{n,n-2}((1, 2, \ldots, n))$ and $\mathcal{L}_n$ satisfy the same recursion. □

The above theorem proves the correctness of the block algorithm. The block algorithm proceeds as follows: It starts with the path $(1, 2, \ldots, n)$ and applies the operations given in $\mathcal{B}_{n,n-2}$ to it. This generates all the paths in K_n. We record one more fact about the block algorithm.

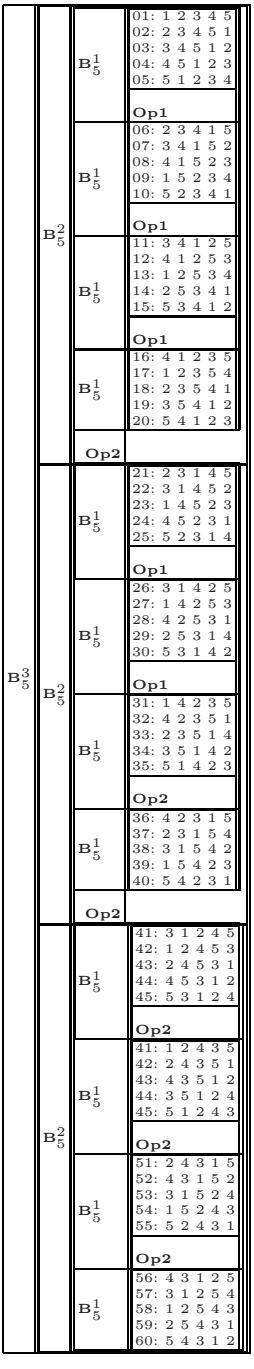
Theorem 4. *Let $P = v_0, \ldots, v_{n-1}$. Then the last path of $B_{n,n-2}(P)$ is given by*

$$v_{n-1}, v_{n-2}, \ldots, v_2, v_0, v_1$$

Proof. We proceed by induction on n. The result being trivial for $n = 2$. For the inductive step let $Q = v_0, \ldots, v_{n-2}$.

$$B_{n,n-2}(P) = B_{n,n-2}(\text{append}(Q, v_{n-1}) = \text{EXPAND}(B_{n-1,n-3}(Q), v_{n-1})$$

Table 1. Expanded version of Block Unit $\mathcal{B}_{5,3}$ for K_5



By induction, the last path L of $B_{n-1,n-3}(Q)$ is given by:

$$L = v_{n-2}, \ldots, v_2, v_0, v_1.$$

Thus the last path of EXPAND$(B_{n-1,n-3}(Q), v_{n-1})$ is

$$\text{prepend}(L, v_{n-1}) = v_{n-1}, v_{n-2}, \ldots, v_2, v_0, v_1. \qquad \square$$

The block algorithm has an easy recursive implementation. An iterative version of this algorithm can also be obtained by implementing a special counter of length $n-2$. The counter keeps track of the OP operation that we have to perform next. In this counter, the value of j^{th} index is incremented up to the maximum value of $n-1-j$. Since OP_j can be implemented in time $O(j+1)$ hence, the total time taken by the algorithm to generate all $\frac{n!}{2}$ spanning paths in K_n is

$$O(\sum_{j=0}^{n-2}(j+1)(n-j)!) = O(n!)$$

and hence, the algorithm runs in constant amortized time.

G.G.Langdon [7] has proposed a method for listing all $n!$ permutations in cyclic order, the list of permutations generated by Langdon's method can be mapped to the list of all spanning paths on n vertices. Langdon's method uses rotate operation to generate all the permutations. Thus, it is not efficient and requires hardware support [5] for the rotate operation to boost its performance. However, we have shown above that our block algorithm takes constant time on average to generate a spanning path with out any need of special hardware.

4 Revolving Door Path Set Algorithm

In this section, we present an idea which is significant in itself and our all path algorithm is also based on it. Consider the following definition:

Definition 1. *A* path set *is a list of distinct paths* $\mathcal{PS} = P_0, P_2, \ldots, P_{t-1}$ *such that the paths are in revolving door order and* $V(P_i) \neq V(P_j)$ *for* $i \neq j$. *Here* $V(P_i)$ *represents set of vertices of path* P_i.

Now we give a recursive scheme to generate the path set $\mathcal{PS}$ on n nodes such that successive paths in $\mathcal{PS}_n$ differs by an edge that is they are in revolving door order. This problem can be restated as to list all 2^n sets on n nodes such that two successive sets differ only by start or end element. We need this condition because any change (add, delete or swap) to the internal node of a path will affect two or more edges of the path. Therefore, two successive path will not be in revolving door order. Many algorithms are proposed [3,5,8] for enumerating all sets and combinations but none full fill our requirement, as ordering in set does not matter.

4.1 Naive Scheme

Consider following two operations which are inverse of the append and prepend operations defined above. Let $P = v_0, \ldots, v_m, k$ then let

$$\text{delrear}(P, k) = v_0, \ldots, v_m.$$

Similarly, if $P = k, v_0, \ldots, v_m$ then let

$$\text{delfront}(P, k) = v_0, \ldots, v_m.$$

We propose a recursive scheme for generating the path set. For $n = 1$ the paths in the list are given as: $\mathcal{PS}_1 = ((1), ())$. Note that PS_1 is a complete path set for K_1 where successive paths differ by a single vertex. Now, assume that we have list

$$\mathcal{PS}_{n-1} = P_0, P_1, \ldots, P_{t-1},$$

where $t = 2^{n-1}$, that enumerates path set for K_{n-1} in revolving door order. To generate the path set for K_n we take two successive paths in PS_{n-1}, namely P_{2i} and P_{2i+1} (these paths differ by exactly one vertex v_k by induction), add two new paths between them by inserting n to P_{2i} and P_{2i+1} such that revolving door order is maintained. Continue this procedure, till path set PS_{n-1} is completely exhausted. More formally, let

$$\mathcal{PS}_n = Q_0, \ldots, Q_{2t-1}$$

where,

$$\begin{aligned} Q_{4j} &= P_{2j} \\ Q_{4j+1} &= \text{add}(P_{2j}, n) \\ Q_{4j+2} &= \text{add}(P_{2j+1}, n) \\ Q_{4j+3} &= P_{2j+1} \end{aligned}$$

Where,

$$\text{add}(P, n) = \begin{cases} \text{prepend}(P, n), & \text{if } n \text{ is odd} \\ \text{append}(P, n), & \text{if } n \text{ is even} \end{cases}$$

Theorem 5.

$$\mathcal{PS}_n = Q_0, \ldots, Q_{2t-1}$$

is a complete list of path sets in revolving door order. Furthermore, if n is even then

$$Q_{2i+1} = \text{append}(Q_{2i}, n) \text{ or } Q_{2i+1} = \textit{delrear}(Q_{2i}, n)$$

and if n is odd then

$$Q_{2i+1} = \textit{prepend}(Q_{2i}, n) \text{ or } Q_{2i+1} = \textit{delfront}(Q_{2i}, n).$$

Proof. The proof is by induction on n and is omitted. □

4.2 Algorithm

An interesting way to generate path sets is by using a binary counter. Let $x = x_0 \cdots x_n$ be a binary string of length n (indexed backwards). We define a set path set $S(x)$ as follows:

$$S(x) = \{i : x_{i-1} \neq x_i : i > 0\}$$

Consider all the strings with $x_0 = 0$. Let us interpret string x in binary and set $y = x + 1$. It is easy to see that two sets $S(x)$ and $S(y)$ differ in exactly one element. Furthermore, one can prove that if $x \neq y$ then $S(x) \neq S(y)$. Thus, we can use a counter to generate the path set as follows:

- Initialize a $n+1$ bit string $x = 0^{n+1}$. Let $S = ()$.
- repeat $2^n - 1$ times.
  - increment x and let j be the bit that was changed to 1.
  - If $x_{j+1} = 0$ and j is odd $S :=$ prepend(S, j)
  - If $x_{j+1} = 1$ and j is odd $S :=$ delfront(S, j)
  - If $x_{j+1} = 0$ and j is even $S :=$ append(S, j)
  - If $x_{j+1} = 1$ and j is even $S :=$ delrear(S, j)

With this simple implementation it is easy to generate the path set in revolving door order.

5 Revolving Door Algorithm for Generating All Paths

A simple idea for generating all paths in K_n is as follows: generate all paths in a path set using the algorithm described in the previous section and apply the block algorithm to each path. This will generate all the paths in K_n. Unfortunately, the paths will not be in revolving door order. Now, the idea is that we use the reverse operation. We can use this operation as a wildcard. By inserting this operation at critical points and in right order we can ensure that all the paths are generated in revolving door order.

Let P be a path of length k. We let $\mathcal{BU}(P)$ denote the list and, with a slight abuse of notation, the last path of $\mathcal{B}_{k,k-2}(P)$. Thus, we assume that the length of the path is implicitly known to us. Let us define

$$Z(0, P) = \mathcal{BU}(P)$$
$$Z(1, P) = \text{ rev}(\mathcal{BU}(P))$$
$$Z(2, P) = \mathcal{BU}(\text{ rev}(P))$$

The option $0, 1, 2$ controls if and when the rev operation is applied. Note, as the reverse operation does not change any edge in a path, therefore, it does not counts as an operation performed the path. It only reverses the orientation of the path.

$$\mathcal{S}_{n-1} = (P_0, \ldots, P_{t-1})$$

Table 2. All Path Sets for K_n

No:	K_1	K_2	K_3	K_4	K_5
1:	()	()	()	()	()
2:	1	2	3	4	5
3:		1 2	3 2	3 4	5 4
4:		1	2	3	4
5:			1 2	3 2	3 4
6:			3 1 2	3 2 4	5 3 4
7:			3 1	2 4	5 3
8:			1	2	3
9:				1 2	3 2
10:				1 2 4	5 3 2
11:				3 1 2 4	5 3 2 4
12:				3 1 2	3 2 4
13:				3 1	2 4
14:				3 1 4	5 2 4
15:				1 4	5 2
16:				1	2
17:					1 2
18:					5 1 2
19:					5 1 2 4
20:					1 2 4
21:					3 1 2 4
22:					5 3 1 2 4
23:					5 3 1 2
24:					3 1 2
25:					3 1
26:					5 3 1
27:					5 3 1 4
28:					3 1 4
29:					1 4
30:					5 1 4
31:					5 1
32:					1

be a list of paths in a path set such that each path appears exactly once in $\mathcal{S}_{n-1}$. Furthermore, suppose we also have $d(i) \in \{0, 1, 2\}$ and paths $Q_0, \ldots, Q_{t-1}$ so that

$$Q_i = Z(d(i), P_i).$$

Here, we think of the numbers $d(i)$ as our guide which inform us if we should reverse the current path before/after applying the block unit to get to Q_i. Furthermore, assume that Q_i and P_{i+1} are in revolving door order. Then the list

$$\mathcal{A}_{n-1} = \mathcal{B}(P_0) \oplus \cdots \oplus \mathcal{B}(P_{t-1})$$

is a list of all the paths of K_{n-1} in revolving door order.

Now we explain how to construct $\mathcal{S}_n$ from $\mathcal{S}_{n-1}$ with similar properties. The list

$$\mathcal{S}_n = R_0, \ldots, R_{2t-1}.$$

We define $e(i) \in \{0, 1, 2\}$ and list

$$S_i = Z(e(i), R_i))$$

such that S_i and R_{i+1} are in revolving door order. To do this we combine the ideas of all spanning path and path set algorithm described in the previous sections.

We will define $R_{4i}, R_{4i+1}, R_{4i+2}, R_{4i+3}, S_{4i}, S_{4i+1}, S_{4i+2}$ and S_{4i+3} in terms of $P_{2i}, P_{2i+1}, Q_{2i}, Q_{2i+1}$. Since, Q_{2i} and P_{2i+1} are in revolving door order, therefore, there must be a k such that:

$$\begin{aligned} P_{2i+1} = \text{append}(Q_{2i}, k) \;\text{ or }\; P_{2i+1} = \;\text{prepend}(Q_{2i}, k) \text{ or} \\ P_{2i+1} = \;\text{delrear}(Q_{2i}, k) \;\text{ or }\; P_{2i+1} = \;\text{delfront}(Q_{2i}, k). \end{aligned}$$

We discuss all these cases one by one.

Case 1: $P_{2i+1} = \text{append}(Q_{2i}, k)$ for some k. We define,

$$\begin{aligned} R_{4i} &= P_{2i}, \\ S_{4i} &= Q_{2i} = Z(d(2i), P_{2i}), \\ R_{4i+1} &= \;\text{prepend}(S_{4i}, n), \\ S_{4i+1} &= Z(1, R_{4i+1}) \\ R_{4i+2} &= \text{append}(S_{4i+1}), k), \\ S_{4i+2} &= Z(1, S_{4i+1}), \\ R_{4i+3} &= \;\text{delfront}(S_{4i+1}), n) = P_{2i+1}, \\ S_{4i+3} &= Z(d(2i+1), P_{2i+1}) = Q_{2i+1}. \end{aligned}$$

Case 2: $P_{2i+1} = \;\text{prepend}(Q_{2i}, k)$ for some k. We define,

$$\begin{aligned} R_{4i} &= P_{2i}, \\ S_{4i} &= Q_{2i} = Z(d(2i), R_{4i}), \\ R_{4i+1} &= \text{append}(S_{4i}, n), \\ S_{4i+1} &= Z(2, R_{4i+1})) \\ R_{4i+2} &= \;\text{prepend}(S_{4i+1}), k), \\ S_{4i+2} &= Z(2, R_{4i+2})) \\ R_{4i+3} &= \;\text{delrear}(S_{4i+2}), n) = P_{2i+1}, \\ S_{4i+3} &= Z(d(2i+1), R_{4i+3}) = Q_{2i+1}. \end{aligned}$$

Case 3: $P_{2i+1} = \text{ delfront}(Q_{2i}, k)$ for some k. We define,

$$\begin{aligned}
R_{4i} &= P_{2i},\\
S_{4i} &= Q_{2i} = Z(d(2i), R_{4i}),\\
R_{4i+1} &= \text{append}(S_{4i}, n),\\
S_{4i+1} &= Z(2, R_{4i+1}))\\
R_{4i+2} &= \text{ delfront}(S_{4i+1}), k),\\
S_{4i+2} &= Z(2, R_{4i+2}))\\
R_{4i+3} &= \text{ delrear}(S_{4i+2}), n) = P_{2i+1},\\
S_{4i+3} &= Z(d(2i+1), R_{4i+3}) = Q_{2i+1}.
\end{aligned}$$

Case 4: $P_{2i+1} = \text{ delrear}(Q_{2i}, k)$ for some k. We define,

$$\begin{aligned}
R_{4i} &= P_{2i},\\
S_{4i} &= Q_{2i} = Z(d(2i), R_{4i}),\\
R_{4i+1} &= \text{ prepend}(S_{4i}, n),\\
S_{4i+1} &= Z(1, R_{4i+1}))\\
R_{4i+2} &= \text{ delrear}(S_{4i+1}), k),\\
S_{4i+2} &= Z(1, R_{4i+2}))\\
R_{4i+3} &= \text{ delfront}(S_{4i+2}), n) = P_{2i+1},\\
S_{4i+3} &= Z(d(2i+1), R_{4i+3}) = Q_{2i+1}.
\end{aligned}$$

We have to show that the path set $R_0, \ldots, R_{2t-1}$ have the desired properties. It is clear that $R_0, \ldots, R_{2t-1}$ is a complete list of paths in path set of $\{1, \ldots, n\}$. Note that we may define $e(4i) = d(2i), e(4i+1) = 1, e(4i+2) = 1$ and $e(4i+3) = d(2i+1)$ (only for Case 1). It remains to show that S_i and R_{i+1} are in revolving door order and $R_{4i+3} = R_{2i+1}$. This is the most critical equality to be established, as the revolving door property of R_i's and S_i's then follow by induction.

Here, we only discuss Case 1 as the remaining cases are similar. Let $Q_{2i} = v_1, \ldots, v_m$ then $P_{2i+1} = v_1, \ldots, v_m, k$. Note that in this case,

$$R_{4i+1} = n, v_1, \ldots, v_m.$$

Hence,

$$\mathcal{BU}(R_{4i+1}) = v_m, v_{m-1}, \ldots, v_2, v_1, n$$

and

$$S_{4i+1} = Z(1, R_{4i+1}) = \text{ rev}(\mathcal{BU}(R_{4i+1})) = v_1, n, v_2, \ldots, v_m.$$

Thus

$$R_{4i+2} = \text{append}(S_{4i+1}, k) = v_1, v_n, \ldots, v_m, k$$

and

$$S_{4i+2} = \mathcal{BU}(R_{4i+2}) = k, v_m, \ldots, v_1, v_n.$$

Thus

$$S_{4i+2} = Z(1, S_{4i+2}) = \ \mathrm{rev}(\mathcal{BU}(S_{4i+2})) = n, v_1, v_2, \ldots, v_m, k.$$

Finally, this shows that

$$R_{4i+3} = \ \mathrm{delrear}(S_{4i+2}, n) = v_1, \ldots, v_m, k = P_{2i+1}.$$

Hence,

$$S_{4i+3} = Z(d(2i+1), R_{4i+3}) = Q_{2i+1}.$$

Where the last equality follows from induction.

The proof that $R_{4i+3} = P_{2i+1}$ is similar for all four cases. The definition of $e(i)$ has to be modified according to the definitions given in each case. Note that for case 3 and 4 the $\mathrm{rev}(P)$ operation is performed only if the length of the path P is greater than 2.

Table 3. All K_3 paths

	$K_n = 3$	
No	Path Set	Last Block Algo Path
1:	1	
2:	3 1	
3:	1 3 2 $\longrightarrow$	2 1 3 $\swarrow$
4:	1 2	
5:	2	
6:	3 2	
7:	3	
8:	()	

5.1 Implementation

The above mentioned scheme can be implemented either recursively or iteratively using counters. Due to the complicated nature of the recursion, the implementation of the above scheme is a bit tricky. It is not hard to show that the algorithm operates in constant amortized time as it is based on two CAT schemes. Using two linked lists: one to store path $P = v_0, \ldots, v_k$ in forward direction and the other to store reverse direction of path, we can implement reverse operation in $O(1)$ time. We have already shown that the cost of the block algorithm is constant amortized. Finally, it is easy to see that the outer-recursion which generates the path set and decides when to apply the reverse operation takes constant time on average. The implementation details will be included in the full version of the paper. Tables 3, 4, 5 list all the paths in K_n generated by our scheme for $n = 3, 4, 5$ respectively. These tables contain only the path set and the

Table 4. All K_4 paths

	$K_n = 4$	
No	Path Set	Last Block Algo Path
1:	1	
2:	1 4	
3:	3 4 1 ⟶	3 1 4 ↙
4:	3 1	
5:	1 3 2 ⟶	2 1 3 ↙
6:	3 1 2 4 ⟶	3 1 4 2 ↙
7:	1 4 2 ⟶	1 2 4 ↙
8:	1 2	
9:	2	
10:	2 4	
11:	3 4 2 ⟶	3 2 4 ↙
12:	3 2	
13:	3	
14:	3 4	
15:	4	
16:	()	

Table 5. All K_5 paths

	$K_n = 5$	
No	Path Set	Last Block Algo Path
1:	1	
2:	5 1	
3:	1 5 4	4 1 5
4:	1 4	
5:	3 4 1	3 1 4
6:	5 3 1 4	4 1 5 3
7:	3 5 1	1 3 5
8:	3 1	
9:	1 3 2	2 1 3
10:	5 3 1 2	2 1 5 3
11:	3 5 1 2 4	4 2 1 3 5
12:	3 1 2 4	3 1 4 2
13:	1 4 2	1 2 4
14:	5 1 2 4	4 2 5 1
15:	1 5 2	2 1 5
16:	1 2	
17:	2	
18:	5 2	
19:	2 5 4	4 2 5
20:	2 4	
21:	3 4 2	3 2 4
22:	5 3 2 4	4 2 5 3
23:	3 5 2	2 3 5
24:	3 2	
25:	3	
26:	5 3	
27:	3 5 4	4 3 5
28:	3 4	
29:	4	
30:	5 4	
31:	5	
32:	()	

last path generated by the block algorithm. All intermediate paths are skipped due to limited space. Note that the path set generated by all path algorithm is in reverse order and slightly different from the list generated by the path set algorithm described in the previous section.

6 Conclusion

Combinatorial path generation is an interesting problem to be studied and its application can be traced to various practical domains. This paper presents efficient algorithms for generating all spanning paths and all paths in a complete

graph, K_n in revolving door order using constant amortized time. Some of the promising ideas that are worth exploring are: generate all paths in an arbitrary graph, investigate ways to list all trees and all linear forests in a graph and develop ranking and unranking schemes for all paths in K_n .

References

1. Nijenhuis, A., Wilf, H.S.: Combinatorial Algorithms for Computers and Calculators. Academic Press, London (1978)
2. Trotter, F.H.: Perm (algorithm 115). Communications of the ACM 8, 434–435 (1962)
3. Ruskey, F., Williams, A.: Generating combinations by prefix shifts. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 570–576. Springer, Heidelberg (2005)
4. Harada, K.: Generation of rosary permutations expressed in hamiltonian circuits. Communications of the ACM 14, 373–379 (1971)
5. Knuth, D.E.: Art of Computer Programming, vol. 4. Addison-Wesley, Reading (2005)
6. Korsh, J., Lafollette, P.: A loopless gray code for rooted trees. ACM Trans. Algorithms 2(2), 135–152 (2006)
7. Langdon, G.G.: An algorithm for generating permutations. Communications of the ACM 10, 298–299 (1967)
8. Savage, C.: A survey of combinatorial Gray codes. SIAM Review 39(4), 605–629 (1997)
9. Johnson, S.M.: Generation of permutations by adjacent transposition. Math. Comp. 17, 282–285 (1963)

Improving Topological Routing in N2R Networks

Jose M. Gutierrez Lopez[1], Ruben Cuevas Rumin[2], Jens M. Pedersen[1], and Ole B. Madsen[1]

[1] Network and Security Department, Aalborg University, Denmark
Niels Jernes Vej 12 9220 Aalborg
jgl@kom.aau.dk, {jens,obm}@control.aau.dk
[2] Departamento de Ingeniera Telemtica Escuela Politcnica Superior Universidad Carlos III de Madrid
Av. Universidad, 30, Edif. Torres Quevedo. E-28911 Legans (Madrid)
rcuevas@it.uc3m.es

Abstract. Topological routing is basically table free, and allows for very fast restoration and thus a high level of reliability in communication. It has already been satisfactorily proposed for some regular structures such as Grid or Honeycomb. An initial proposal has also been developed for the N2R structures. This paper proposes a modification of this previous algorithm, and in addition two other alternatives. The three options are systematically analyzed in terms of executing time and path distances, showing that trade-offs are needed in order to determine which algorithm is best for a given case. Also, the possible practical applications the methods could have, are discussed for different traffic scenarios.

Keywords: Topological routing, N2R, QoS.

1 Introduction

Topological routing is an alternative to traditional routing methods, based on tables. It allows for very fast restoration, and is particularly well suited for large-scale communication where table updates can be time consuming and introduces significant overheads. It has been successfully proposed for a few regular structures such as Honeycomb and Grid [1]. Related to this topological routing idea there are Several studies about Small world networks (SWN) which demonstrate that with limited knowledge of the network, a greedy algorithm can construct short paths using only local information. Examples of these studies are Kleinberg's Small-world models [2] or the Watts and Strogatz Ring Model [3] which are examples of the huge number of publications that prove the existence of topological routing algorithms. The question is if an existing algorithm for N2R structures will perform efficiently.

An initial proposal for using it in N2R structures can be found in [4]; an N2R structure is a generalized Double Ring (DR) topology, where the inner ring links do not interconnect physically neighbour nodes. See Fig 1. A deeper introduction

J. Janssen and P. Prałat (Eds.): CAAN 2007, LNCS 4852, pp. 131–148, 2007.
© Springer-Verlag Berlin Heidelberg 2007

to the N2R structures is given in [2]. As there is already a base to work with, this paper intends to improve the initial proposal and make a more systematic analysis of the performance in terms of execution time and path distances, to decide which is the best option to continue working on, to obtain at least a method with similar properties as table routing techniques, if not better.

The future networks will demand better characteristics for the different services supported. Those characteristics can be related with reliability and a number of other performance metrics [5].

One of the motivations of this paper was the further work described at [4], where an idea of improving the path distances obtained, and especially the maximum distance in communication between nodes, was launched. The path length is directly related with the availability [5] of the communications, and the budget related with fiber placement [6]. The availability of a system is proportional to the possible failure points on a communication. Therefore, if the number of those failure points is reduced, the availability will be higher. The maximum path length is an important factor to be able to guarantee a certain level of QoS at a given network.

When building a network, the budget is always a conditioning factor. The highest investment on a fiber optic network is the fiber placement (about 70 %) [6]. Minimizing the paths reduces the length of ditches needed to dig up to install the fiber, hence, this reduction directly implies an important reduction on the budget.

Wired networks (such as FTTH) have the handicap of the fiber physical cuts which implies the loss of connectivity between nodes [7]. The topological routing allows a fast adaptation for a network in case of a failure [4]. If more than one path can be offered to establish a communication between any pair of nodes, the consequence is a higher reliable network, decreasing the probability of loss of communication.

When offering topological routing, the routers do not need routing tables reducing the number of operation to be developed. The question is if more complex routing algorithms are capable of obtaining better communication paths with no dramatic impact on the delay of the packet which can be a critical factor. Then, if this goal is achieved, the performance of the network can be improved.

The goal is to find a general algorithm that gives the opportunity of routing topologically a packet from any node to any node at any possible N2R configuration, improving the previous results. It is assumed that if there is a general algorithm for all the configurations, there is feasible topological routing solution for any specific case.

In case that the improvement is achieved, then more arguments can support the N2R structure to be considered as an option on the network structure design at any layer level.

The contribution in this paper forms the first step towards a useful algorithm for topological routing in N2R. We provide algorithms based on node addresses instead of tables, and evaluate their performance in terms of execution times

and path lengths. The adaptiveness, which is indeed crucial, must be dealt with in future research.

The structure of the rest of the document is as follows. Section 2 treats the definitions and the proper notation to understand the network structure under study. Section 3 introduces the modification of the previous algorithm, and the two new proposals implemented for the topological routing. In Section 4 the proposed algorithms are explained with formulas and examples, and in section 5 the algorithms are simulated and the results are commented. Finally, Section 6 exposes the conclusions extracted from this paper, and it introduces the further work to be developed in Section 7.

2 Definitions and Notation

The first of the definitions are given to understand the whole concept of this paper, topological routing. Topological routing is understood as follows:

At a given address scheme, from any node, any packet can be routed given only knowledge of the addresses of the current node and the destination node, with no routing tables involved [1].

The number of nodes in the N2R structure is any positive even integer, larger or equal to 6. These rings each contain the same number of nodes (p). Links in the outer ring and the links interconnecting the two rings can be described in the same way as the DR structure, but links in the inner ring are interconnecting node I_i and node $I_{(i+p)modq}$, where q is a positive integer. To avoid forming two separated networks in the inner ring, q must fulfil $gcd(p,\ q) = 1$ (Greatest Common Divisor), also q is evaluated from 1 to p/2 [8].

These N2R structures have been studied and compared to other degree three topologies. The results show that N2R structures are preferable regarding low delays, high bandwidth and reliability, and also when errors occur [8]. Therefore, the effort of improving the topological routing might end up in possible solutions that, in the future, can be compared to the table routing techniques at a real scenario. At the moment, the goal is to solve the problems identified at the first algorithm proposed at [4], to be able to implement an efficient routing algorithm in terms of path distances and delay of communications.

The addresses of the nodes are given in a certain way to make the algorithm as easy and fast as possible. The outer ring nodes addresses vary from *0* to *p-1* counterclockwise and the inner ring addresses vary from *p* to *2*p-1*. The relation between the outer and the inner nodes, is in the way that the outer node X is connected to the inner node $X+p$.

This address system allows simple operations at the programming of the algorithm, and the addresses of the neighbours are well known by every node.

Each node is connected to the neighbours with three links. In each of the cases there is a possibility of naming them as left, right and center *(L, R, C)*. Every node knows the address of the neighbors by looking at the name of the link it is connected to. At a given address X of a outer node, to follow the link L means to reach the node $X+1$, link R means $X-1$ and link C means $X+p$. In

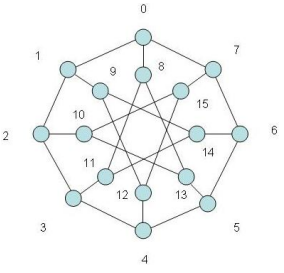


Fig. 1. N2R(8,3) notation

the same way with the inner nodes, to follow the link L means to reach the node $X+q$, link R means X-q and link C means X-p (considering *mod* p for all the neighbours addresses at the same ring). Fig. 1 illustrates this idea.

The node mechanism to treat a packet is to identify the destination, run the proper algorithm, and then forward it using one of the three links it is connected to. Before and after this operation, the node has no task to work on such as neighbours recognition or distance vectors. Therefore the node tolerance to saturation is higher due to the less number of operations to execute.

3 Node Decision

When routing a packet, the nodes have to make the decision of which of their three links should be used. This decision is based on an internal algorithm that calculates the values of the different possibilities (distance in number of hops).

In this section the methodology and the different options available to obtain the different algorithms to support topological routing over N2R structures are analyzed. This analysis is theoretical, and based on the discussion of the different options, at Section 4 the algorithms implementation is explained.

The assumed information implicit at every node is:

- p and q of the N2R structure
- Node Id (address): N_x

Each of the nodes has the possibility of knowing exactly which link to forward the packet, without knowing the source node.

For the purpose of the packet routing, three possibilities are studied and compared to obtain the feasibility of the use of topological routing at a N2R structure. The subsections 3.1, 3.3 and 3.2 describe the three algorithms proposed: Fast Response Algorithm, Optimal Path Algorithm and Balanced Algorithm.

3.1 Fast Response Algorithm (*FRA*)

The algorithm proposed is based on the previous algorithm treated in [4]. The mathematical properties used are the same, but it is implemented with some slight differences.

The previous algorithm made a difference between the source and the middle nodes. The procedure was to identify at the source node the kind of communication, (using the outer ring or the inner ring), and then add flags to the header of the packet to identify the type of communication at the middle nodes. These flags are helpful when the current and destination nodes are to reduce the number of instructions to execute:

- When current and destination is both at the outer ring, and the source node was also at the outer ring.
- When current and destination is both at the inner ring.

At the rest of the cases the middle nodes will have to execute the whole algorithm, since this flags are not helpful. Further explanation at [4]. In our implementation, however, we do not use these flags. Thus, in the modified algorithm it does not matter if nodes are source or middle nodes. The motivation of the modification was also to try to have a header as small as possible to complete a communication and to try to adapt it to the theoretical definition of topological routing (just current and destination addresses).

The rest of the mechanism is the same as the previous algorithm, but in any case it is described and commented to give a better understanding of the two other proposals. The attribute *"Fast Response"* is given due to the goal of a low execution time algorithm by using simple operations and a relatively short script.

A main reason to implement this algorithm was to obtain values for the executing time, (at the previous work it was not considered,) in order to be able to include another testing factor in the analysis of the three possibilities. Hence, even though there will be executed fewer instructions at some occasions, (the cases when the flags can be used), the executing time will be assumed to be the same due to their similarity.

This algorithm gives as a result the same paths as the previous one. Hence, it is already known that the result of the paths used in the communications using this method will not always give an optimal solution in terms of distance (number of hops). This result will be compared with the *"Optimal Path"* and *"Balanced"* solution at Section 5 to find the balance between total path length and path completion time.

The algorithm is theoretically described below and the implementation information is explained in Section 4.

A source node is selected to start the communication, then the relative position of the destination node is calculated to determine the orientation of the communication. The structure is virtually split in half from the source node, if the relative position is at the right half of the structure, the communication will be established in the right direction, and vice versa with the left half. The same method is applied for all the possible communications, without mattering the position of the nodes (at the inner or outer ring). This method simplifies the implementation of the algorithm to a few instructions. Fig. 2 illustrates this method with an example of the communication obtained.

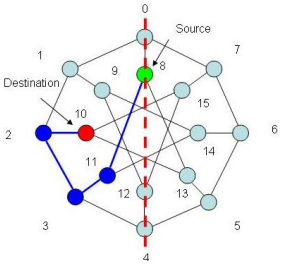


Fig. 2. N2R(8,3) Fast Response Path calculation

Then, two of the three links are left as possible gateways. The decision between these two is made by comparing the distances using the outer ring, the inner ring, or both for the communication. Each type of communication is directly related with a link, hence the link can be easily recognized.

The best result is used to forward the packet, and at the next node the same operations are realized, until the destination node is reached. Thus, in every node the next hop is found by a qualified guessing on which potential hop is closest to the destination.

3.2 Balanced Algorithm (*BA*)

The name *"Balanced Algorithm"* is given because it is a priori expected that the values of path length and execution time will be in between the values of the other two algorithms.

This method considers to use the three link to forward the packet (the *FRA* only considers two from the very beginning, the third one is discarded at the side recognition task). The idea is to find the values of the different ways the communication can be established, using the outer ring, the inner ring or both. The procedure is the same as at the *FRA* but also including the opposite orientation of the communication (if the source and destination are at the right half of the network, the communication oriented to the left is also considered).

Including the third possible gateway has the consequence that more instructions and conditions are needed for running the algorithm. Hence, the difference between the executing time and the path length obtained must be analyzed, to see if the costs make it a real possibility.

The Fig. 3 illustrates the same example as at Section 3.1 obtaining a better result (in terms of distance) at the end of the communication.

3.3 Optimal Path Algorithm (*OPA*)

In this case the algorithm implemented obtains the optimal solution in terms of number of hops. The results obtained were compared to the results of previous works, to verify the attribute *"Optimal Path"* [9]. This algorithm includes a loop to be able to find the best solution, which makes the execution time much

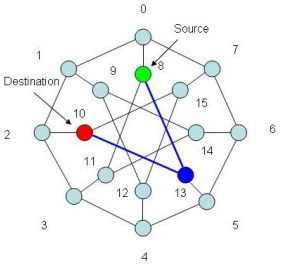


Fig. 3. N2R(8,3) Balanced Algorithm calculation

longer, directly related with a longer delay at the nodes. Therefore, the time a node will be treating a packet, is obviously longer than the *FRA* or the *BA* and the discussion must be focused on the trade off between execution time-path distance.

In case that the paths are shorter, less nodes are involved in the communication between a pair of nodes. Hence, the algorithm needs to be executed less times to reach the destination. At Section 5 this discussion will give the parameters involved in the three cases.

4 Algorithm

At this section the algorithms used to prove the theoretical statements formulated at Section 3 are explained in detail. From the theoretical point of view of the topological routing methods are quite simple, but at the time of programming an unified algorithm for all the possible configurations, it turned to have some difficulties for very specific cases (the *OPA*).

Next, the basic mechanism of the algorithms is deeply commented and explained with graphic examples. This explanation focuses on the mathematical properties of the N2R to define the general rules that must be followed to work on the topological routing issue. One of the bases of the algorithms is that a packet is never routed to the same node twice. Therefore there is no possibility of loops, count to infinite problems, or suboptimal results.

4.1 Fast Response Algorithm (*FRA*)

The algorithm implemented in this case, following the rules defined at Section 3.1, is described as follows:

– Side Recognition: Being S and D any source and destination nodes and their id N_S and N_D, the orientation of the communication is easily calculated by formula (1).

$$D_{out} = N''_D - N_S \tag{1}$$

For this calculation the ring that the nodes belong to must be considered. The following operations, formulas (2) and (3), are required to obtain a result in the necessary range to make the comparison [1]:

$$N'_D = \begin{cases} N_D & \text{if } N_S < p \ \&\ N_D < p \text{ (O - O)} \\ N_D & \text{if } N_S \geq p \ \&\ N_D \geq p \text{ (I - I)} \\ N_D - p & \text{if } N_S < p \ \&\ N_D \geq p \text{ (O - I)} \\ N_D + p & \text{if } N_S \geq p \ \&\ N_D < p \text{ (I - O)} \end{cases} \tag{2}$$

$$N''_D = \begin{cases} N'_D & \text{if } N'_D > N_S \\ N'_D + p & \text{if } N'_D < N_S \end{cases} \tag{3}$$

Applying these properties, the value of D_{out} gives the orientation of the communication as it is explained at (4):

$$D_{out} \begin{cases} \leq p/2 & \text{Left orientated} \\ > p/2 & \text{Right orientated} \end{cases} \tag{4}$$

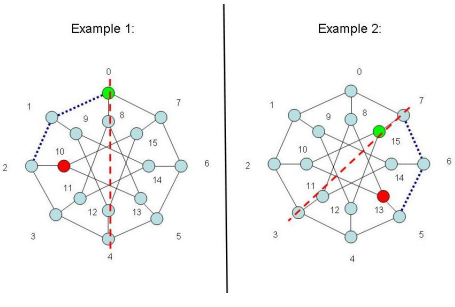


Fig. 4. Examples of Side Recognition

In Fig. 4 there are two examples illustrating this idea. Example1 (N2R(8,3), p=8 ,q=3):

- $N_S = 0$ (outer ring) and $N_D = 10$ (inner ring)
- The nodes are located at different rings:$N'_D = N_D - p = 10 - 8 = 2 = N''_D$.
- $D_{out} = N''_D - N_S = 2 - 0 = 2 \leq p/2$
- Left side orientated communication.

Example 2 (N2R(8,3), p=8 ,q=3):

- $N_S = 15$ and $N_D = 13$. Both at the inner ring.
- $N'_D = N_D$ and $N'_D < N_S$. Then $N''_D = N'_D + p = 13 + 8 = 21$.
- $D_{out} = N''_D - N_S = 21 - 15 = 6 > p/2$
- Right side orientated communication.

[1] O=Outer Ring, I=Inner Ring.

– Link Decision: The link decision method is to find the best option between the two links available for the communication (the third one was already discarded at the side recognition step). Hence, the criterion is to find the shortest distance (using the outer ring, the inner ring or both). The value of the distance using the outer ring (DT_{out}) is related with the previous value of D_{out} (11). There are two considerations related with this value:
  - Depending on the orientation of the communication, the value of D_{out} must be converted as it is shown in (5):

$$D'_{out} = \begin{cases} p - D_{out} & \text{if } D_{out} > \text{p}/2 \\ D_{out} & \text{if } D_{out} \leq \text{p}/2 \end{cases} \tag{5}$$

    The reason is that the value obtained was the distance orienting the communication to the left. Hence, since the number of nodes at the outer ring is p, this operation will give the distance using a right oriented communication in the case of being shorter. At Fig. 4 the hops taken at the outer ring (D'_{out}) are represented with a blue dotted line at both examples.
  - To obtain the total value of the distance using the outer ring (DT_{out}), the hops in between rings must be considered (D_{rout}). The total distance is given by formula (6).

$$DT_{out} = D'_{out} + D_{rout} \tag{6}$$

    The value of D_{rout} depends on in which ring the source and the destination are located, the possible values are presented at (7) :

$$D_{rout} = \begin{cases} 0 \text{ if } N_S < p \ \& \ N_D < p \text{ (O - O)} \\ 1 \text{ if } N_S \geq p \ \& \ N_D < p \text{ (I - O)} \\ 1 \text{ if } N_S < p \ \& \ N_D \geq p \text{ (O - I)} \\ 2 \text{ if } N_S \geq p \ \& \ N_D \geq p \text{ (I - I)} \end{cases} \tag{7}$$

The inner ring distance (DT_{in}) is related with the number of hops at this ring to reach the destination (D_{in}). This value is easily calculated by formula (8)[2].

$$D_{in} = Round(D'_{out}/q) \tag{8}$$

There are two possibilities for (DT_{in}), using the inner ring or both, it is explained at (9):

- If D'_{out}/q is an integer, the communication will only need to use the inner ring.
- If D'_{out}/q is not an integer, the communication will use both of the rings.

$$DT_{in} = \begin{cases} D_{in} + D_{rin} & \text{if } D'_{out}/q \in \mathbb{I} \\ D_{in} + D_{rin} + Dq_{out} & \text{if } D'_{out}/q \notin \mathbb{I} \end{cases} \tag{9}$$

The values in these cases for D_{rin} depending on the ring location of the nodes are presented at (10).

[2] The Round function converts a real number to the closest integer.

$$D_{rin} = \begin{cases} 2 \text{ if } N_S < p \ \& \ N_D < p \text{ (O - O)} \\ 1 \text{ if } N_S \geq p \ \& \ N_D < p \text{ (I - O)} \\ 1 \text{ if } N_S < p \ \& \ N_D \geq p \text{ (O - I)} \\ 0 \text{ if } N_S \geq p \ \& \ N_D \geq p \ \& \ D'_{out}/q \in \mathbb{I} \text{ (I - I)} \\ 2 \text{ if } N_S \geq p \ \& \ N_D \geq p \ \& \ D'_{out}/q \notin \mathbb{I} \text{ (I - I)} \end{cases} \quad (10)$$

The value of Dq_{out} is the distance from the node where the communication jumps from the inner ring to the outer ring (N'_q), to the destination node. This value is calculated as D_{out} by formula (1) and realizing the same necessary conversions. The value of N_q is calculated by formula (11), "+" for left oriented and "-" for right oriented:

$$N_q = N_S \pm D_{in} * q \quad (11)$$

The same as in formula (1) the values must be given in the proper range. Therefore, assuming that $0 \leq N_S < p$ (value converted if necessary) there are two conversions depending on the orientation, see (12):

$$N'_q = \begin{cases} N_q - p \text{ if } N_q \geq p \text{ (Left orientated)} \\ N_q + p \text{ if } N_q < 0 \text{ (Right orientated)} \end{cases} \quad (12)$$

In this way the values of N'_q are in the proper range to work with.

– Comparison: At the end, the comparison between the values of DT_{in} and DT_{out} (the shortest one is chosen) determines how the communication will be established, hence, the link to use to forward the packet. The algorithm does not calculate the paths, it calculates the possible distances to be able to decide which link to use to forward the packet.

Fig. 5 illustrates two examples to better understand the real meaning of the values calculated. At the Example 3 (same procedure for example 4):

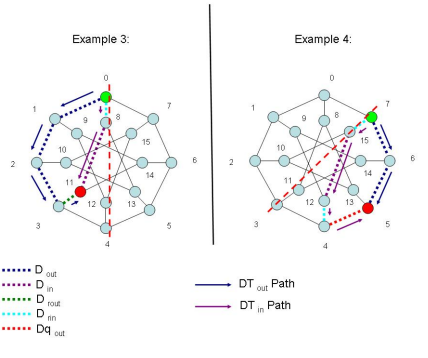


Fig. 5. Examples Node Decision

- N2R(8,3), p=8 ,q=3
- DT_{out}:
  $N_S = 0$ and $N_D = 11$, different rings: Convert $N'_D = N_D - p = 11 - 8 = 3 = N''_D$
  $D_{out} = N''_D - N_S = 3 - 0 = 3 = D'_{out}$ (Left orientated)
  $D_{rout} = 1$ (Outer-Inner)
  $DT_{out} = D'_{out} + D_{rout} = 3 + 1 = 4$
- DT_{in}:
  $D_{in} = D'_{out}/q$=3/3=1, integer. Only inner ring used.
  $D_{rin} = 1$ (Outer-Inner)
  $DT_{in} = D_{in} + D_{rin} = 1 + 1 = 2$
- $DT_{in} < DT_{out}$, therefore the link is used C (center). Start all over from the next node.

4.2 Balanced Algorithm (*BA*)

The basic idea is the same, to find the DT_{out} and DT_{in} to make a decision. In this case there are two distances related with the inner ring to calculate, one for a clockwise communication and the other for counterclockwise.

Being D_{in1} and D_{in2} the hops at the inner ring to establish a connection in both directions, D_{in1} is the value obtained using the already commented formula (8). D_{in2} corresponds to formula (13).

$$D_{in2} = Round((p - D'_{out})/q) \tag{13}$$

The procedure to calculate the values of DT_{out}, DT_{in1} and DT_{in2} are exactly the same as in Section 4.1, applying the same proper conversions. These three values are compared and the best option is used to forward the packet. The procedure is the same at the next node until the destination is reached.

4.3 Optimal Path Algorithm (*OPA*)

This algorithm obtains at the end of any communication the shortest path possible for all the situations and N2R configurations. In order to achieve this goal the algorithm must be much more complex than the *FRA* and the *BA*.

The problem identified is that the *BA* did not consider to take a whole loop or more at the inner ring to establish the communications. Hence, when $q \approx p/2$ the distances obtained were not the optimal ones, this problem was already identified at [4]. To solve this problem it is unavoidable to try out this looping around when implementing the algorithm. Instead of calculating directly the values of D_{in1} and D_{in2} it tries all the possible values to find the best in each direction. This loop range is proportional to p.

Then when the loop has found all the values of D_{in1} and D_{in2} and the best option is considered, the values of the rest of the necessary parameters are calculated to obtain the best DT_{in1} and DT_{in2} to compare them with the value of DT_{out}, just as the previous cases.

At the time that the three best values are calculated, compared and the optimal is chosen, the link related to that option is selected to forward the packet. Then the procedure starts all over again from the next node until the destination is reached.

To better understand the effect of using the loop at Fig. 6 the path obtained for the *FRA*, *BA*, and *OPA* are represented. The configuration chosen is N2R (12,5) since it was one of the problematic situations found.

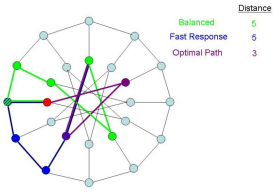


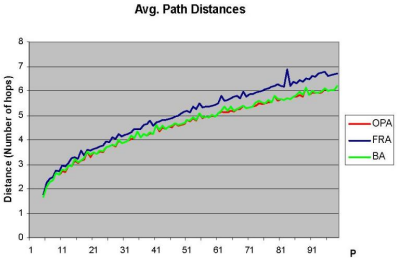
Fig. 6. N2R(12,5) Communication Example

5 Analysis

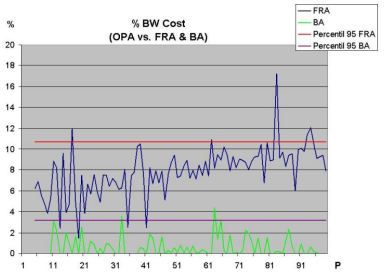
In this section the comparison of the results of the different cases is discussed. The different natures of traffic demand different QoS levels and different requirements [4]. Therefore, two parameters, among others, are analyzed to study the feasibility of each algorithm depending on the traffic nature:

- Path Length: The fact of obtaining shortest paths as possible is related with the availability of the network, the BW cost of the packet routing and the network installation cost. Minimizing the paths, reduces as much as possible the probability of failure caused by links or nodes failures, there are less failure points. If the paths are shorter, in connection with the installation, it will be necessary to use less fibre to build the network (less fiber at each link to support the traffic). In the case of taking minimum routes reduces the cost in BW of the communications, the less cost the more communications are allow with the same link capacity. All these properties are explained in detail in [5].

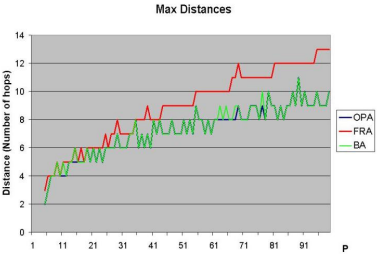
  The P2P file sharing traffic creates, in most countries, more than 50% of the total traffic. The delay is not a critical factor for this kind of traffic, therefore it would be possible to use the longer algorithm (execution time) [10]. In this way the amount of fiber required to support these services can be minimized, due to the shortest distances. This also applies to similar kinds of traffic such as ftp or SAN (Storage Area Network).
- Delay: Obtaining minimum delay at the nodes to complete a communication improves the performance of the network by reducing working tasks at the nodes [5]. A short delay is needed for the upcoming real time networks [1]. For real time traffic, such as video conference or TVoIP, the delay becomes



(a) Average Path Distances



(b) % BW Cost



(c) Maximum Path Distances

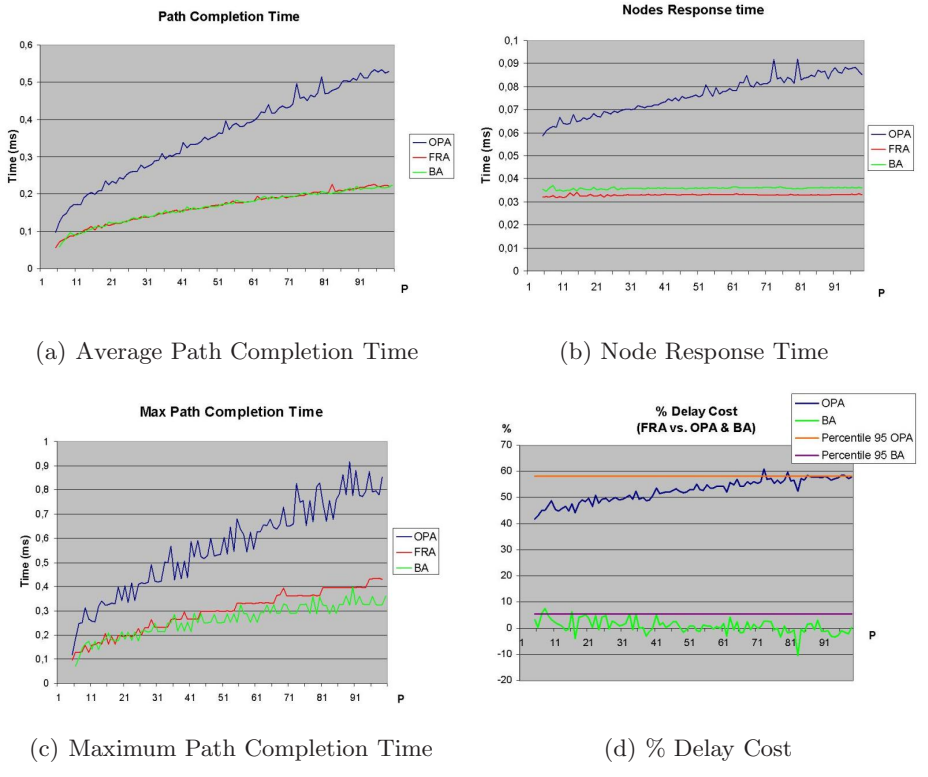
Fig. 7. Distance Graphs

a critical factor. Therefore, the method used must be able to minimize this, but with the consequence of more installation budget and more BW cost per packet if the distances are longer. The path length can impose delays as well; even though an algorithm has a fast execution, if the paths are too long, the algorithm must be executed too many times (delaying the packet more at each execution).

Hence, the three algorithms were run to obtain the average distance for a communication, diameter (maximum distance), average path completion time (the time spent at the nodes to reach the destination) and maximum path completion time[3] varying the value of p from 5 to 100 (200 nodes). The q selected for the graphics at each value of p is the one that obtains the shortest average path length. These values were obtained simulating all the possible communications (from all the nodes to all the nodes), and then the average was calculated.

[3] The execution time was obtained under the same conditions, since depending on the machine where these algorithms are executed the result will vary. It is assumed though that the proportion will be maintained. The machine used is a Genuine Intel(R) CPU T2050 @1.60 GHz (2 CPU) and 1GB of RAM.

Fig. 7(a) illustrates the average path length for the p, for the ranges of the three algorithms studied. The best option as expected is the *OPA*. The *BA* obtains very small differences for the average values, hence it could be also a possibility to consider depending on the rest of the factors. In the result for the *FRA* there is a significant difference, increasing at the same time as p.



(a) Average Path Completion Time

(b) Node Response Time

(c) Maximum Path Completion Time

(d) % Delay Cost

Fig. 8. Delay Graphs

Fig. 7(b) illustrates the relative percentage of the BW cost of using the longer solutions under the same conditions (the same network capacity), in other words, the percentage of pps (packets per second) reduction. The value of the *"Percentile 95"* [4] is represented for both cases. This value gives a pseudo maximum avoiding the maximum peaks at very special situations. The use of *BA* implies around maximum 2.3% reduction of the pps and the use of the *FRA* implies a higher reduction, around 10.7% of the maximum. These values can be applied in case of analyzing the cost of having the same throughput for the different cases. To

[4] The pth percentile is a value such that at most (100p)% of the observations are less than this value and that at most 100(1 - p)% are greater. (p is a value between 0 and 1).

obtain the same performance, the percentages given are the extra amounts of fibre needed to be installed at the links. Therefore, the increment of the budget concerning the civilian construction when implanting this kind of networks.

This analysis assumes symmetric traffic (same traffic from all the nodes to all the nodes) to obtain a general overview of the behaviour of the algorithms. With asymmetric traffic models the results will vary, e. g. if there is a high traffic between nodes close to each other, the cost of the *BA* and *FRA* will be lower since they obtain optimal, or close to optimal, paths. But if there is a higher traffic between pairs of nodes further away, then it can be assumed that the cost would be higher due to the differences on the distances to the optimal solution.

To be able to guarantee a certain QoS level for any communication the maximum values should also be considered. The Fig. 7(c) illustrates the maximum distance for the p given range. As in Fig. 7(a) the best option is the *OPA*, but noticing the small difference with the *BA*. For 93.7% of the p values the maximum value is the same. Only in very specific cases (the rest 6.3%) this value changes only by one hop. The difference with *FRA* is very significant. Only for 9.5% of the p values the maximum value is the same.

This small difference between the *OPA* and *BA* is due to the q values represented. It is assumed that when implementing a network, the best configuration is used for any number of nodes required. At the most of the N2R(p,q) there are two optimal q values. One is $q \approx p/2$, and the other is much smaller [4].

In Section 4.3 it was already identified that the problem of the *BA* with high q values. Hence, for equal results for the *OPA* distances, the smallest q value (in case there are two optimal) was used for the analysis and the graphics. If the two algorithms are executed over configurations with larger q values, the differences would be much more significant.

A priori an idea could be to add a condition to the *OPA* to decide when it has to use the loop or not. It is clear that for values of $q \ll p/2$ it will not be necessary. The problem is that there are apparently no mathematical relation between p and q to be able to give a limit for the condition. Therefore it was not implemented.

Fig. 8(a) illustrates the average path completion time for the three algorithms. The conclusion is the opposite than the path length analysis. The cost in delay of obtaining the optimal paths *(OPA)* is very high. The completion time for the other two cases is very similar; depending on the number of nodes the *BA* completes the path even faster than the *FRA*. This fact is due to the shorter paths obtained, and despite the node response time always being smaller for the *FRA*. Fig. 8(b) illustrates the Node Response Time for the three cases as a function of p.

The nodes response time at the *FRA* is independent of p and always being the *FRA* time (around 32 μs) shorter than the *BA* time(around 36 μs). The opposite situation is the *OPA*, the time is much longer than the other two cases, and increases with p which makes it scale badly.

Fig. 8(c) illustrates the maximum path completion time. Obviously, the maximum time of the *OPA* is much longer than the maximum time for the other

two cases. Out of the comparison between the *FRA* and the *BA* an interesting conclusion is obtained. For 88% of the p values at the *BA* the maximum delay for a communication is the minimum among the possibilities. For $p < 50$ (100 nodes) the maximum times are similar, being in some situations shorter than the *BA* and in some situations longer. But for $p > 50$ the maximum time is always longer with the *FRA*, and in addition to the fact that it always obtains the longest paths, this algorithm can be discarded for these p values.

The last result obtained is shown in Fig. 8(d) which represents the cost in delay of using the *BA* or *OPA* versus the *FRA*. This cost is represented as a percentage, depending on the value of the *FRA* delay, the values for the other two cases is "$(1 + 0.a) * FRA_{delay}$" being a percentage. As for the distance analysis the percentile 95 is represented for both cases. The *OPA* values, as in the rest of the execution time analysis, are extremely high compared with the other two cases. At the *BA* in some cases the percentage is even negative, (no cost, delay reduction) since it takes less time to reach the destination.

6 Conclusion

This paper introduced and compared different algorithms for topological routing in N2R networks, which can increase the reliability and availability compared to traditional routing schemes. At this first stage the schemes are still not adaptive, and as such further research is needed before an implementation can take place.

Three different algorithms were proposed, a modification of a previous algorithm and two new proposals. They all achieved the goal of obtaining paths for any communication between nodes. Those different algorithms have much different characteristics, but none of them obtains the optimal result for all the parameters tested. Thus, it is a matter of trade-offs between parameters.

There is a possibility of an algorithm which, at the end of the communication between any pair of nodes, obtains the shortest path for all the possible N2R configurations, Optimal Path Algorithm *(OPA)*. But the cost in delay for this case is the highest among the possibilities, probably at some circumstances unacceptable.

On the other hand there is no possibility of an algorithm that obtains the smallest delay in all situations. Related to this factor the Fast Repose Algorithm *(FRA)* and the Balanced Algorithm *(BA)*obtain the best result, depending on the number of nodes. The BW cost for the *FRA* is higher than the cost for the *BA*, and the maximum value for the path distances makes it an improbable solution. In the same way the previous algorithm would have the same BW cost, hence even if there is a reduction of a few instructions on the algorithm of the middle nodes, due to the flags use to identify the path at some situation it could be discarded.

The *BA* does not obtain the best results possible for delay and path distance in all the situations, but the difference with those minimum values is not very costly (the most around 2.3% in BW cost and and 5.3% in delay cost[5]).Another

[5] Percentile 95 values.

important characteristic is the maximum values for the two factors (paths length and completion time). Both are very similar to the optimal, therefore almost the same QoS can be guaranteed (only in 12% of the cases for the delay and 6.3% of the cases for the path distances, the minimum values are not obtained).

These results demonstrate the availability improvement over the previous studies about the N2R topological routing. The average path lengths and the maximum path lengths have been reduced. The budget for the installation of the fiber can be reduced if this new method is applied over the previous algorithm.

But there is still work to do in order to improve the reliability of the network. A number of tasks mentioned in [4] are not yet solved. There might be some possible methods to be able to route a packet using a second path (in case of a failure of any element of the first one) to provide a reliable network system. This idea is explained at Section 7.

7 Further Work

As an introduction to further work three theoretical solutions are proposed to be simulated, tested and proved in the next step of this topological routing issue, in order to offer adaptive path redundancy and fault tolerance.

- Link Restoration: The packets are routed normally through the network. In case a next hop in the communication is not reachable, the packet is sent directly using the only link available (one of the other two is the failure and the last one is the arrival link). At the next steps the shortest path is followed considering that the packet cannot return to a previous node. This method is fast but at the same time, some of the communications could take too long unnecessary paths.
- Path Restoration: When a failure occurs, a node detects it when it tries to send a packet through that failure. At that moment the node sends a warning message to the source node and immediately the source uses another algorithm that can route the packet using an independent (disjoint) second path. This solution would increase the routing time at the nodes due to the complexity of the algorithm, but with shorter solutions (in distance). For this method additional information about the source is needed. Another problem is the packet loss while the source does not receive the warning message, therefore this method must be improved.
- Combination: Theoretically the best option found is the combination of both of the previous proposals. At the transition time, while the source does not receive the warning message, the solution used must be the “Link Restoration” and at the moment that the network is stabilized the “Path Restoration” solution.

These proposals must be analyzed as the first path methods, the distance and delays, and simulated to obtain clear results of the possible solutions. These analyses should take into account that different failure characteristics may impose different requirements to the restoration schemes used.

References

1. Pedersen, J.M., Knudsen, T.P., Madsen, O.B.: Topological Routing in Large-Scale Networks. In: IEEE/ICACT 2004, Korea, February 2004, p. 912 (2004), Available: http://ieeexplore.ieee.org/iel5/9073/28787/01293001.pdf
2. Martel, C., Van Nguyen: Analyzing Kleinberg's (and other) small-world Models. In: Annual ACM Symposium on Principles of Distributed Computing archive Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing, St. John's, Newfoundland, Canada, pp. 179–188 (2004), ISBN:1-58113-802-4
3. Watts, D., Strogatz, S.: Collective dynamics of small-world networks. Nature 393, 440–442 (1998)
4. Pedersen, J.M., Riaz, M.T., Madsen, O.B.: A Simple, Efficient Routing Scheme for N2R Network Structures. In: IT&T Annual Conference 2005, Cork, Ireland, pp. 69–80 (2005)
5. Grover, W.D.: Mesh-Based Survivable Networks, Options and Strategies for Optical, MPLS, SONET and ATM Network, 1st edn. Prentice Hall PTR, Englewood Cliffs (2003)
6. Sanchez, M.G.S., Rumin, R.C., Gutierrez, J.M.: Triple Play Network Modelling for Spain. MsC Thesis. Control Department Aalborg University (June 2006)
7. Pedersen, J.M.: Structural Quality of Service in Large-Scale Network. PhD Thesis. Control Departament, Aalborg University (April 2005)
8. Jorgensen, T., Pedersen, L., Pedersen, J.M.: Reliability in single, double and N2R ring network structures. In: CIC 2005. The International Conference on Communications in Computing, Las Vegas, Nevada, United States, June 2005, pp. 2–4 (2005)
9. Madsen, O.B., Knudsen, T.P., Pedersen, J.M.: SQoS as the Base for Next Generation Global Infrastructure. In: Proc. of IT&T 2003, Information Technology and Telecommunications. Annual Conference 2003, Letterkenny, Ireland, October 2003, pp. 127–136 (2003), Available: http://www.kommunikation.aau.dk/ddn/Filertildownload/SQoSastheBaseforNextGenerationGlobal.pdf
10. Karagiannis, T., Papagiannaki, K., Faloutsos, M.: BLINC: Multilevel traffic classification in the dark. ACM SIGCOMM 35(4), 4 (2005), Available: http://www.acm.org/sigs/sigcomm/sigcomm2005/paper-KarPap.pdf

Author Index