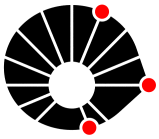
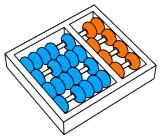
,





Anais do XV Workshop de Teses, Dissertações e Trabalhos de Iniciação Científica do IC Unicamp

Technical Report - IC-20-08 - Relatório Técnico
December - 2020 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Anais do XV Workshop de Teses, Dissertações e Trabalhos de Iniciação Científica
Instituto de Computação - Unicamp

Apresentação

Este relatório técnico contém os resumos de 13 trabalhos cujos artigos foram autorizados a serem publicados no XV Workshop de Teses, Dissertações e Trabalhos de Iniciação Científica (WTD)[1], do Instituto de Computação (IC) da Universidade Estadual de Campinas (Unicamp), edição 2020.

O XV Workshop ocorreu entre os dias 09 e 11 de Dezembro de 2020 e contou com cerca de 129 participantes, entre ouvintes e apresentadores de trabalhos. O evento contemplou 13 short papers, 29 lightning talks e 50 produções em vídeo. Aos alunos foi dada a possibilidade de escolher a forma de apresentação (lightning talk ou produção em video), bem como escolher se desejasse publicar ou não seu trabalho nos anais do evento. A publicação dos resumos sob forma de relatório técnico tem por objetivo divulgar os trabalhos em andamento e concluídos e registrar, de forma sucinta, o estado da arte da pesquisa do Instituto de Computação no ano de 2020.

Neste ano ocorreram 5 palestras em 3 dias de evento. A primeira, intitulada "Pós-graduação: Avaliação, Qualidade do Programa e Publicações", foi proferida pelo Prof. Dr. José Palazzo Moreira de Oliveira, Professor do Instituto de Informática da Universidade Federal do Rio Grande do Sul - UFRGS. A segunda, intitulada "Um Tour pela Teoria da Computação", foi proferida pelo Prof. Dr. Rafael Schouery, do Instituto de Computação, IC, Unicamp. A terceira intitulada "Como (não) Arruinar a Redação de um Texto Científico", foi proferida pela Prof. Dra. Claudia Maria Bauzer Medeiros, professora do Instituto de Computação, IC, Unicamp.

A quarta palestra, "Olhe, mire, atire e acerte! Suas escolhas determinam o seu sucesso", foi proferida pelo Dr. Allan Pinto, pós-doutorando na Faculdade de Educação Física, FEF, Unicamp. A quinta palestra, "Misinformation Dissemination on the Web: From E-mail to WhatsApp Groups", foi proferida pela Profª. Dra. Jussara M. Almeida do Departamento de Ciência da Computação, da Universidade Federal de Minas Gerais - UFMG.

Também foi oferecido o minicurso intitulado "Tecnologia Biométrica no Dia-a-dia", ministrado pela empresa Griaule e uma mesa redonda intitulada "Contribuições da Computação no combate à COVID-19', composta por Dr. Allan Pinto (FEF/UNICAMP), Prof. Dr. Anderson Rocha (IC/UNICAMP), Prof. Dr. Ricardo da Silva Torres (NTNU, Norway) e Prof. Dr. Paulo Silva e Silva (IMECC/UNICAMP).

Agradecemos aos alunos que participaram do evento, em particular àqueles que se dispuseram a apresentar seus trabalhos, seja oralmente ou em short papers, bem como aos orientadores que os incentivaram a fazê-lo. Agradecemos, também, aos professores, alunos de mestrado, doutorado e pós doutorado do IC que compuseram as bancas de avaliação dos trabalhos e aos colaboradores da secretaria que apoiaram a organização do evento. Agradecemos ao Professor Doutor Anderson de

[1] https://www.ic.unicamp.br/wtd/2020/

Rezende Rocha, diretor do IC, e a Professora Titular Cecília Mary Fischer Rubira, coordenadora da Pós-Graduação, pelo forte incentivo, apoio e patrocínio ao evento.

Agradecemos às empresas NeuralMind, Matera, ProFusion, Griaule, GoBots e BuildBox, que engrandeceram o evento como patrocinadoras.

Finalmente, agradecemos imensamente aos alunos do programa de Pós-Graduação do IC que efetivamente organizaram o evento e que são coeditores deste relatório – André Gomes Regino, Enio de Jesus Pontes Monteiro, Gustavo Caetano Borges, Letícia Bomfim e Sheila Venero. A eles dedicamos o XV Workshop de Teses, Dissertações e Trabalhos de Iniciação Científica do Instituto de Computação da Unicamp.

Prof. Julio Cesar Dos Reis
Profa. Esther Luna Colombini
Profa. Juliana Freitag Borin
Coordenadores do XV WTD
Professores do Instituto de Computação - Unicamp

Sumário

1 Programação

Apresentamos a programação e algumas estatísticas do XV Workshop de Teses, Dissertações e Trabalhos de Iniciação Científica (WTD) do Instituto de Computação (IC) da Unicamp.

Nesta edição, tivemos 3 dias de evento. No primeiro dia (Figura 1) tivemos a abertura, 4 sessões de lightning talks e 4 palestras. No segundo dia (Figura 2) tivemos 3 sessões de lightning talks, 1 curso e 2 palestras. No terceiro e último dia (Figura 3) tivemos 2 sessões de lightning talks, 4 palestras e 1 mesa redonda, além do encerramento do WTD.

O evento contou também com sorteio de inúmeros brindes, fornecidos pelos patrocinadores do evento.

Os autores do WTD que se inscreveram na modalidade de vídeo compartilharam com o evento os vídeos relacionados as suas pesquisa, com duração de 10 a 15 minutos. Ao todo, 47 vídeos foram submetidos. Os 3 vídeos que tiveram mais visualizações e votações (votação via formulário do evento - peso 5; quantidade de likes no vídeo - peso 3; quantidade de visualizações - peso 2) foram premiados (Figura 4).

O evento foi finalizado com a sessão de brindes e premiações.

Dia 1 (09/12/2020)				
	Aluno(a)	Trabalho	Orientador(a)	Docente
09:30-10:00	*Abertura*			
10:00-11:30	*Palestra: Pós-graduação: Avaliação, Qualidade do Programa e Publicações - Prof. Dr. José Palazzo Moreira de Oliveira*			
Sessão 1 - 11:30 - 12:00				
11:30-11:40	Carla Doris Cardoso Cusihuall	Parallelization of applications using OmpCluster	Hervé Yviquel/Guido Araújo	Leonardo
11:40-11:50	Daniela Maria Casas Velasco	DRSIR: Routing in Software-Defined Networks based on Deep Reinforcemen	Nelson Luis Saldanha da Fonseca	Rubira
11:50-12:00	Eduardo de Souza Gama	Future Generation Multi-tier Video Streaming Fog-Cloud Computing	Luiz Fernando Bittencourt	
Sessão 2 - 12:00 - 12:30				
12:00-12:10	Eduardo Seiti Ito	Identidade Auto Soberanas para setor elétrico	Luiz Fernando Bittencourt	Rubira
12:10-12:20	Frances Albert Santos	Automatic Extraction of Urban Outdoor Perception from Geolocated Free Te	Leandro A Villas	
12:20-12:30	Giovane de Morais	Virtualização completa, paravirtualização e virtualização leve: comparativo d	Luiz Fernando Bittencourt	
Almoço				
14:00-15:35	*Palestra: Um Tour pela Teoria da Computação - Prof. Dr. Rafael Schouery*			
15:35-15:50	*Palestra: A formação na prática: desenvolvedores full stack - ProFUSION*			
Sessão 3 - 15:50 - 16:30				
15:50-16:00	Guilherme Lima Hernandez R	Avaliação de desempenho do protocolo HTTP 3.0	Nelson Fonseca	Leonardo
16:00-16:10	Joahannes Bruno Dias da Cos	Gerenciamento de Nuvens Dinâmicas em Ambientes de Redes Veiculares	Leandro Villas	Rubira
16:10-16:20	Juliane Regina de Oliveira	Melhora de Qualidade de Dados de Aplicações IoT	Lucas Francisco Wanner	
16:20-16:30	Jose Italo da Costa Silva	Inteligência Artificial aplicada à criação e implementação de planos de adap	Cecilia Mary Fischer Rubira	Eliane
Sessão 4 - 16:30 - 17:00				
16:30-16:40	Marcelo Claudio Sousa Araújo	CMFog	Luiz Fernando Bittencourt	Rubira
16:40-16:50	Matteus Vargas Simão da Silv	Federated Learning para estimativa de tráfego de veículos em tempo real	Luiz Fernando Bittencourt	
16:50-17:00		CANCELADO	CANCELADO	Eliane
17:00-17:15	*Palestra: Griaule: tecnologia biométrica no dia a dia - Griaule*			

Figura 1: Programação do dia 1 do WTD

Dia 2 (10/12/2020)				
	Aluno(a)	Trabalho	Orientador(a)	Docente
Sessão 1 - 09:30 - 10:00				
09:30-09:40	Wellington Viana Lobato Junior	A Cache Strategy for Intelligent Transportation System to Connected Autonor	Leandro Aparecido Villas	Rubira
09:40-09:50	Tomás Silva	Criptografia Pós-Quântica: Reticulados, Códigos e o Estado da Arte	Ricardo Dahab	
09:50-10:00	Ana Clara Zoppi Serpa	Criptografia de Caixa Branca e ataques de canal lateral adaptados para softwa	Ricardo Dahab	
10:00-12:00	Curso de Biometria Griaule			
12:00-12:15	*Palestra: Buildbox - Além do código - Buildbox*			
Almoço				
14:00-16:00	*Palestra: Como (não) Arruinar a Redação de um Texto Científico - Profª. Dra. Claudia Maria Bauzer Medeiros*			
Sessão 2 - 16:00 - 16:30				
16:00-16:10	José Nascimento	Caso Paraisópolis - Análise de Dados de um Evento Real	Anderson Rocha	Rubira
16:10-16:20	Juan Albarracín	Modelos Generativos Profundos para Aprendizado de Representações Isolad	Adín Ramírez Rivera	
16:20-16:30	Leonardo de Melo João	Um arcabouço para estimativa de saliência em múltiplas iterações em diferer	Alexandre Xavier Falcão	
Sessão 3 - 16:30 - 17:00				
16:30-16:40	William Dias	Cross-dataset emotion recognition from facial expressions through convoluti	Anderson Rocha	Rubira
16:40-16:50	Gabriel Oliveira dos Santos	#PraCegoVer: Audiodescrição Automática de Imagens	Sandra Avila e Esther Colombini	
16:50-17:00	Samuel Gomes Fadel	Principled Interpolation in Normalizing Flows	Ricardo da Silva Torres	
17:00-17:15	*Palestra: BERTimbau - BERT treinado para o Português do Brasil - Contribuição da NeuralMind para a comunidade - NeuralMind*			

Figura 2: Programação do dia 2 do WTD

Dia 3 (11/12/2020)				
	Aluno(a)	Trabalho	Orientador(a)	Docente
Sessão 1 - 09:30 - 10:00				
09:30-09:40	Gabriel Capiteli Bertocco	Unsupervised Domain Adaptation	Anderson Rocha	cmbm
09:40-09:50	Geise Santos	Manifold learning for gait recognition using motion sensor data	Anderson Rocha	
09:50-10:00	Gustavo Caetano Borges	A literature review about usage of metadata standards and repositories	Claudia Maria Bauzer Medeiros	
10:00-11:30	*Palestra: Olhe, mire, atire e acerte! Suas escolhas determinam o seu sucesso - Dr. Allan Pinto*			
Sessão 2 - 11:30 - 12:10				
11:30-11:40	Felipe de Castro Belém	Geração de Superpixels pela Floresta Geradora Iterativa usando Informação de	Alexandre Xavier Falcão	
11:40-11:50	Felipe Nunes Gaia	Assurance Cases Applied to Dependability of Emergent Behaviors in System-of-	Cecilia Mary Fischer Rubira	
11:50-12:00	Gustavo Eloi de Paula Rodrigues	Detectando maus-usos de métodos criptográficos com aprendizado de máquir	Ricardo Dahab	
12:00-12:10	Jordão Bragantini	Segmentação Interativa de Imagens no Espaço das Características	Alexandre Xavier Falcão	
12:10-12:25	*Palestra: GoBots - Gerando conhecimento de perguntas e respostas - GoBots*			
Almoço				
14:00-16:00	*Palestra: Misinformation Dissemination on the Web: From E-mail to WhatsApp Groups - Profª. Dra. Jussara M. Almeida*			
16:00-17:00	Mesa redonda: Contribuições da Computação no combate à COVID-19			
17:00-17:15	*Palestra: Matera e o futuro do Mercado financeiro - Matera*			
17:15	*Encerramento*			

Figura 3: Programação do dia 3 do WTD

- 1º Maria Vitória Rodrigues - (16 / 215 / 523) 177,1
  - Serviço de Sugestão de Rotas para Carrinheiros na Coleta Seletiva de Materiais Recicláveis
- 2º Tomás dos Santos Rodrigues - (55 / 45 / 200) 81
  - Um estudo sobre Reticulados e seus Limitantes
- 3º Raysa Masson Benatti - (22 / 19 / 107) 38,1
  - Revealing Gender Biases in Court Decisions with Natural Language Processing

(Votos : Form - 50% / Likes - 30% / Views - 20%)

Figura 4: 3 Primeiros Colocados do Concurso de Vídeos do WTD

2 Estatísticas

Apresentamos as estatísticas colhidas em relação ao teor dos trabalhos apresentados durante o evento. A Figura 5 mostra a divisão dos trabalhos dentre os 3 tipos disponibilizados de submissão. A Figura 6 mostra a quantidade de alunos de graduação, mestrado e doutorado que apresentaram seus trabalhos. Por fim, a Figura 7 mostra as áreas de concentração dos trabalhos.

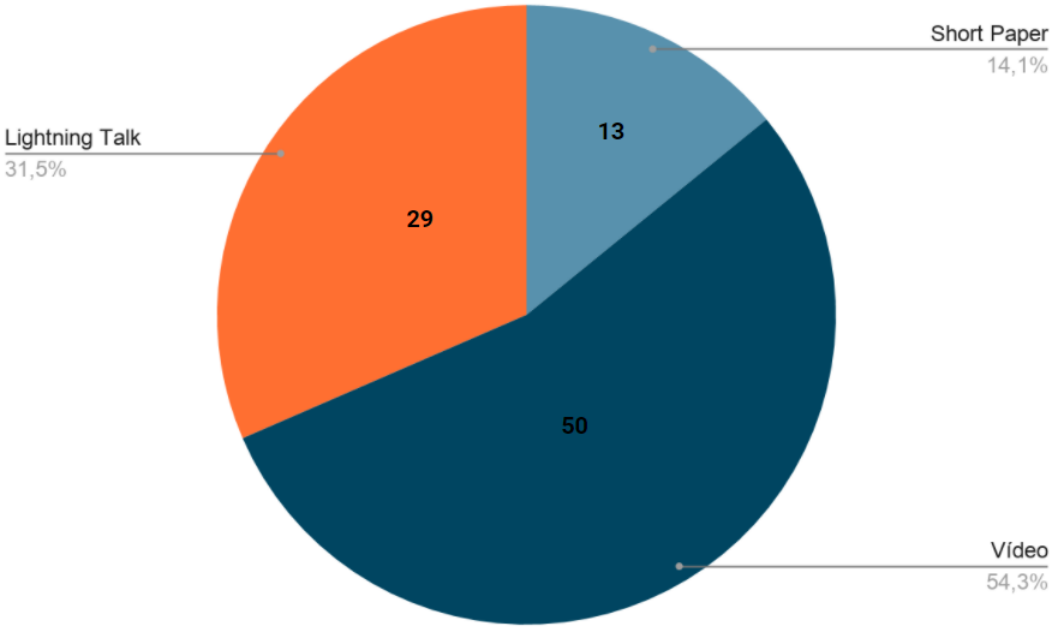


Figura 5: Tipos de Apresentações

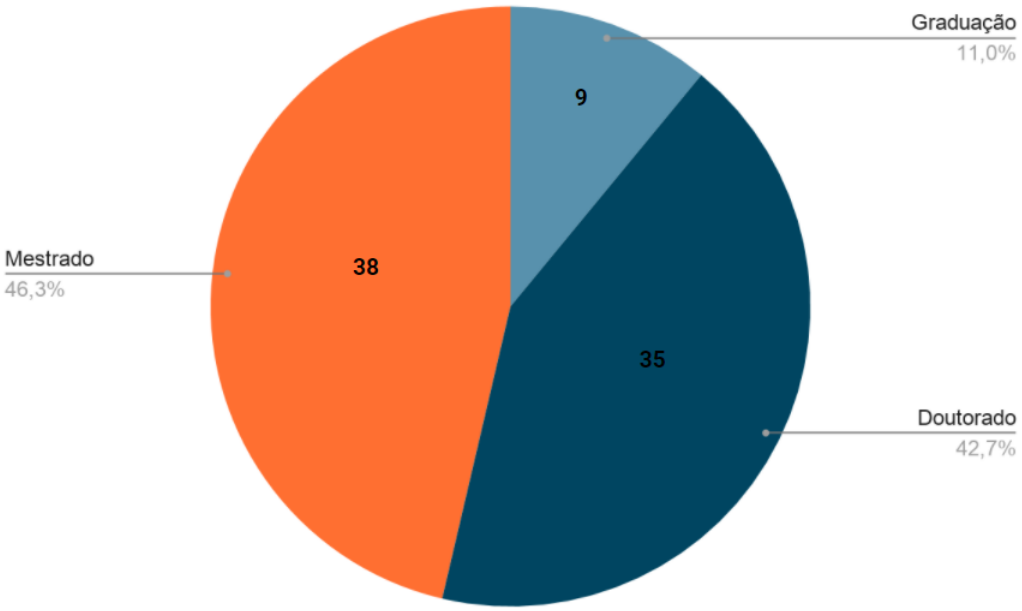


Figura 6: Titulação dos Alunos

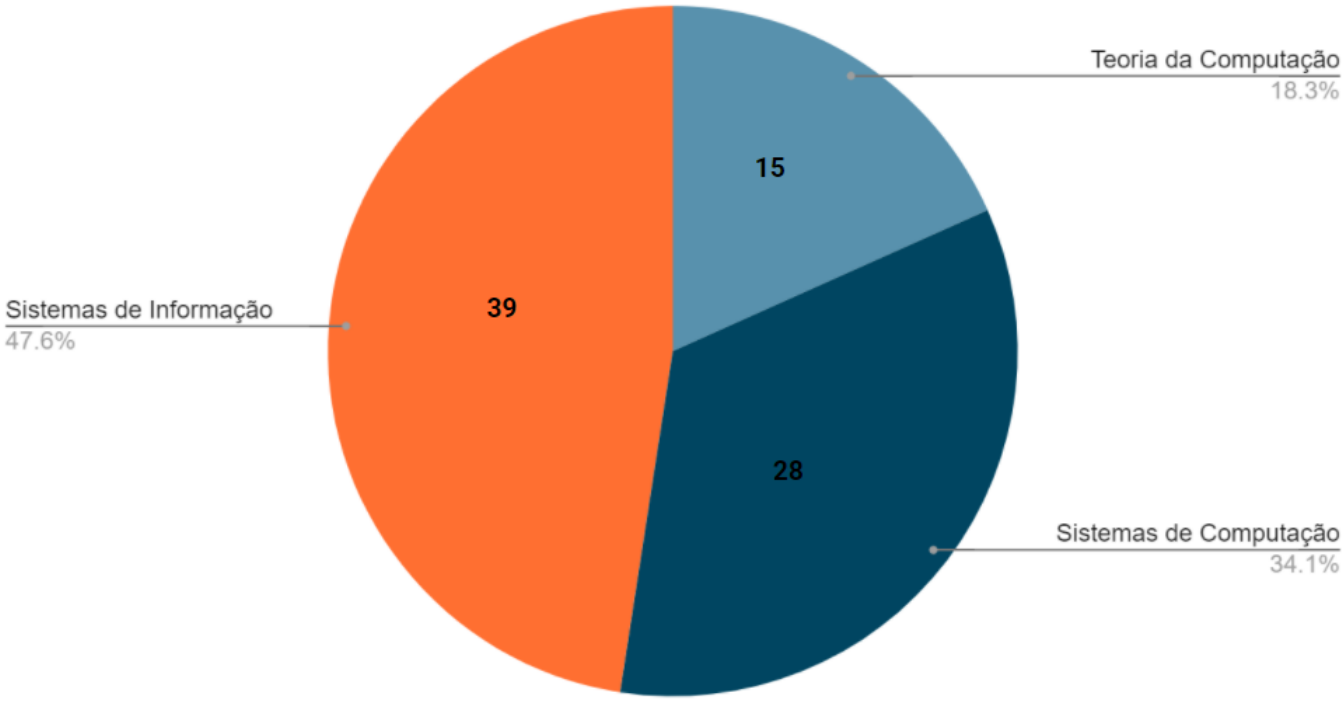


Figura 7: Áreas de Concentração

A Figura 8 apresenta a nuvem com termos mais frequentes dentre os trabalhos. Dentre esses termos, encontramos “networks”, “learning”, “software” e “models”.

Figura 8: Termos mais Frequentes

3 Short Papers

An MLIR Dialect for Dataflow Networks

Pedro Ciambra[1], **Hervé Yviquel**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Campinas – SP – Brazil

ra137268@dac.unicamp.br, herve.yviquel@ic.unicamp.br

Abstract. *MLIR is a new compiler framework focused on the creation and manipulation of intermediate representations; "dataflow" is a family of models of computation that allow for intuitive parallelism from design time. This project consists of using MLIR in the development of a dataflow compiler, in particular static and quasi-static Models of Computation, and of its validation against signal processing workloads.*

Resumo. *O MLIR é um novo framework de compilação dedicado à criação e manipulação de representações intermediárias; "dataflow" é uma família de modelos de computação que permite programação paralela de uma maneira intuitiva. Este projeto consiste em usar o MLIR para desenvolver um compilador de dataflow, particularmente para modelos estáticos e quase-estáticos, e em validar os produtos do compilador em aplicações de processamento de sinais.*

1. Introduction

Nowadays, parallelization is the most effective way to scale the performance of computing systems. With the slowing of improvements to processor frequencies, the only way to increase the workload is to split it up among several processors. However, this can be delicate, manual programming work that requires considerable specific knowledge and depends heavily on the application and hardware in question.

A programming paradigm that addresses this issue is the dataflow network. It is a layer of abstraction that models data transformation as a directed graph of actors and queues, making both data and task parallelism considerably easier to identify and optimize in an automated fashion. This means that, once described as a dataflow network, a program can be easily and optimally converted into a wide variety of targets and architectures, including classic von Neumann architectures, acceleration hardware such as GPUs and TPUs, and distributed systems. However, the currently available dataflow frameworks are numerous and don't offer much in terms of interoperability, having generally been designed for domain-specific purposes (traditionally, data signal processing). As there is no universally recognized industry standard for dataflow programming yet, there is still significant resistance to its adoption.

In this work, we plan to create a dataflow design pipeline that is integrated with a mainstream compiler framework (LLVM). In particular, we adopt the recently released MLIR, a toolset for the creation of modular intermediate representations that allows for powerful manipulation of abstractions inside of a compiler. MLIR's focus on compatibility and reuse is especially suited for our use-case, as it matches nicely with the high versatility of dataflow networks.

2. Dataflow networks

Dataflow is a family of models of computation that addresses the demand for abstract representation of parallel programs. It is agnostic to the underlying parallelization mechanism, being applicable in a broad range of situations. It is related to the similar *stream processing* and *reactive programming* paradigms.

Dataflow models represent algorithms as directed graphs, where nodes ("actors") represent processing steps and edges represent queues (first-in-first-out data structures, or FIFOs) that transport data between each processing step.

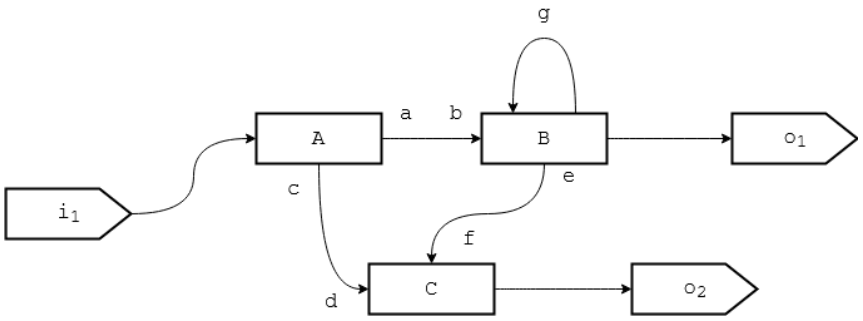


Figure 1. An example of dataflow network (specifically, a SDF graph, with fixed data rates on each port)

Each actor represents an operation that can be activated ("fired") to do work. Each firing consumes a quantity of values ("tokens") from its incoming edges and produces output tokens into its outgoing edges. These actions are regarded as black-boxes, and treated as indivisible units of work. By scheduling these actors to fire in a suitable order, the inputs flow through the graph from edge to edge as they are transformed, reaching the sink nodes as the final output of the algorithm. As this representation naturally encodes the data dependencies between each part of the algorithm, the task of selecting parallelizable regions becomes simple enough that it can be done automatically, either by a dynamic runtime or statically at compilation time.

There are different levels of generality when implementing dataflow systems. Each successive level increases the expressiveness and flexibility of the system, but sacrifices some property that could be taken advantage of at compile time. These levels have been formalized as the following Models of Computation (MoCs).

These are roughly split into three categories: *static*, *quasi-static* and *dynamic*. Static MoCs are the most constrained, but allow for efficient scheduling at compile-time, minimizing runtime overheads; quasi-static MoC's relax some of these constraints, allowing for some limited reconfiguration during execution; and dynamic MoCs are mostly unconstrained and flexible enough to adapt to any input, at a runtime cost.

In this project, we are mostly interested in the static and quasi-static models, particularly *static dataflow* (SDF) [Lee and Messerschmitt 1987], which fixes the data rates in each node's port, and *parameterized/interfaced dataflow* (piSDF) [Desnos et al. 2013], which allows for periodic reconfiguration and

There are a number of existing projects based on dataflow networks. Some of the existing user-facing textual languages are LUSTRE [Halbwachs et al. 1991], SIGNAL

[LeGuernic et al. 1991], CAL [Eker and Janneck 2003], DIF [Hsu et al. 2004] and HoCL [user *jserot* 2019]. Of these, CAL and DIF noteworthy, as they are supported by multiple different projects.

Some compiler frameworks and runtimes are the Ptolemy project [Eker et al. 2003], Grape-II [Lauwereins et al. 1992], Preesm and SPIDER [Heulot et al. 2014], Orcc [Yviquel et al. 2013] and StreamIt [Thies et al. 2002]. And, while not being considered a dataflow framework in the traditional sense, the TensorFlow [Abadi et al. 2016] neural network framework shares many properties with them.

3. MLIR

The MLIR [Lattner et al. 2020] project is part of the LLVM project [Lattner 2002], an industrial-grade compiler infrastructure and toolset. One of LLVM's main features is LLVM IR, a generic Intermediate Representation (IR) for von Neumann architectures that is used by many compilers, including Clang, Rust, Swift, Julia and others. It became apparent, however, that while these compilers did in fact make full use of the LLVM toolset, there was still a lot of repeated work done by each project, which indicated that LLVM IR is not generic enough and that a more expressive IR would be useful; furthermore, LLVM IR is designed to be compiled into traditional processor targets such as x86 and ARM, and therefore has poor support to accelerator hardware.

Thus, the MLIR project (Multi-Level Intermediate Representation) was created; it allows for arbitrarily-defined domain-specific IRs to operate between each other with a common toolset. Each subset of the IR is called a "dialect", and, in contrast with traditional IRs, they can carry and transmit as much abstract data as is convenient. For instance, instead of transforming ("lowering") a matrix multiplication into a nested loop and then somehow detecting that and transforming it back into accelerator bytecode, the matrix multiplication can stay in abstract notation the whole time until it is picked up by the target's dialect, without having to annotate the source code with vendor-specific syntax or function calls and without having to process low-level IR.

This allows for a great amount of modularity, as the hardware optimization of the matrix multiplication itself can be deferred to each vendor, and the programmer can focus on the semantics of the problem instead. Meanwhile, all of the tools used for the transformation and manipulation of the source code and dialects remain the same across all levels of the compilation, allowing for high-quality, easily maintainable and extendable compilers. And, as the user no longer depends on a specific vendor's toolset to express a generically parallel program, they are enabled to adopt a more expressive declarative programming style.

This is particularly useful for domain-specific languages, as each project can define its own dialect and defer the implementation of other layers of abstraction to a different dialect. A popular example is the domain of neural networks; by using a MLIR dialect, the modeling of the neural network's architecture can be decoupled from the implementation of the computations themselves (in fact, the MLIR project has been directly worked on by the TensorFlow team).

4. Our project

Our project consists of a compiler infrastucture for dataflow networks that taps into MLIR's ecosystem. We hope that, by integrating dataflow tools with a industry-standard general purpose compiler, we can lower the barrier of entry to the advantages that dataflow programming can offer. MLIR is especially well-suited for that, since it already facilitates integration into many different targets and provides powerful language processing tools.

The resulting workflow is a compiler that can generate performant executable code for a wide range of targets and target configurations, working directly from an abstract, human-readable graph representation. Little specific knowledge about parallel programming will be required from the user, and extensions or integration into other tools will be straightforward to develop. Hopefully, this will allow for more agile HPC and DSP development, especially for heterogenous targets.

The planned contributions are as follows:

A DIF parser. Our dataflow networks should be designed in a friendly user-facing interface. DIF is a good pick for our input format, as it is widely supported by current dataflow applications and covers all abstractions we will need.

A dataflow dialect for MLIR. Once a suitable DIF file is available, it needs to be translated into MLIR. For this, we propose a dialect for network description that supports actors, ports and edges. It should have feature parity with the subset of DIF that we will use (specifically, the features necessary for SDF and piSDF support).

MLIR's nested regions are designed for describing rooted trees, such as ASTs; meanwhile, dataflow networks are directed graphs. As there is no unique way to serialize a graph, some convention must be decided on to describe the actors and the edges that connect them.

One possibility is to use a declarative approach where each graph has a region that describes actors, connected edges and ports. This structure is similar to other graph description languages, including mentioned dataflow languages such as DIF and HOCl. Differently from those languages, however, the actor implementations can be described in any dialect, as long as it can be lowered into a function call with the appropriate signature. The similarity also means that existing dataflow algorithms can be easily translated into the dialect and vice-versa.

SDF and piSDF schedulers. To generate a schedule for a given dataflow network, a number of preprocessing steps is required. First, it is necessary to ascertain if the network is compatible with SDF or piSDF; this requires an analysis of the actors' port rates, to ensure that a static schedule is possible. For SDF and cSDF networks in particular, there is a fixed minimum feasible amount of memory, and increasing the amount of memory in discrete steps allows for more efficient schedules. These configurations are knowable at static time. Methods for this analysis are well detailed in [Lee and Messerschmitt 1987] and [Desnos et al. 2013].

A lot of data processing happens on memory-constrained devices nowadays, considering the rise of IoT and SoC use-cases; in such occasions, the trade-off between compute and memory changes. To address that, we also plan to add a compilation option that prioritizes memory efficiency instead of performance.

By writing these steps to work directly with the MLIR dialect, we can keep the process flexible for future extension and implementation of new MoCs and targets. The SDF analysis is performed by means of operations on a matrix representation of the network; this matrix can be generated and stored in the IR itself, remaining available for subsequent transformations if needed.

The scheduler will be implemented in the form of a compiler pass that transforms the dataflow network representation into a lower-level dialect that contains the static schedule. This consists of lists of actors, representing the order in which they will be fired on each thread. For heterogenous targets such as GPUs, this step may diverge.

The last step will be the translation of the schedule into a compilation-ready MLIR dialect, which can be processed using the currently available tools. For the traditional von Neumann stack, this will be the "standard" MLIR dialect, which can, for instance, be lowered into LLVM IR and compiled into von Neumann architectures.

An integrated visual graph editor. We are interested in revisiting the visual graph interface approach. To achieve that, we will develop a graphical editor capable of interfacing with our compiler infrastructure, providing real-time feedback on the design process.

An exploration of neural networks on our framework. As MLIR is expected to be widely adopted by the machine learning community, we could experiment with the translation of ML dialects to our dataflow dialect and consider the effects in performance.

A translation tool for existing dataflow languages. It may be useful to allow interoperation between dataflow languages into our dialect (and back), and an analysis of the performance effects of this translation.

Schedulers for different targets. Heterogenous architectures are becoming increasingly common in the context of HPC applications, especially in the form of GPUs and TPUs. To ensure competitive production performance, it may be necessary to support these targets.

Benchmarking. We intend to benchmark our solution against the state-of-the-art by measuring its performance at some popular workloads in the field. Particularly, we are interested in the Sobel edge-detection filter, the scale-invariant feature transform (SIFT, a image recognition algorithm) and MNIST, a simple but well researched neural network.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283.

Desnos, K., Pelcat, M., Nezan, J., Bhattacharyya, S. S., and Aridhi, S. (2013). Pimm: Parameterized and interfaced dataflow meta-model for mpsocs runtime reconfiguration. In *2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pages 41–48.

Eker, J. and Janneck, J. (2003). Cal language report. Technical report, Tech. Rep. ERL Technical Memo UCB/ERL.

Eker, J., Janneck, J. W., Lee, E. A., Jie Liu, Xiaojun Liu, Ludvig, J., Neuendorffer, S., Sachs, S., and Yuhong Xiong (2003). Taming heterogeneity - the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144.

Halbwachs, N., Caspi, P., Raymond, P., and Pilaud, D. (1991). The synchronous data flow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320.

Heulot, J., Pelcat, M., Desnos, K., Nezan, J.-F., and Aridhi, S. (2014). Spider: A synchronous parameterized and interfaced dataflow-based rtos for multicore dsps. In *2014 6th European Embedded Design in Education and Research Conference (EDERC)*, pages 167–171. IEEE.

Hsu, C.-J., Keceli, F., Ko, M.-Y., Shahparnia, S., and Bhattacharyya, S. S. (2004). Dif: An interchange format for dataflow-based design tools. In *International Workshop on Embedded Computer Systems*, pages 423–432. Springer.

Lattner, C. (2002). LLVM: An Infrastructure for Multi-Stage Optimization. Master's thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, Urbana, IL. *See* `http://llvm.cs.uiuc.edu`.

Lattner, C., Amini, M., Bondhugula, U., Cohen, A., Davis, A., Pienaar, J., Riddle, R., Shpeisman, T., Vasilache, N., and Zinenko, O. (2020). Mlir: A compiler infrastructure for the end of moore's law.

Lauwereins, R., Engels, M., and Peperstraete, J. A. (1992). Grape-ii: a tool for the rapid prototyping of multi-rate asynchronous dsp applications on heterogeneous multiprocessors. In *[1992 Proceedings] The Third International Workshop on Rapid System Prototyping*, pages 24–37.

Lee, E. A. and Messerschmitt, D. G. (1987). Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, C-36(1):24–35.

LeGuernic, P., Gautier, T., Le Borgne, M., and Le Maire, C. (1991). Programming real-time applications with signal. *Proceedings of the IEEE*, 79(9):1321–1336.

Thies, W., Karczmarek, M., and Amarasinghe, S. (2002). Streamit: A language for streaming applications. In *International Conference on Compiler Construction*, pages 179–196. Springer.

user *jserot*, G. (2019). Higher order dataflow coordination language. https://github.com/jserot/hocl.

Yviquel, H., Lorence, A., Jerbi, K., Cocherel, G., Sanchez, A., and Raulet, M. (2013). Orcc: Multimedia development made easy. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 863–866.

Aprendizagem Ativa no Ensino de Programação: Integrando Autograders, Inventário de Conceitos e Análise do Impacto na Carga de Trabalho dos Docentes

Eryck Pedro da Silva[1]*, Rodolfo Jardim de Azevedo[1], Ricardo Edgard Caceffo[1]

[1]Instituto de Computação – Universidade Estadual de Campinas

{eryck.silva,rodolfo,caceffo}@ic.unicamp.br

Abstract. *This work in progress, representing a Ph.D thesis, has the purpose of using Active Learning in programming teaching for undergraduate students, with special regards to utilizing automatic grading of codes. The specific objectives are composed of the integration of Concept Inventories with automatic grading of codes while aiming at providing more feedback. In addition this research also aims at analyzing the outcome of the aforementioned methods in the instructor's total work load, expecting it to be reduced.*

Resumo. *Este trabalho em andamento, representando uma tese de doutorado, propõe a utilização de Aprendizagem Ativa no ensino de programação básica para alunos de graduação, em especial, o uso de sistemas de avaliação automática de código. Os objetivos específicos envolvem a integração de Inventário de Conceitos em conjunto com os sistemas de avaliação automática de código de forma a prover mais formas de feedback. Além disso, esta pesquisa também busca analisar como a inclusão de tais métodos impactam no trabalho total do professor, esperando sua redução.*

1. Introdução

A multidisciplinaridade das áreas de trabalho e pesquisa da atualidade tem impulsionado os currículos de vários cursos de graduação além da área de exatas a incluírem disciplinas de introdução à programação em suas grades curriculares, no entanto, essas matérias exibem um alto índice de retenção e evasão [Bosse 2020]. A alta taxa de alunos por curso, expectativas equivocadas que os alunos tem da disciplina antes de cursá-la e a grande variedade dos contextos educacionais prévios dos discentes são exemplos de causas desses índices [Walker 2017]. Além disso, Anwar (2019) menciona que características do ensino tradicional utilizado também afetam tais índices, como manter o professor como principal ator de passagem de conteúdo, deixando os alunos numa posição predominantemente passiva na aprendizagem [Anwar 2019]. Uma alternativa a este método é a Aprendizagem Ativa (*Active Learning*) [Bonwell and Eison 1991], que estabelece que os estudantes devem incluir papéis além de ouvintes somente. Atividades que promovam a construção de pensamentos para discussão, análise, síntese e avaliação são sugeridas para que haja um engajamento maior dos alunos.

*Agradecemos ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo processo 142476/2020-0.

2. Problema de Pesquisa

Autograders ou *Online Judges* são ferramentas que permitem análise automática de características de programas. Uma das primeiras versões a serem utilizadas para esta finalidade foi criada por Hollingsworth em 1960, avaliando códigos feitos por alunos em cartões perfurados [Hollingsworth 1960]. Desde então, ferramentas similares, mas com propósitos ligeiramente distintos surgiram: algumas tem foco restrito à avaliação de código; outras tem funcionalidade em promover desenvolvimento guiado por testes; e também existem as construídas especificamente para apoiar estudantes de programação [Prather et al. 2018]. Estes diversos usos permeiam o fato de que *autograders* possibilitam outros cenários no ensino e aprendizagem.

A utilização deste tipo de ferramenta, segundo Prather *et al*. (2018), com a mesma apenas avaliando a saída do programa auxilia nas correções de tarefas, poupando recursos e retirando parte da carga de trabalho do docente [Galvão et al. 2016, Ellis et al. 2019]. No entanto, se considerarmos que na disciplina de introdução a programação são ensinadas estruturas diferentes que podem ter resultados equivalentes, o professor pode desejar que o aluno resolva o exercício com uma estrutura específica para fins didáticos. Ou seja, por mais que a *saída* de programas criados de formas equivalentes esteja correta, um dos *códigos* em si não estaria, pois não utilizou a estrutura requisitada pela atividade, e um *autograder* convencional não identifica essa ocorrência.

Nome	Variável de parâmetro com valor atribuído externamente.
Descrição	Atribuir valor a uma variável de parâmetro de forma externa ao escopo da função.
Exemplo	1. def func(n): 2. n = eval(input()) 3. (...)
Justificativa	Alunos não compreendem que quando uma função é definida para um ou mais parâmetros, os mesmos são preenchidos de acordo com os argumentos que são passados na hora que a função é chamada. Dessa forma, eles acham que é preciso definir tais parâmetros por uma fonte externa, nesse caso, chamando a função *input()*.
Consequências	O valor original da variável de parâmetro será perdido.
Detecção	**Onde**: Em qualquer linha de função que receba um parâmetro. **Como**: O parâmetro é alterado de forma errada.
Prevenção	Alunos devem ser orientados a verificarem o valor de algum parâmetro dentro de uma função, com auxílio do *print()*. Se eles estiverem convencidos de que o valor do parâmetro já foi preenchido corretamente, eles provavelmente não tentarão modificá-lo.

Tabela 1. Exemplo de equívoco comum sobre parâmetros de funções e escopo identificado por Gama *et al.* (2018) [Gama et al. 2018]. Tradução nossa.

Inventários de Conceitos (*Concept Inventories*) tem o objetivo de mapear problemas de compreensão, também conhecidos como antipadrões ou *misconceptions*, sobre um determinado assunto [Caceffo et al. 2016]. Considerando a existência de *autograders*

específicos para auxílio de alunos de disciplina de programação [Prather et al. 2018], a integração do mesmo ao detectar tais problemas de compreensão pode ser explorada, novamente disponibilizando *feedbacks* mais elaborados.

A Tabela 1 representa um exemplo de um problema de compreensão listado no trabalho de Gama *et al*. (2018), descrevendo informações a respeito de alunos não considerarem que uma variável de parâmetro passada em uma função já contém um valor atribuído, achando necessário realizar essa tarefa de alguma forma e acabam atribuindo um outro valor por fonte externa ao escopo da função [Gama et al. 2018].

3. Objetivos e Trabalhos Relacionados

Os objetivos envolvem a análise do impacto da utilização *autograders*, que possuirão *feedback* aprimorado, no ensino e aprendizagem de alunos de graduação de cursos de introdução a programação, bem como a relação dessa abordagem com a carga de trabalho dos docentes. A disciplina MC102[1] da Universidade Estadual de Campinas é a preferencial para aplicação desta abordagem pois é um curso coordenado, ministrado para as engenharias e demais áreas de exatas (exceto a computação), e, por consequência, abrange um grande número de alunos por semestre (aproximadamente 600). A linguagem de programação Python 3 é atualmente ensinada nesta disciplina.

Caceffo *et al*. (2016) desenvolveram um Inventário de Conceitos mapeando problemas de compreensão através de questionários de múltipla escolha de forma a utilizá-los numa abordagem de Tutoria em Pares, técnica de Aprendizagem Ativa [Caceffo et al. 2016]. Sorva (2012) também listou problemas do mesmo tipo, independente da linguagem de programação, ao realizar uma extensa revisão de literatura [Sorva 2012], já Savage e Piwek (2019) catalogaram dúvidas de alunos em fórum virtual sobre curso de introdução à programação na linguagem Python [Savage and Piwek 2019].

Ureel e Wallace (2019) abordam a diferença entre *autograders* e *critiquers*, em que o segundo possui ferramentas com um grau de *feedback* altamente interativo, utilizando esses para identificar problemas de compreensão de programação orientada a objetos em Java [Ureel and Wallace 2019]. A pesquisa de Galvão *et al*. (2016) utilizou um *autograder* como ferramenta de apoio ao ensino e aprendizagem em disciplina de programação, e, por mais que o seu uso tenha sido estabelecido para diminuir o trabalho do professor na correção dessas novas atividades, os autores não aprofundaram suas observações em como esse fator foi desenvolvido [Galvão et al. 2016].

4. Metodologia e Resultados Iniciais

Um conjunto com dois problemas de compreensão simples de serem detectados foram idealizados para realizar uma pré-avaliação desta pesquisa. A fundamentação desta análise baseou-se no fato de que a disciplina MC102 contempla uma série de laboratórios de atividades de programação em seu currículo, abordando os diversos tópicos ensinados durante o curso. Um sistema *online* de submissão é utilizado para envio e correção dos programas, o SuSy[2].

[1] `https://www.ic.unicamp.br/~mc102/`
[2] `https://www.ic.unicamp.br/~susy/`

Para o desenvolvimento de ambos casos, as submissões finais dos alunos das 20 turmas ofertadas no primeiro semestre letivo de 2020 foram captadas do repositório gerado pelo SuSy. Os códigos foram analisados com um programa, também construído em Python 3, utilizando a biblioteca *AST*[3] que permite a inspeção e manipulação de árvores sintáticas abstratas de códigos na mesma linguagem.

4.1. Laboratório 04: Classificação de Triângulos

Este caso de detecção em código baseou-se nos conceitos de operadores booleanos com estruturas condicionais (*if-else*). De uma forma geral, um programa usando dois *if's* seguidos aninhados pode equivaler a outro que utilize apenas uma estratura condicional em conjunto com operador booleano *and*. O professor pode, por exemplo, querer verificar se o aluno entendeu o conceito de operadores booleanos ao utilizá-los em um mesmo *if*, e não aninhando condicionais consecutivos.

Uma análise foi feita comparando a utilização de *if's* de um laboratório ministrado para as 20 turmas de MC102 do período letivo de 2020-1. A tarefa trabalhava os conceitos de condicionais na classificação de triângulos[4]. Um total de 506 programas foram analisados, com os resultados sendo condensados em um histograma representado pela Figura 1.

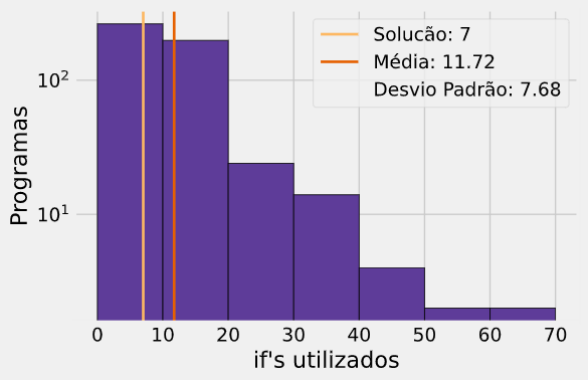


Figura 1. Histograma do total de *if's* utilizados no laboratório 4 das turmas de MC102 de 2020-1.

Os resultados mostram que embora a média da utilização de *if's* das turmas esteja próxima ao total utilizado pela solução, existem programas que chegam a utilizar quase dez vezes mais. A situação a ser detectada nesse caso poderia, por exemplo, alertar ao professor ou aluno sobre um demasiado uso de condicionais que poderiam ser evitados com o emprego de conectores lógicos.

4.2. Laboratório 14: Recursão

A verificação idealizada por este caso foi baseada no exemplo do professor analisar se um dado programa, em que é explícito o uso de funções recursivas, realmente possui tais

[3] `https://docs.python.org/3/library/ast.html`

[4] `https://www.ic.unicamp.br/~mc102/mc102-1s2020/labs/roteiro-lab04.html`

funções implementadas e utilizadas, novamente sobre a mesma situação de equivalência, dessa vez a respeito de algoritmos recursivos e não-recursivos.

O laboratório escolhido para este caso trabalhava recursão com a criação de triângulos em arte ASCII[5], onde o uso do conceito era explicitamente exigido. De forma análoga a anterior, a análise foi feita com a quantidade de declarações de funções recursivas no código, verificando funções que chamem elas mesmas pelo menos uma vez em seu escopo. Neste caso, um total de 43 programas foram processados, novamente com os resultados condensados em um histograma representado pela Figura 2.

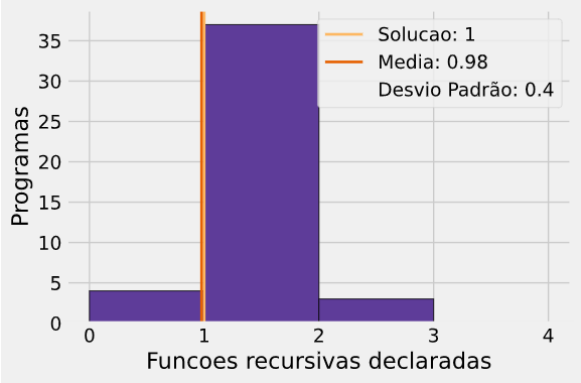


Figura 2. Histograma do total de funções recursivas declaradas no laboratório 14 das turmas de MC102 de 2020-1.

É importante notar que o número de programas avaliados muda drasticamente em comparação ao caso anterior: esse fator é consequência da natureza do laboratório 14 que foi de realização **opcional**. No entanto, mesmo com um menor número ainda foi possível notar que existem códigos que não declararam nem utilizaram nenhuma função recursiva, sendo este o indicador que poderia alertar tanto ao aluno quanto ao docente responsável pela correção.

5. Conclusões e Próximos Passos

As análises realizadas indicam que há espaço para utilização de *autograders* com *feedback* aprimorado ao detectar, por exemplo, casos em que por mais que a saída do programa esteja correta, podem existir outros fatores no código necessitando observação. Com esses alertas, o aluno poderia ser notificado sobre uma possível revisão dos conceitos da disciplina; já o professor não precisaria verificar todos os programas para encontrar os que apresentem tais características.

Os próximos passos desta pesquisa estão concentrados na elaboração e implementação da detecção de novos casos, de forma a ter um conjunto catalogado para ser testado nas turmas de introdução à programação de forma a analisar o impacto da utilização desta tecnologia em sala de aula, avaliando o *feedback* no ensino e aprendizagem dos alunos quanto na carga de trabalho dos docentes, esperando que a mesma seja reduzida devido às características exploradas com os *autograders*.

[5] `https://www.ic.unicamp.br/~mc102/mc102-1s2020/labs/roteiro-lab14.html`

Referências

Anwar, S. (2019). Impact of educational technology-based learning environment on students' achievement goals, motivational constructs, and engagement. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, ICER '19. Association for Computing Machinery.

Bonwell, C. and Eison, J. (1991). *Active Learning: Creating Excitement in the Classroom*. J-B ASHE Higher Education Report Series (AEHE). Wiley.

Bosse, Y. (2020). *Padrões de Dificuldades Relacionadas com o Aprendizado de Programação*. PhD thesis, Universidade de São Paulo.

Caceffo, R., Wolfman, S., Booth, K. S., and Azevedo, R. (2016). Developing a computer science concept inventory for introductory programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, page 364–369. Association for Computing Machinery.

Ellis, M., Shaffer, C. A., and Edwards, S. H. (2019). Approaches for coordinating etextbooks, online programming practice, automated grading, and more into one course. *SIGCSE 2019 - Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 126–132.

Galvão, L., Fernandes, D., and Gadelha, B. (2016). Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação. *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação - SBIE)*, 27(1):140.

Gama, G., Caceffo, R., Souza, R., Bennati, R., Aparecida, T., Garcia, I., and Azevedo, R. (2018). An antipattern documentation about misconceptions related to an introductory programming course in python. *Institute of Computing, University of Campinas, Tech. Rep. IC-18-19*.

Hollingsworth, J. (1960). Automatic graders for programming classes. *Commun. ACM*, 3(10):528–529.

Prather, J., Pettit, R., McMurry, K., Peters, A., Homer, J., and Cohen, M. (2018). Metacognitive difficulties faced by novice programmers in automated assessment tools. *ICER 2018 - Proceedings of the 2018 ACM Conference on International Computing Education Research*, pages 41–50.

Savage, S. and Piwek, P. (2019). Full report on challenges with learning to program and problem solve: an analysis of first year undergraduate open university distance learning students' online discussions. *The Open University, Milton Keynes*.

Sorva, J. (2012). *Visual program simulation in introductory programming education*. Aalto University.

Ureel, L. C. and Wallace, C. (2019). Automated critique of early programming antipatterns. *SIGCSE 2019 - Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 738–744.

Walker, H. M. (2017). Acm retention committee retention of students in introductory computing courses: Curricular issues and approaches. *ACM Inroads*, 8(4):14–16.

Desafios no aprendizado profundo aplicado ao projeto de anticorpos a partir de epítopos

Amaury Mausbach Filho[1], **João Meidanis**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas
Av. Albert Einstein, 1251 – Cidade Universitária, Campinas – SP, 13083–852

amausbach_filho@hotmail.com, meidanis@ic.unicamp.br

Abstract. *The design of antibodies from epitopes has been widely researched as a potential treatment for diseases that are difficult to treat by other means, such as certain viral diseases and certain types of cancer. In our project, we explored the use of deep learning for automatic design of the most critical part of an antibody, the so-called CRD region, which is is direct contact with the epitope. Challenges in this endeavor include the reduced amount of annotated data available, and lack of generalization ability for models derived from natural language translation.*

Resumo. *O desenho de anticorpos a partir de epítopos tem sido muito pesquisado como possível tratamento para doenças de difícil tratamento por outros meios, como certas doenças virais e certos tipos de câncer. Em nosso projeto, exploramos o uso de aprendizado profundo no projeto automática da parte mais crucial de um anticorpo, a chamada região CRD, que está em contato direto com o epítopo. Os desafios encontrados incluem reduzido número de exemplos bem curados, e dificuldade de generalização em modelos derivados da área de tradução de linguagens naturais.*

1. Introdução

Anticorpos são elementos criados pelo sistema imunológico adaptativo, a partir da identificação de um epítopo (segmento de uma parte externa de um antígeno) estranho ao corpo humano, com a finalidade de combate específico ao epítopo identificado. Esta especificidade decorre de sequências de aminoácidos(resíduos) em regiões de hiper–variação do anticorpo chamadas CDRs (*Complementarity-Determining Regions*) que levam a um acoplamento Ab-Ag (Anticorpo-Antígeno) eficaz e capaz de produzir a destruição do antígeno que contém o epítopo, como ilustrado na figura 1. Esta resposta imunológica é memorizada pelas células B, produtoras de anticorpos, o que permite uma resposta mais rápida do sistema adaptativo em um novo contato com o mesmo epítopo. Assim, os principais desafios na determinação da sequência de resíduos dos CDRs é garantir, simultaneamente, um forte e eficaz acoplamento ao epítopo e a estabilidade da estrutura do anticorpo. [Doan et al. 2006].

2. Objetivo

Avanços recentes em aprendizado de máquina têm se mostrado muito promissores quando aplicados em biologia molecular por permitirem uma abordagem menos complexa e com acurácia nos resultados. Destacam-se entre as tecnologias as redes neurais de aprendizado

profundo, em especial, arquiteturas como RNN (do ingles, *Reccurent Neural Network*, CNN (do inglês, *Convolutional Neural Network*) e Autoencoders, que aprendem as características necessárias ao objetivo da previsão, extraindo-as de maneira automática.

O objetivo desta pesquisa é avaliar técnicas de aprendizado profundo aplicadas ao desenho de regiões CDR3 que permitam a criação de anticorpos para um epítopo específico. Mais especificamente, o objetivo é avaliar um modelo baseado em RNN para a predição do CDR3, reduzindo neste caso, a complexidade de uma abordagem tradicional através de estruturas tridimensionais.

Durante a pesquisa estão sendo avaliadas técnicas estado da arte, de aprendizado profundo, na predição de CDRs a partir de epítopos. A construção de anticorpos a partir de epítopos é de grande importância no desenvolvimento de novos tratamentos, principalmente contra o câncer.

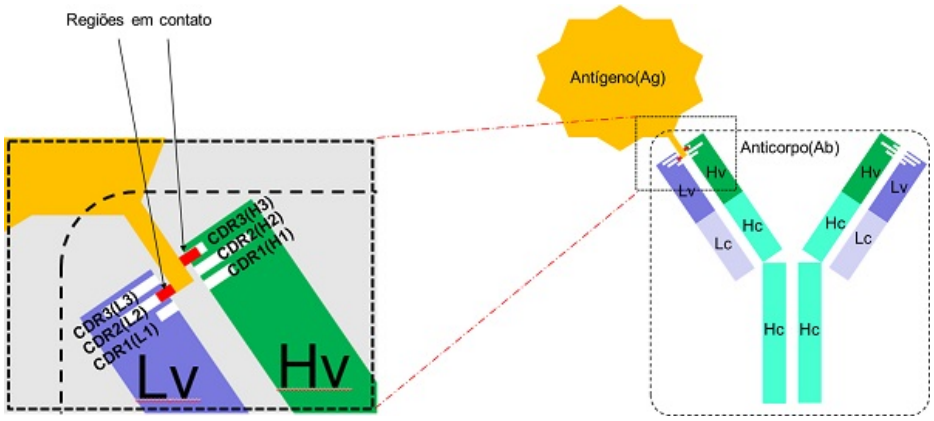


Figura 1. Estrutura de um anticorpo

Oportunamente, as regiões CDR3 propostas pela pesquisa podem ser copiadas em anticorpos e testadas em laboratório (*in vitro*), como ilustrado na figura 2.

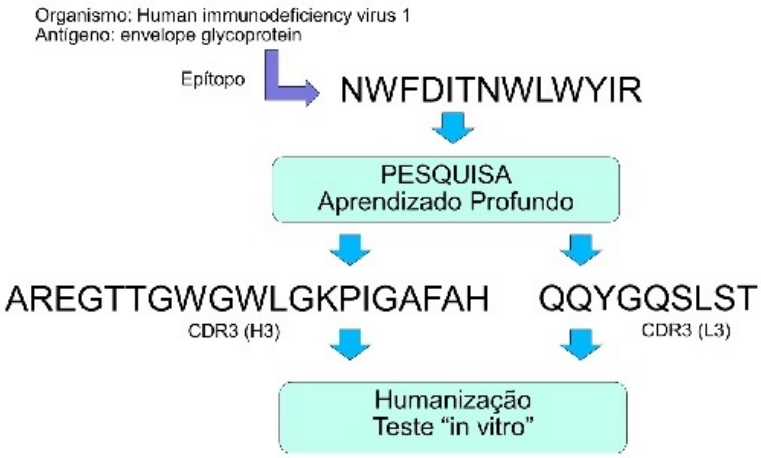


Figura 2. Objetivo da Pesquisa

3. Método

O método a ser utilizado possui as seguintes etapas:

- encontrar e avaliar fontes de dados de qualidade e com tamanho suficiente para o treinamento dos modelos;
- identificar e entender métodos estado-da-arte, disponíveis atualmente;
- definir métricas apropriadas para avaliação da performance e resultado dos modelos;
- implementar e avaliar um modelo para predição de CDR3 a partir de epítopos.

O modelo foi implementado em `Python` usando bibliotecas `tensorflow` e `keras`, em um ambiente `docker`, com três GPUs Nvidia [1].

4. Resultados

A pesquisa está sendo desenvolvida desde o início de 2020, já tendo passado pela avaliação da fonte de dados, definição do modelo de referência e definição de métricas. Atualmente, o modelo implementado passa por um processo contínuo de ajustes e avaliação, onde se busca melhorar a qualidade dos resultados.

4.1. Fonte de Dados

Algumas bases públicas passaram por uma avaliação inicial, sendo que a IEDB [Vita et al. 2019] se mostrou mais adequada para o projeto, pois contém pares epítopo-CDR3 e permite uma seleção flexível. Em uma primeira seleção foram separados 472 tuplas distintas com epítopos contíguos, resultando, em 360 tuplas com epítopos distintos. Para validação cruzada, foram gerados dez conjuntos de dados, cada um segmentado em 80% para treino e 20% para validação.

Um dos principais problemas encontrados é o reduzido tamanho da base de treino, o que pode não ser suficiente para permitir ao modelo aprender as características essenciais necessária para a predição do CDR3.

4.2. Modelo de Referência

O modelo proposto por Luong e colegas [Luong et al. 2015], que obteve resultado muito relevantes na área de tradução, foi escolhido como ponto de partida, por ser um modelo baseado em sequência, o que evita a complexidade e demanda de recursos computacionais requeridos em modelos tridimensionais. Modelo semelhante já havia sido avaliado por Akbar e colegas, porém sem resultados relevantes [Akbar et al. 2019].

O modelo foi adaptado para predição de duas sequências H3 e L3 a partir do epítopo, utilizando um dicionário com vinte duas entradas, correspondentes aos vinte aminoácidos, que tipicamente constituem os anticorpos e dois *tokens* indicadores de início e fim das sequências (`A C D E F G H I K L M N P Q R S T V W Y <start> <end>`).

Desta forma, o modelo proposto deve ser capaz de aprender, a partir das sequências de resíduos, os aspectos relevantes da interação Ab-Ag bem como da estabilidade da estrutura do CDR no anticorpo. Um das dificuldades neste sentido é a existência de várias sequências de CDRs possíveis para um mesmo epítopo, algumas com acoplamento mais forte e outras com acoplamento mais fraco.

[1] Agradecimentos ao Projeto Fapesp 2018/00031–7 pela infraestrutura de processamento

4.3. Métricas

Um das principais diferenças de tradução de textos em linguagem natural (elementos de entrada e saída do modelo de referência usado como base) e sequência de aminoácidos, é a potencial aceitação de sequências com substituições de resíduos, conforme sugerido, por exemplo, pela tabela BLOSUM [Henikoff and Henikoff 1992]. A aceitação de pequenos deslocamentos nas sequências de CDR também é um fator importante a ser considerado.

Desta forma, duas métricas foram criadas, uma baseada na distância de Levenshtein e outra baseada em similaridade. Estas métricas são mais adequadas que *Cross Entropy* para avaliar a similaridade dos resultados obtidos em relação aos resultados esperados, pois podem também avaliar os resultados com relação a deslocamentos e substituições.

4.4. Avaliação do Modelo Implementado

A partir da criação do modelo de referência, iniciou-se um ciclo de testes e aprimoramento buscando sempre melhorar os resultados de predição. Neste primeiro modelo, foi obtido um nível elevado de aprendizado ($loss \sim 0$) após 50 épocas, porém com clara indicação de *overfitting*.

4.4.1. Avaliação das métricas

O primeiro teste com outras métricas do modelo envolveu a avaliação da substituição da função de perda original, no caso, *CategoricalCrossEnropy* por uma função de perda baseada na distância de levenshtein, no caso, $leven\_loss = 1 - leven(pred)/leven(targ)$, sendo $leven(pred)$ a distância de levenshtein entre o valor estimado e o valor esperado e, $leven(targ)$, a distância de levenshtein entre o valor esperado e o valor esperado. Entretanto, utilizando $leven\_loss$, o modelo não foi capaz de aprender, mesmo após 100 épocas de treino.

Como o treino com $leven\_loss$ tornou os testes mais demorados, devido ao cálculo da distância de levenshtein, também foi implementado um modelo de $similar\_loss$ baseado em similaridade. Apesar do treino ter ficado mais rápido com cálculo da similaridade, o modelo também não foi capaz de aprender após 100 épocas. Assim, devido aos resultados negativos com a variação das métricas, retornou-se ao modelo original com $loss = CategoricalCrossEntropy$.

4.4.2. Repetições de epítopos

Desde as avaliações iniciais dos dados de treino, já era de conhecimento a existência de tuplas (epítopo, H3 ,L3) distintas, porém com mesmo epítopo. Suspeitou-se que tais repetições poderiam distorcer os resultado ou dificultar o aprendizado.

Para evitar a repetição de epítopos, foram criados novos conjuntos de treino (iedb2), primeiramente apenas adicionando um resíduo aleatório ao final dos epítopos das tuplas com epítopos repetidos. Os resultados, entretanto, não tiveram diferença significativa em relação a base inicial. Optou-se então, por avaliar uma base sem repetição

de epítopos (iedb3). Neste caso os resultados foram semelhantes ($loss \sim 0$), porém por volta de 70 épocas e com uma pequena piora nos resultados dos testes.

Concluiu-se que, nos diversos conjuntos para validação cruzada, poderiam ocorrer casos onde o mesmo epítopo poderia aparecer no conjunto de treino e no conjunto de validação, o que, de certo modo, tenderia a piorar os resultados, pois mesmo com a memorização do valor de treino de uma tupla para um epítopo x, o valor esperado para o mesmo epítopo x na validação seria diferente. Desta forma, definiu-se que todos os testes futuros deveriam evitar tuplas com epítopos repetidos.

4.4.3. Regularização

Desde os primeiros testes, a indicação de ocorrência de *overfitting* indicava que algum tipo de regularização deveria ser avaliada.

Dentre as diversas opções propostas por Goodfellow [Goodfellow et al. 2016, Cap.5], as seguintes foram avaliadas:

early stopping : Este foi o primeiro a ser avaliado, porém verificou-se que não seria aplicável, pelo menos no estágio atual, visto que em nenhum dos modelos avaliados até o momento produzia resultado com erro de predição minimamente aceitável.

data augmentation : Dois cenários envolvendo *data augmentation* foram avaliados, sendo que em ambos foi aplicado 25% de mutações aleatórias nos epítopos:

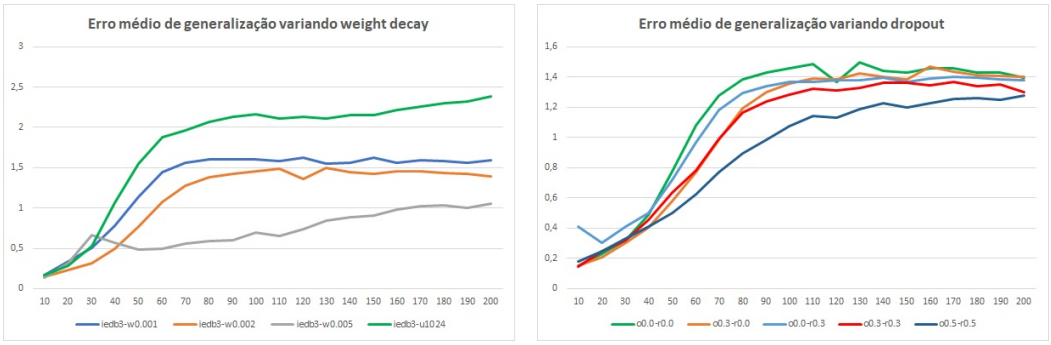
- mutações aplicadas duas vezes nos registros de treino;
- mutações aplicadas cinco vezes nos registros de treino:

Ambos os casos levaram o treino a alcançar um ($loss \sim 0$) em poucas épocas, porém piorando substancialmente o erro de generalização. Decidiu-se então experimentar uma variante de predição, reduzindo a previsão de CDR correspondente ao par (H3,L3) para apenas o elemento H3. Esta alteração permitiu a redução do erro de generalização, porém ainda em um patamar elevado. De qualquer forma, dada esta redução, optou-se por continuar as avaliações apenas com a predição de H3. A limitação dos resultados obtidos com as técnicas de *data augmentation* utilizadas mostra o grande desafio que envolve a geração de novos casos de treino relevantes para a pesquisa, no caso, tuplas (epítopo, H3, L3);

weigth decay : Avaliou-se a técnica *weigth decay* o que permitiu uma pequena melhora na generalização para $weigthdecay = 0.002$, porém, ainda insuficiente;

noise : O modelo de referência utiliza uma camada de rede com função de *embedding*. Com o objetivo de avaliar *noise* como uma técnica de regularização, a camada de *embedding* foi substituída por uma camada com codificação *one–hot*, sobre a qual aplicou-se um ruído aleatório. O modelo resultante foi avaliado para ruídos nos intervalos [-0.2 0.2] e [-0.4 0.4], os quais pioraram o erro de generalização e também para codificação *one–hot* pura, o qual teve desempenho semelhante a camada *embedding*;

dropout : O uso da técnica de *dropout* em RNNs é mais complexa e envolve escolher o uso, combinado ou não, de *dropout* na entradas e nas unidades recorrentes. Em nosso modelo, avaliou-se a técnica *dropout* em ambos os casos para valores 0.0, 0.3 e 0.5. Observou-se uma pequena melhora na generalização, porém insignificante para o desempenho real do modelo.



weight decay *dropout*

Figura 3. Erro médio de generalização

Alguns dos resultados comparativos da aplicação da regularização podem ser vistos na figura 3.

5. Conclusões

Atualmente, o modelo proposto ainda está em processo de aprimoramento e avaliação. Os resultados mostram que vários dos modelos avaliados aprendem bem o conjunto de treino, mas deixam a desejar na generalização, para o conjunto de validação. Os próximos passos incluirão ainda alguns ajustes para avaliação de mecanismos diferentes de atenção, modelo de substituição (BLOSUM) e otimização dos hiper-parâmetros do modelo.

Acreditamos que os resultados ainda devem melhorar nos próximos testes, principalmente durante a fase de otimização dos hiper-parâmetros, entretanto, o reduzido tamanho dos dados de treino e a baixa eficácia do processo de *data augmentation* se mostram, no momento, o principal limitante para a melhoria dos resultados.

Referências

Akbar, R., Jeliazkov, J. R., Robert, P. A., Snapkov, I., Pavlović, M., Slabodkin, A., Weber, C. R., Safonova, Y., Sandve, G. K., and Greiff, V. (2019). A finite vocabulary of antibody-antigen interaction enables predictability of paratope-epitope binding. *bioRxiv*, page 759498.

Doan, T. T., Melvold, R., and Waltenbaugh, C. (2006). *Imunologia médica essencial*. Guanabara Koogan.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Henikoff, S. and Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Vita, R., Mahajan, S., Overton, J. A., Dhanda, S. K., Martini, S., Cantrell, J. R., Wheeler, D. K., Sette, A., and Peters, B. (2019). The immune epitope database (iedb): 2018 update. *Nucleic acids research*, 47(D1):D339–D343.

Design and Implementation of Collective Operations in a Distributed Task-based Runtime

Rodrigo C. Freitas[1], **Hervé Yviquel**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)

r176848@dac.unicamp.br, herve.yviquel@ic.unicamp.br

Abstract. *Many industrial and academic applications require high-performance computing, but since the single-chip performance hit the power wall, multi-core and multi-node arrangements appeared. Nevertheless, writing readable and portable code for those systems is a challenge, and as larger heterogeneous clusters are constructed, lucid and efficient distributed programming models are paramount. This work proposes the implementation of collective operations on the OmpCluster runtime to improve inter-node communication times through calls to collective operations of the underlying communication system (MPI) and to implemented asynchronous versions of those algorithms. An extension to call MPI_Bcast is already successfully implemented.*

Resumo. *Diversas aplicações requerem computação de alta performace, mas o desempenho máximo de uma unidade de processamento é limitado pela power wall, tornando necessário o uso de sistemas multi-núcleo e multi-nó. Ainda assim, a escrita de código paralelo para esses sistemas é um desafio, tornando importante o desenvolvimento de runtimes distribuídos legíveis e eficientes. Este trabalho propõe a implementação de operações coletivas no runtime OmpCluster, para reduzir o tempo de comunicação inter-nó, através de chamadas para as operações coletivas do sitema de comunicação subjacente (MPI) e para variantes assíncronas dos algoritmos coletivos propostos. Uma extensão para chamar MPI_Bcast já foi implementada com sucesso.*

1. Introduction

Scientists and engineers benefit from the growth in computing performance that allows them to run detailed simulations and analyze complex phenomena. But since the single-chip performance hit the power wall, big volumes of computation are tackled by multi-core and multi-node systems (HPC clusters) [Villa et al. 2014, Chatterjee et al. 2013]. However, as programs require more computational power, their inherent complexity grows, and it becomes a challenge for programmers to write efficient parallel code that utilizes heterogeneous clusters equipped with accelerators, leading to less readable and more hardware-depended code [Pancake 1996, Tullsen et al. 1995]. In fact, current programming models and runtimes are not suitable for extreme-scale parallel systems, and none of the recently proposed programming models that aimed to facilitate the construction of HPC programs combined a simple programming model with high-performance inter-node data transfer and accelerator offloading [Costa et al. 2015]. Therefore, it is of

paramount importance that new distributed runtimes are constructed to allow programmers to develop and maintain portable and readable applications that fully utilize modern HPC systems. In this context, the task-based model aims to abstract the architecture details, as the writer describes portions of code that can be executed as soon as their dependencies are met [Agullo et al. 2017]. This abstraction makes it easy to express any degree of parallelism in terms of the nature of the problem, instead of the underlying hardware.

1.1. MPI+X

The Message Passing Interface (MPI) is the predominant programming model used in HPC to perform inter-node communication, allowing the construction of portable and efficient parallel software. MPI is highly standardized, which provides a high degree of portability, allowing the user to compile and execute his code under any implementation. Nonetheless, modern heterogeneous HPC systems are composed of multi-core processors in addition to different accelerators (i.g. GPUs and FPGAs) to achieve larger computing power [Costa et al. 2015], which leads to the combination of MPI with other programming models to fully utilize those resources. Such combination is called MPI+X, where X usually corresponds to OpenMP. [Laguna et al. 2019] found that almost 75% of the MPI programs studied use this hybrid approach, and more than 15% use both OpenMP and CUDA in addition to MPI. This model shows good performance in many scenarios, but combining different models can hide communication information across those layers that could be used for optimization and can potentially lead to unnecessary resource competition [Costa et al. 2015]. On top of that, mixing programming models complicate code readability and makes it harder to maintain, requiring more expertise and endeavor from programmers [Yviquel et al. 2018].

1.2. OmpCluster

OpenMP is an industry-standard shared memory programming model that uses code annotation to describe parallelism, making code implementation considerably easier and more readable, since the directive (*pragmas*) indicate the parts that will run in parallel. OpenMP supports tasks and computing offloading to accelerator devices (i.g. GPUs and FPGAs). OmpCluster is an extension to the OpenMP model, allowing the offloading of tasks to a computer cluster. The example code in Figure 1 shows directives demarcating tasks. If the *target* directive is added, the OmpCluster runtime may allocate the task to a remote node, using MPI as the underlying communication system. The corresponding dependency graph is represented in Figure 2, and it is possible to see that T2 and T3 can run concurrently.

2. Goals

In the task-based model, the runtime must schedule the work efficiently across the available resources. But as the number of nodes in a system increases, the time spent performing inter-node communication becomes more expressive. For that reason, collective operations can be used to improve performance, since collective operations utilize optimized algorithms to perform group data exchange.

The main proposal of this work is to reduce communication time on OmpCluster, a task-based distributed runtime, using explicit written and auto-detected collective operations. The first case is exemplified in Figure 4, in which a list of items is broadcast to

```
#pragma omp parallel
#pragma omp single
{
#pragma omp task depend(out: i)
prepare_data(&data); \\ T1

#pragma omp task depend(in: i, out: j)
a=do_something(data); \\ T2

#pragma omp task depend(in: i, out: k)
b=do_other_thing(data); \\ T3

#pragma omp task depend(in: j,k)
print_results(a,b); \\ T4
}
```

Figure 1. OpenMP code example that generates the tasks shown in Figure 2.

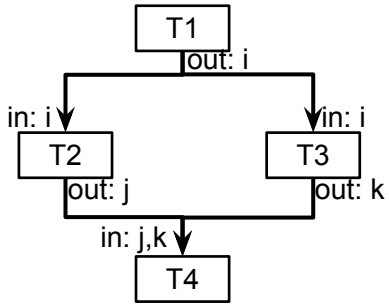


Figure 2. Task graph generated from the code in Figure 1.

every node, as a set of tasks - that can be assigned to any worker - will use that data. The second case works by detecting patterns in pending communication queues and replacing them with the correspondent collective call [Hoefler and Schneider 2012]. Collective operations are interaction patterns that employ efficient algorithms to perform group communication. Each kind of collective algorithm propagates the data in a systematized way over the network to optimize communication time. Besides being topology-aware, we also intend to implement and test GPU-aware collective algorithms, to efficiently transfer memory that is allocated on accelerator devices, since second-level offloading will be added to OmpCluster in the future. Figure 3 shows some commonly implemented broadcast algorithms, with different approaches to propagate the data across the nodes. The optimal choice varies according to the network characteristics, the message size, and the number of nodes. For instance, case *iii* relies on low package loss on the network to be efficient.

3. Methodology

As Section 2 states, collective operations are imperative for communication time performance. As a matter of fact, collective operations are more common than peer-to-peer data exchange on high-performance MPI applications [Laguna et al. 2019]. But some MPI operations may not fit properly on the task-based model, as they require synchronization. Fortunately, [Denis et al. 2020] showed that it is possible to construct efficient collective algorithms that do not require synchronization. Therefore, we will propose and benchmark asynchronous versions of other collective operations on the OmpCluster runtime.

This asynchronous approach proposed by [Denis et al. 2020], called Dynamic Broadcast, works using self-contained messages that perform a binomial tree broadcast, instead of asking every process to call MPI_Bcast. The root process simply sends the data to a list of processes, considering a communication topology. The receiving processes do not know in advance that they will be participating in a collective operation, but as they receive the message payload, they also receive meta-information about their role in the collective algorithm. This means that as soon a process receives a broadcast

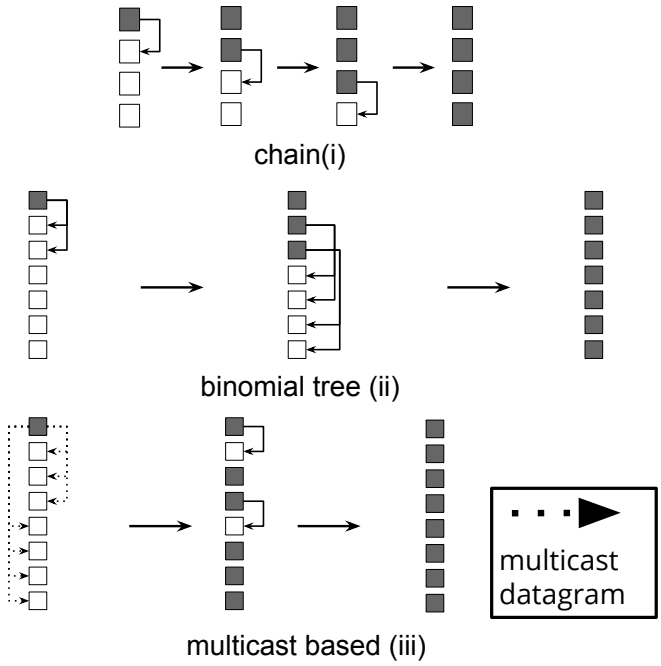


Figure 3. Examples of broadcast implementations. (i) and (ii) may be preferred depending on the number of nodes, the message size, and the network latency and bandwidth. Note that (iii) relies on the network characteristics to be efficient, since its performance is highly tied to the reliability with which a multicast datagram is received.

message, it also receives the list of nodes that they will need to forward the data to. For comparison, the drawbacks of synchronously using a traditional call to MPI_Bcast can be visualized in Figure 5. As the figure shows, the root process must first send a message to every participant, which is a huge overhead, considering that this communication is itself another broadcast. It is important to note that this asynchronous variation can also be used on other collectives algorithms besides broadcast, employing the same principle: transmit the communication topology with the payload, allowing the participants to asynchronously perform their role when they process the message.

4. Conclusions

We expect that the usage of collective operations can improve the performance of distributed programs that use the OmpCluster runtime, particularly when the computation is performed on a large number of nodes.

As Section 2 states, this work goal is to implement collective operations and measure their performance impact. Benchmarks like Task bench [Slaughter et al. 2019], constructed to evaluate task-based runtimes performance, will be used to measure the improvements. The mechanism to call MPI_Bcast, as shown in Figure 4, is already implemented. Succeeding, dynamic broadcasts will be added to the runtime. Other collective algorithms, as reduce, gather and allgather operations are the natural following steps to be implemented.

```
#pragma omp target data map(to : a, b, c, d)
{
 #pragma omp target nowait
 operate_on_data_1(a,b,c,d);

 #pragma omp target nowait
 operate_on_data_2(a,b,c,d);

 #pragma omp target nowait
 operate_on_data_3(a,b,c,d);
}
```

Figure 4. OmpCluster code example that translates to a call to MPI_Bcast, since the mapped data may be used by any node that is assigned any task inside the region.

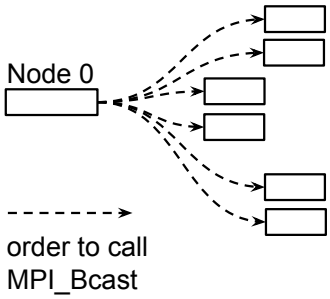


Figure 5. Synchronized call to MPI_Bcast on a task-based runtime. MPI requires that every process calls the collective operation.

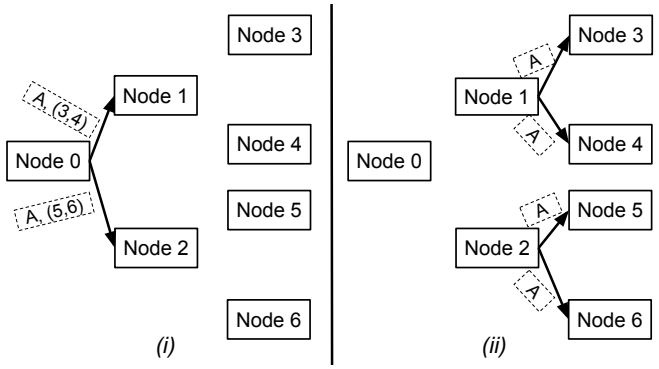


Figure 6. Example of a Dynamic Broadcast: in (i), the root process (0) sends the data (A) to nodes 2 and 3, as it would in a binomial tree algorithm. With the payload, it also sends to each node how they should forward the message. In (ii), node 2 and 3 asynchronously conclude the broadcast.

References

Agullo, E., Aumage, O., Faverge, M., Furmento, N., Pruvost, F., Sergent, M., and Thibault, S. P. (2017). Achieving High Performance on Supercomputers with a Sequential Task-based Programming Model. *IEEE Transactions on Parallel and Distributed Systems*, 9219(c):1–14.

Chatterjee, S., Taşirlar, S., Budimlić, Z., Cavé, V., Chabbi, M., Grossman, M., Sarkar, V., and Yan, Y. (2013). Integrating asynchronous task parallelism with MPI. *Proceedings - IEEE 27th International Parallel and Distributed Processing Symposium, IPDPS 2013*, pages 712–725.

Costa, G. D., Fahringer, T., Rico-Gallego, J. A., Grasso, I., Hristov, A., Karatza, H. D., Lastovetsky, A., Marozzo, F., Petcu, D., Stavrinides, G. L., Talia, D., Trunfio, P., and Astsatryan, H. (2015). Exascale machines require new programming paradigms and runtimes. *Supercomputing Frontiers and Innovations*, 2(2):6–27.

Denis, A., Jeannot, E., Swartvagher, P., Thibault, S., Denis, A., Jeannot, E., Swartvagher, P., Thibault, S., Dynamic, U., Denis, A., Jeannot, E., and Swartvagher, P. (2020). Using Dynamic Broadcasts to improve Task-Based Runtime Performances.

Hoefler, T. and Schneider, T. (2012). Runtime detection and optimization of collective communication patterns. *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, pages 263–272.

Laguna, I., Marshall, R., Mohror, K., Ruefenacht, M., Skjellum, A., and Sultana, N. (2019). A large-scale study of MPI usage in open-source HPC applications. *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*.

Pancake, C. M. (1996). What computational scientists and engineers should know about parallelism and performance. *Computer Applications in Engineering Education*, 4(2):145–160.

Slaughter, E., Wu, W., Fu, Y., Brandenburg, L., Garcia, N., Kautz, W., Marx, E., Morris, K. S., Lee, W., Cao, Q., Bosilca, G., Mirchandaney, S., Treichler, S., McCormick, P., and Aiken, A. (2019). Task Bench: A Parameterized Benchmark for Evaluating Parallel Runtime Performance.

Tullsen, D. M., Eggers, S. J., and Levy, H. M. (1995). Simultaneous multithreading. *ACM SIGARCH Computer Architecture News*, 23(2):392–403.

Villa, O., Johnson, D. R., O'Connor, M., Bolotin, E., Nellans, D., Luitjens, J., Sakharnykh, N., Wang, P., Micikevicius, P., Scudiero, A., Keckler, S. W., and Dally, W. J. (2014). Scaling the Power Wall: A Path to Exascale. *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*, 2015-Janua(January):830–841.

Yviquel, H., Cruz, L., and Araujo, G. (2018). Cluster programming using the OpenMP accelerator model. *ACM Transactions on Architecture and Code Optimization*, 15(3).

Monitoramento automatizado da evolução da coesão em Arquiteturas de Microsserviços

Mateus G. Moreira[1], **Breno B. N. de França**[1]

[1]Instituto de Computação – Universidade de Campinas (UNICAMP)
Campinas – SP – Brazil

mateusgabimoreira@gmail.com, breno@ic.unicamp.br

Abstract. *Microservice Architecture (MSA) has become a new architecture style, but there are some unsolved problems related to maintainability. One of the software design properties influencing maintainability is cohesion. Cohesion is defined as the degree to which the elements of a module belong together. Most of the cohesion metrics found in the literature can be used in MSA with limitations, but existing studies provide no evidence on the validity of these metrics. Software architects need to be proactive and identify potential issues in a microservice architecture before they are too complex to be solved. We are proposing a method for tracking the evolution of cohesion in microservices using cohesion metrics. We collected a set of cohesion metrics from literature, and we are developing a tool to assess these cohesion metrics at delivery-time.*

Palavras-chave: Microsserviços; Evolução de Software; Métricas de Software

1. Introdução

Um dos objetivos de uma Arquitetura em Microsserviços (MSA) é a redução da complexidade da aplicação por meio da decomposição das funcionalidades em pequenas unidades independentes [Wolff 2016, Tyszberowicz et al. 2018]. Cada unidade é chamada de microsserviço e é caracterizada por ser uma aplicação independente, por ter interfaces de acesso leve (comumente HTTP, gRPC, ou serviços de mensageria como Kafka e RabbitMQ) e possuir código-fonte, banco de dados e ciclo de vida igualmente independente [Dragoni et al. 2017, Lewis and Fowler 2014, Wolff 2016]. Existem varios problemas ainda não solucionados encontrados literatura sobre microsserviços [Wolff 2016, Dragoni et al. 2017, Bogner et al. 2017], um deles é a manutenibilidade.

Manutenibilidade é um atributo de qualidade definida pela ISO/IEC SQuaRE como o grau de eficácia e eficiência com que um produto ou sistema pode ser modificado por seus mantenedores [ISO/IEC 2011]. Para seu monitoramento e avaliação, utiliza-se comumente suítes de métricas, nas quais métricas de coesão normalmente são parte dessas suítes. Arquitetos de software precisam ser pró-ativos e identificar possíveis problemas em uma arquitetura de microsserviços antes que esses problemas sejam complexos o suficiente para inviabilizar a correção. A manutenibilidade de um software é influenciada por atributos de software como acoplamento e coesão [Card et al. 1986, Briand et al. 2000, Dagpinar and Jahnke 2003]. Stevens at al. [Stevens et al. 1974] definiram coesão como o grau em que os elementos de um módulo pertencem a si.

A proposta deste trabalho é um método que auxilia arquitetos a encontrar possíveis problemas arquiteturais utilizando a coleta de métricas de coesão de forma sistemática a cada entrega de um microsserviço.

Este trabalho esta dividido nas seguintes seções. A Seção 2 apresenta os trabalhos relacionados para a execução deste trabalho. A Seção 3 apresenta a proposta de solução. A Seção 4 apresenta os resultados obtidos até o momento. A Seção 5 discute sobre a continuidade e expectativa da continuidade deste trabalho.

2. Trabalhos Relacionados

Conduzimos uma revisão de literatura com objetivo de identficar trabalhos relacionados. Encontramos métricas para avaliar sistemas orientados a objetos [Chidamber and Kemerer 1992, Bieman and Kang 1995, Briand et al. 1998] e algumas propostas para arquiteturas orientadas a serviços [Perepletchikov et al. 2007, Bogner et al. 2017, Athanasopoulos et al. 2015]. Contudo, não encontramos métricas de coesão exclusivamente desenvolvidas para Arquiteturas de Microsserviços.

No trabalho de [Perepletchikov et al. 2007] foram introduzidas métricas de coesão a fim de se avaliar a manutenibilidade de em Arquiteturas Orientadas a Serviços. Ele definiu as seguintes métricas: i) *Service Interface Data Cohesion* (SIDC) calcula a coesão da interface de um serviço dado os tipos dos parâmetros de entrada. Um serviço é coeso quando as operações providas compartilham os mesmos tipos de parâmetros de entrada. ii) *Strict Service Implementation Cohesion* (SSIC) calcula a coesão das operações providas de um serviço por meio de sua implementação. Um serviço é coeso quando todas as operações providas são implementadas pelos mesmos elementos (objetos, classes, *scripts*, interfaces, etc). iii) *Loose Service Implementation Cohesion* (LSIC) é similar a SSIC, contudo as dependências indiretas dos elementos são consideradas durante a avaliação da métrica. iv) *Service Interface Usage Cohesion* (SIUC) calcula a coesão de um serviço dado o comportamento de seus clientes. Um serviço é coeso quando todas as operações de um serviço são invocadas por todos os clientes. v) *Service Sequential Usage Cohesion* (SSUC) calcula a coesão de um serviço dado a cadeia de invocações das operações. Um serviço é coeso se a resposta de uma operação é a entrada da próxima operação. vi) *Total Interface Cohesion of a Service* (TICS) calcula a coesão de um serviço dado os valores obtidos pelas outras métricas. TICS é a média dos valores de todas as métricas anteriores.

[Athanasopoulos et al. 2015] decompõe interfaces de serviços utilizando métricas de coesão. Para isso, [Athanasopoulos et al. 2015] propõe algumas métricas. i) *Message Level Cohesion* (MLC) que avalia a similaridade entre as mensagens de duas operações. Duas operações são similares quando as mensagens de entrada e saída são similares. ii) *Conversation Level Cohesion* (CLC) avalia a similaridade entre operações. Duas operações são similares se a saída de uma operação é similar a entrada de outra operação. iii) *Lack of Message-Level Cohesion* (LoCMes) calcula a coesão da interface de um serviço dado os tipos dos parâmetros de entrada e saída. Um serviço é coeso quando as operações providas compartilham os mesmos tipos de parâmetros de entrada e saída.

[Bogner et al. 2020] propôs uma ferramenta chamada RESTful API Metric Analyzer (RAMA CLI) [Bogner et al. 2020] que é uma ferramenta em linha de comando que pode avaliar a manuteniblidade de APIs RESTful por meio de métricas arquiteturais. Essas métricas são calculadas com base na documentação das interfaces dos serviços pro-

vidos. Ele recebe como entrada um arquivo de especificação e retorna um relatório com métricas de interface de serviço.

O trabalho de [Bogner et al. 2020] foca na avaliação de métricas de complexidade, coesão e tamanho por meio da análise apenas de suas interfaces, emquanto a nossa proposta avalia a coesão de um microsserviço utilizando métricas que consideram aspectos das interfaces e implementação dos serviços. Ademais, [Bogner et al. 2020] avalia apenas serviços RESTful e nós avaliamos interfaces gRPC. Pretendemos rapidamente oferecer suporte a RESTful, HTTP, e serviços de *stream* de dados e mensageria.

3. Solução Proposta

A proposta deste trabalho é um método que auxilia arquitetos a encontrar possíveis problemas arquiteturais utilizando a coleta de métricas de coesão de forma sistemática a cada entrega de um microsserviço. Para isso nos implementamos uma ferramenta que coleta as métricas de coesão de forma automatizada.

O nosso trabalho se restringe a análise estática de código porque estamos interessados na estrutura do código-fonte dos microsserviços. Isto posto, não utilizamos as métricas que avaliam trocas de mensagens entre microsserviços e o uso de operações por clientes. Tais métricas necessitam de análise dinâmica de todo o sistema. As métricas que avaliam a coesão das classes também não foram utilizadas uma vez que microsserviços podem ou não ser desenvolvidos utilizando o paradigma orientado à objetos.

A nossa ferramenta chamada *Aegeus* tem como objetivo calcular as métricas de coesão de um microsserviço a cada entrega. *Aegeus* possui seu código aberto e disponível gratuitamente para download e extensão no Github[1]. A *Aegeus* possui três módulos no estilo *Pipes and Filters*: *Readers*, *Metric Calculator* e *Exporters*. Os *Readers* lê o arquivo de entrada, o interpreta e envia para o *Metric Calculator*. *Metric Calculator* é responsável por calcular as métricas e enviar o resultado para os *Exporters* que apresenta os resultados ao usuário da ferramenta. Essa arquitetura permite que a nossa ferramenta seja adicionada no ciclo de CI/CD, facilitando a automação da avaliação da qualidade interna dos microsserviços.

4. Resultados Parciais

Selecionamos diferentes projetos de código-aberto disponíveis no GitHub que utilizam a arquitetura de microsserviços a fim de avaliar o uso do método proposto. Atualmente, a ferramenta suporta aplicações que utilizam a linguagem de programação Java e interfaces gRPC.

A plataforma de código-aberto *SiteWhere*[2] permite gerenciamento de dispositivos, áreas de atuação e agendamento de tarefas no domínio de Internet das Coisas (IoT) . Entre os microsserviços existentes em sua arquitetura, selecionamos o microsserviço chamado *device management* devido a sua importância no domínio de Internet das Coisas e por ter um longo histórico de alterações em seu código-fonte durante as entregas. Esse microsserviço possui um total de 17 entregas, entre candidatas e estáveis.

[1]Disponível em: https://github.com/MateusGabi/Aegeus
[2]Disponível em: https://sitewhere.io

A ferramenta foi executada para cada umas das 17 entregas a fim de se observar uma variação na coesão, seja decaimento ou aumento, desse microsserviço. Atualmente apenas as métricas SIDC e SSIC estão implementadas. Pretendemos implementar e avaliar as métricas LoCMes e TICS posteriormente. A Figura 1 mostra a evolução da métrica SSIC. A métrica SSIC analisa a implementação das operações, no qual um serviço é altamente coeso quando todas as operações providas são implementadas pelos mesmos elementos. No caso do microsserviço *device management*, os elementos são classes e interfaces. Ao verificar a evolução da implementação desse microsserviço, podemos notar o aumento no número de operações, Figura 2, e também a falta de compartilhamento das classes, o que resulta em um baixo grau de coesão. Ao comparar os valores obtidos da execução das métricas, encontramos um decaimento de 45,75% na coesão do microsserviço *device management* ao comparar a primeira com a última entrega. Por outro lado, há a adição de 42,62% novas operações, Figura 2, em comparação da primeira com a última entrega.

A métrica SIDC também foi calculada para todas as entregas e resultaram no valor zero, o que mostra falta de coesão nas API desse serviço. Quando analisamos a arquitetura dessa aplicação, percebemos que para cada uma das operações providas pelos serviços, diferentes abstrações são criadas para os parâmetro de entrada e para a resposta da operação. Esse comportamento não é a ideal no contexto de coesão de software, pois são criadas diferentes abstrações para representar as mesmas entidades.

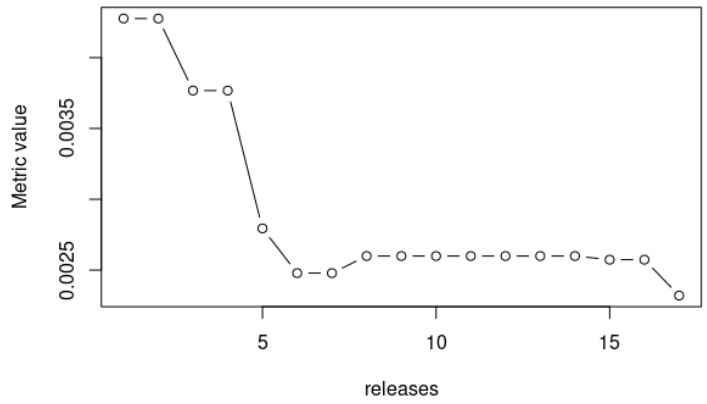


Figura 1. Evolução da métrica SSIC

5. Considerações Finais

Os resultados são iniciais contudo podemos visualizar a evolução da coesão em arquiteturas de microsserviços, ainda que sob a perspectiva de apenas duas métricas. Um dos objetivos específicos trata de alertar possíveis tendências de decaimento na coesão a fim de antecipar os problemas e corrigí-los antes de se tornar um problema mais grave.

Pretendemos oferecer suporte a outros protocolos de comunicação como HTTP, Kafka e RabbitMQ bem como outras linguagens que são comumente utilizadas em arquiteturas de microsserviços, tais NodeJS, Python, TypeScript e outras. O suporte a diversas protocolos de comunicação e linguagens nos permite verificar a viabilidade da utilização das métricas e da nossa ferramenta em diversos domínios e *stacks*.

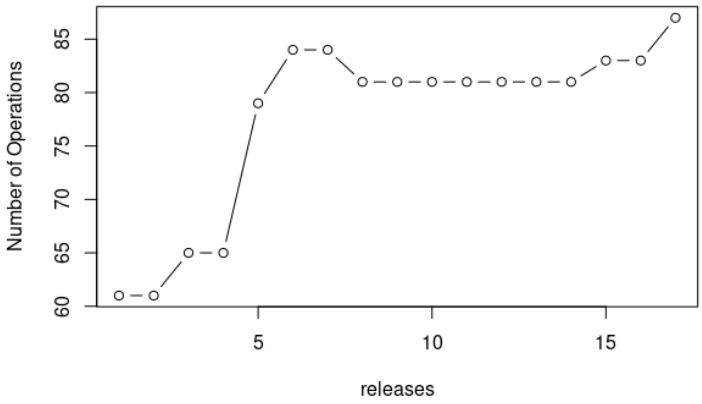


Figura 2. Evolução da quantidade de operações

Acknowledgement

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 88887.388238/2019-00.

Referências

Athanasopoulos, D., Zarras, A. V., Miskos, G., Issarny, V., and Vassiliadis, P. (2015). Cohesion-Driven Decomposition of Service Interfaces without Access to Source Code. *IEEE Transactions on Services Computing*, 8(4):550–5532.

Bieman, J. M. and Kang, B.-K. (1995). Cohesion and reuse in an object-oriented system. *SIGSOFT Softw. Eng. Notes*, 20(SI):259–262.

Bogner, J., Wagner, S., and Zimmermann, A. (2017). Automatically measuring the maintainability of service- and microservice-based systems - a literature review. In *ACM International Conference Proceeding Series*, volume Part F1319, pages 107–115.

Bogner, J., Wagner, S., and Zimmermann, A. (2020). Collecting Service-Based Maintainability Metrics from RESTful API Descriptions: Static Analysis and Threshold Derivation. pages 1–14.

Briand, L. C., Daly, J. W., Urgen, J., and Ust, W. (1998). A Unified Framework for Cohesion Measurement in Object-Oriented Systems. 117:65–117.

Briand, L. C., Wüst, J., Daly, J. W., and Porter, D. V. (2000). Exploring the relationships between design measures and software quality in object-oriented systems. *Journal of Systems and Software*, 51(3):245 – 273.

Card, D. N., Church, V. E., and Agresti, W. W. (1986). An empirical study of software design practices. *IEEE Transactions on Software Engineering*, SE-12(2):264–271.

Chidamber, S. and Kemerer, C. (1992). *A METRICS SUITE FOR OBJECT ORIENTED DESIGN*.

Dagpinar, M. and Jahnke, J. H. (2003). Predicting maintainability with object-oriented metrics -an empirical comparison. In *10th Working Conference on Reverse Engineering, 2003. WCRE 2003. Proceedings.*, pages 155–164.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L. (2017). *Microservices: Yesterday, Today, and Tomorrow*, pages 195–216. Springer International Publishing, Cham.

ISO/IEC (2011). *SO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. ISO/IEC.

Lewis, J. and Fowler, M. (2014). Microservices.

Perepletchikov, M., Ryan, C., and Frampton, K. (2007). Cohesion metrics for predicting maintainability of service-oriented software. *Proceedings - International Conference on Quality Software*, (Qsic):328–335.

Stevens, W. P., Myers, G. J., and Constantine, L. L. (1974). Structured design. *IBM Syst. J.*, 13(2):115–139.

Tyszberowicz, S., Heinrich, R., Liu, B., and Liu, Z. (2018). Identifying Microservices Using Functional Decomposition. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10998 LNCS:50–65.

Wolff, E. (2016). *Microservices: flexible software architecture*. Addison-Wesley Professional.

Serviço de Sugestão de Rotas para Carrinheiros na Coleta Seletiva de Materiais Recicláveis

Maria Vitória R. Oliveira[1], **Islene C. Garcia**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (Unicamp)
Caixa Postal 6176 – 13083-970 – Campinas – SP – Brasil

m262884@dac.unicamp.br, islene@ic.unicamp.br

Abstract. *"Carrinheiros"are collectors of recyclable materials that use vehicles of human propulsion in the selective collection, following routes with specific stops. The problem is that route can be very tiring for waste pickers according to the increase in vehicle weight and the slope of the roads. Therefore, this work proposes a route suggestion service to minimize the physical effort of the pickers on the collection route. The characteristics of the scenario addressed in this proposal are the distance between collect points, the geographical position of the depot, the inclination of the roads, and the use of vehicles with human propulsion. Besides, this work should consider the variation in vehicle weight along the route. According to the availability of materials in a specific coverage area of the waste picker, stopping points should be defined, following a data model based on the scheme of the waste advertising platforms, such as Cataki and Destino Sustentável. The validation of the proposal must be performed through computer simulations, using Simulation of Urban MObility (SUMO) and geographic data from Open Street Map.*

Resumo. *Os carrinheiros são catadores de materiais recicláveis que utilizam veículos de propulsão humana na coleta seletiva, seguindo roteiros com pontos de parada específicos. O problema é que o roteiro pode se tornar extremamente cansativo para os catadores, de acordo com o aumento do peso do veículo e da inclinação das vias. Portanto, este trabalho propõe um serviço de sugestão de rotas para minimizar o esforço físico dos catadores na rota coleta. As características do cenário consideradas nesta proposta são distância entre os pontos de coleta, a posição geográfica do depósito, a inclinação das vias e a utilização de veículos de propulsão humana. Além disso, este trabalho deve considerar a variação do peso do veículo ao longo do percurso. De acordo com a disponibilidade de materiais na área de cobertura específica do catador, os pontos de parada devem ser definidos, seguindo um modelo de dados embasado no esquema das plataformas de anúncios de resíduos, como Cataki e Destino Sustentável. A validação da proposta deve ser realizada por meio de simulações computacionais, utilizando o Simulation of Urban MObility e dados geográficos do Open Street Map.*

1. Introdução

A gestão de Resíduos Sólidos Urbanos (RSU) no Brasil foi regulamentada pela Política Nacional de Resíduos Sólidos (PNRS), em 2010, a qual estabeleceu a responsabilidade

compartilhada da destinação adequada de RSU entre cooperativas, poder público e empresas privadas [CEMPRE 2012]. As cooperativas são entidades sem fins lucrativos, formadas por catadores [SEBRAE 2017]. Esses profissionais realizam a coleta seletiva, definida pela PNRS como a segregação prévia dos materiais conforme a constituição ou composição [BRASIL 2010]. De acordo com o censo do Instituto Brasileiro de Geografia e Estatistica (IBGE) (2010), existem 387.910 catadores de materiais recicláveis no Brasil [Silva et al. 2013]. Porém, o Movimento Nacional dos Catadores de Materiais Recicláveis estima que são 800 mil coletores [MNCR 2019].

O recolhimento de materiais recicláveis em ambiente urbano envolve catadores que trabalham individualmente ou coletivamente, nas cooperativas e associações de reciclagem. Tanto os coletores autônomos, como os vinculados a organizações, em sua maioria, utilizam veículos de tração humana na coleta dos resíduos [Castilhos Junior et al. 2013]. Nesse sentido, os profissionais que utilizam esse tipo de veículo são chamados de *carrinheiros*. Dependendo do peso do carrinho e da inclinação das vias ao longo do roteiro, o percurso pode ser demorado e cansativo, visto que os carrinheiros precisam realizar grande esforço físico durante a coleta dos materiais.

Os trabalhos [Tavares et al. 2009] [Fujdiak et al. 2016] [Ahmad et al. 2020] e [Benjamin and Beasley 2010] sugerem soluções de geração de rotas otimizadas para a coleta seletiva, com o objetivo de diminuir a distância e o tempo de percurso, assim como minimizar os custos financeiros nesse tipo de roteamento. No entanto, as soluções propostas nesses trabalhos não consideram a utilização de veículos de propulsão humana com peso variável ao longo do percurso.

Este trabalho propõe um serviço de sugestão de rotas para a coleta seletiva, com o objetivo de minimizar o esforço físico exigido dos carrinheiros para realizar o trajeto de coleta. Para tanto, deve-se identificar os melhores caminhos, considerando as características do cenário, como distância, inclinação das vias e a utilização de veículos de propulsão humana com peso variável. Além disso, os pontos de parada devem ser definidos a partir da disponibilidade de resíduos recicláveis, de acordo com dados vinculados a plataformas de anúncios de materiais. Sendo assim, a relevância social desta pesquisa está em contribuir para a melhora da qualidade de vida de carrinheiros. A relevância científica encontra-se em atender veículos de propulsão humana com peso variável ao longo do roteiro, que é uma abordagem original.

Para os catadores que não têm acesso à Internet durante a coleta, deve-se possibilitar o *download* da rota antes de iniciar o percurso. Além disso, propõe-se disponibilizar a rota de coleta por e-mail, permitindo que os catadores possam imprimi-la, no caso de não possuírem *smartphones* e houver impressora acessível no galpão de triagem, por exemplo. No entanto, se o carrinheiro possuir *smartphone* e acesso à Internet móvel, propõe-se a utilização da solução dinâmica, que permite *feedbacks* sobre o peso do veículo ao longo da coleta e possibilita o re-roteamento.

Um dos diferenciais desta proposta está na utilização de dados de sistemas de anúncios de materiais recicláveis para identificar os pontos de coleta. Sendo assim, a quantidade, peso, tipo e localização dos materiais recicláveis disponíveis para recolhimento podem ser verificados a partir de sistemas de anúncios como o Cataki [CATAKI 2020] e o Destino Sustentável [Destino Sustentável 2020]. Estas plata-

formas permitem que os usuários descrevam as características dos materiais que querem descartar. A partir dos dados sobre os materiais extraídos de sistemas abertos, é possível calcular previamente a carga dos resíduos que serão coletados, bem como os pontos de parada dos catadores.

Para definir as rotas dos carrinheiros, deve-se considerar que o cenário é um grafo, onde os pontos de coleta são os clientes/vértices. Além disso, a inclinação das vias, a distância e o peso do veículo influenciam nos pesos das arestas. A validação da proposta deve ser realizada por meio de simulações computacionais utilizando o SUMO, um simulador *opensource* que permite criar cenários, analisá-los, gerar padrões de mobilidade e simular estratégias de gerenciamento de tráfego. Além disso, os dados geográficos dos cenários serão extraídos do *software* livre *Open Street Map*. Dentre os desafios da pesquisa, é possível destacar a criação do cenário considerando informações reais ou relativas de inclinação das vias e a geração de soluções de forma computacionalmente eficiente.

A proposta está organizada da seguinte maneira: A Sessão 2 apresenta a Justificativa da proposta. A Sessão 3 demonstra os Trabalhos Relacionados. A Sessão 4 revela a Metodologia a ser empregada no trabalho. Por fim, a Sessão 5 apresenta a Conclusão.

2. Justificativa

Os pontos de coleta representam locais onde os geradores de materiais recicláveis entregam os resíduos para o coletor. Se a coleta for de porta em porta, estes pontos são as residências das pessoas. Existem também os pontos comunitários, que são *containers*, nos quais grupos de pessoas depositam os resíduos. Para que o recolhimento dos materiais efetivamente aconteça, o catador precisa saber a localização do ponto de coleta, assim como os dias e horários disponíveis para fazer o recolhimento.

No processo de gerenciamento de resíduos sólidos, a coleta primária corresponde à fase de recolhimento dos materiais de porta em porta ou nos pontos comunitários. Nesse sentido, existem três metodologias principais: a coleta direta, a de dois estágios e a de três estágios. Na coleta direta, utiliza-se um único veículo, desde o recolhimento dos materiais durante a coleta primária, até a disposição final. Na coleta de dois estágios, utiliza-se inicialmente um veículo motorizado de pequeno porte, que transporta os materiais até um local de depósito. A partir daí, um veículo de grande porte recolhe os resíduos deixados por diversos pequenos veículos, e transporta até o ponto de disposição final.

Na metodologia de três estágios, a coleta primária é realizada por catadores que utilizam veículos de propulsão humana. Depois, os materiais coletados por cada carrinheiro são levados até o depósito utilizando caminhões de pequeno porte. Por fim, um veículo de grande porte transporta os resíduos do depósito até o ponto de disposição final. Em regiões onde não se pode trafegar veículos de grande porte, como ruas estreias, bairros com casas muito antigas e áreas caracterizadas pelo congestionamento de veículos, a metodologia de três estágios é o único método possível de realizar a coleta.

No Brasil, muitas cooperativas de reciclagem utilizam a coleta de três estágios com o objetivo de minimizar custos financeiros. No entanto, os catadores que utilizam veículos de tração humana durante a coleta primária precisam realizar jornadas de trabalho cansativas, que exigem grande esforço físico. Sendo assim, cerca de 51% dos coletores entrevistados em [Castilhos Junior et al. 2013] classificaram o peso do veículo

como um dos problemas desse tipo de coleta. Consequentemente, a inclinação das vias também influencia no trabalho realizado. Nesse contexto, o roteamento eficiente considerando a variação da carga do veículo, a distância total do roteiro e a inclinação das vias pode melhorar a qualidade de vida dos carrinheiros durante a coleta primária.

3. Trabalhos relacionados

Os trabalhos relacionados referem-se ao roteamento de veículos para a coleta de resíduos sólidos recicláveis. Selecionou-se quatro artigos, publicados de 2009 a 2020.

A pesquisa de Tavares et al [Tavares et al. 2009] propõe um modelo 3D baseado no Sistema de Informação Geográfica (SIG) ArcGIS para gerar rotas de coleta seletiva otimizadas, de modo que o percurso dos veículos tenha o mínimo consumo de combustível possível. Para tanto, considera-se a inclinação das vias e o peso do veículo na modelagem das rotas. Dessa forma, a pesquisa mostra que a rota mais curta muitas vezes não é a que gera menor custo.

O trabalho de Fujdiak et al [Fujdiak et al. 2016] aborda o roteamento de veículos de coleta no contexto das Cidades Inteligentes e *Internet of Things* (IoT), considerando um cenário onde as lixeiras possuem sensores que indicam a porcentagem de resíduo depositado. Nesse sentido, este trabalho tem como objetivo identificar as rotas de coleta mais eficientes para coletar os resíduos das lixeiras que já atingiram a capacidade máxima. Para tanto, o custo das rotas é calculado a partir da distância, densidade do tráfego e qualidade da via. Nesse sentido, desenvolveu-se um Algoritmos Genéticos para gerar as rotas e os resultados evidenciaram que, em média, é possível reduzir os custos da coleta com o caminhão em 15%.

O trabalho de Ahmad et al [Ahmad et al. 2020] propõe um sistema ideal de recomendação de rotas para veículos de coleta seletiva. O objetivo desta pesquisa é minimizar a distância das rotas, o consumo de combustível do caminhão coletor e maximizar a quantidade de materiais recolhidos. Para tanto, emprega-se o algoritmo BAT, fundamentado no comportamento de caça de morcegos, Algoritmos Genéticos (AG) e *Particle Swarm Optimization* (PSO), baseado com um enxame de partículas que se movem para encontrar um máximo global. Nesse sentido, realizou-se o processamento de dados reais de descarte de materiais da cidade de Jeju, na Coreia do Sul, para prever o perfil da região.

A pesquisa realizada por Benjamim e Beasley [Benjamin and Beasley 2010] tem como objetivo gerar rotas de coleta de resíduos para veículos considerando janelas de tempo, período de descanso dos condutores e múltiplas instalações de disposição de resíduos. Nesse sentido, os veículos partem do depósito, realizam a coleta em pontos específicos, seguem até a instalação de disposição para esvaziar a carga e, por fim, retornam ao depósito. As metaheurísticas utilizadas para solucionar o problema foram Busca Tabu (BT), *Variable Neighbourhood Search* (VNS) e um algoritmo combinando as duas, chamado VNTS. De acordo com os experimentos, o VNS apresentou os melhores resultados em comparação com a BT, VNTS e algoritmos da literatura.

A Tabela 1 apresenta a comparação dos trabalhos relacionados ao roteamento para a coleta seletiva, considerando as características tipo de veículo utilizado e parâmetros como capacidade de armazenamento do veículo, distância das rotas, custos financeiros e inclinação das vias. Além disso, acrescentou-se a coluna Veículo, que representa o tipo de

meio de transporte abordado nos trabalhos. Novamente, as características utilizadas como métricas de avaliação ou dados de entrada dos algoritmos nos trabalhos foram marcadas na tabela com o símbolo (✓). A sigla NE representa Não Especificado.

Tabela 1. Comparação entre os trabalhos relacionados

Referência	Veículo	Capacidade	Distância	Custos	Inclinação
[Tavares et al. 2009]	Caminhão		✓	✓	✓
[Benjamin and Beasley 2010]	NE	✓	✓		
[Fujdiak et al. 2016]	Caminhão		✓	✓	
[Ahmad et al. 2020]	Caminhão	✓	✓	✓	

4. Metodologia

Para o desenvolvimento da pesquisa, dividiu-se o trabalho nas seguintes fases: modelagem do problema, escolha e desenvolvimento das heurísticas, determinação dos cenários de simulação, codificação, simulações, escrita de artigos e da dissertação. Para isso, deve-se utilizar o SUMO, bem como as ferramentas associadas a este simulador: *netedit*, *netgenerate* e *netconvert*. Na modelagem do problema, a solução deve considerar a inclinação das vias e a variação do peso do carrinho como restrições. Nesse sentido, é preciso calcular uma estimativa do esforço físico do roteiro de coleta, a qual depende do Trabalho da Força necessária para empurrar o carrinho.

O roteiro é definido de acordo com os pontos de coleta que o carrinheiro precisa passar, extraídos das plataformas de anúncios de materiais recicláveis, como Cataki e Destino Sustentável. Nesses sistemas, o carrinheiro pode atender ofertas de materiais, que deverão ficar reservados até o dia da coleta especificado no anúncio. Além disso, é possível verificar previamente o tipo de material que está sendo ofertado, a quantidade e a localização (latitude e longitude). A partir da localização do ponto de coleta, é possível verificar o ângulo de inclinação das vias que compõem o roteiro utilizando o *Open Street Map*.

O peso dos materiais que serão recolhidos em cada ponto de parada pode ser estimado antecipadamente, de acordo com o tipo e quantidade dos materiais. Sendo assim, é possível verificar o esforço físico necessário para percorrer cada via (aresta) e o melhor caminho deve ser gerado heuristicamente. Por conseguinte, a rota gerada deve ser utilizada nas soluções estática e dinâmica. A diferença entre as duas soluções é que na solução dinâmica, deve-se considerar o *feedback* do coletor sobre o peso do veículo ao longo do percurso para fazer o re-roteamento, caso seja necessário.

5. Conclusão

Este trabalho propôs um serviço para gerar rotas de coleta seletiva que otimizam o esforço físico exigido dos carrinheiros, considerando as características do cenário como distância, inclinação das vias, a utilização de veículos de propulsão humana e variação do peso do carrinho. A partir do desenvolvimento do serviço proposto, vislumbra-se a implementação de um sistema para disponibilizar o serviço de roteamento utilizando a API do Google Maps aos catadores.

Referências

[Ahmad et al. 2020] Ahmad, S., Imran, Jamil, F., Iqbal, N., and Kim, D. (2020). Optimal route recommendation for waste carrier vehicles for efficient waste collection: A step forward towards sustainable cities. *IEEE Access*, 8:77875–77887.

[Benjamin and Beasley 2010] Benjamin, A. M. and Beasley, J. E. (2010). Metaheuristics for the waste collection vehicle routing problem with time windows, driver rest period and multiple disposal facilities. *Computers and Operations Research*, 37(12):2270–2280.

[BRASIL 2010] BRASIL (2010). Política nacional de resíduos sólidos, lei nº 12.305, 2 de agosto de 2010. *Diário Oficial da República Federativa do Brasil*.

[Castilhos Junior et al. 2013] Castilhos Junior, A. B. d., Ramos, N. F., Alves, C. M., Forcellini, F. A., and Graciolli, O. D. (2013). Catadores de materiais recicláveis: análise das condições de trabalho e infraestrutura operacional no sul, sudeste e nordeste do brasil. *Ciência & saúde coletiva*, 18(11):3115–3124.

[CATAKI 2020] CATAKI (2020). Pimp my carroça. `https://www.cataki.org/pt/`. Acessado em: 2020-04-04.

[CEMPRE 2012] CEMPRE (2012). Política nacional de resíduos sólidos–a lei na prática. *São Paulo*.

[Destino Sustentável 2020] Destino Sustentável (2020). Destino sustentável: Reciclar para recriar o futuro. `https://www.destinosustentavel.org`. Acessado em: 2020-04-04.

[Fujdiak et al. 2016] Fujdiak, R., Masek, P., Mlynek, P., Misurec, J., and Olshannikova, E. (2016). Using genetic algorithm for advanced municipal waste collection in Smart City. *10th International Symposium on Communication Systems, Networks and Digital Signal Processing, CSNDSP 2016*.

[MNCR 2019] MNCR (2019). Movimento nacional dos catadores de materiais recicláveis. `http://www.mncr.org.br/`. Acessado em: 2020-05-21.

[SEBRAE 2017] SEBRAE (2017). Minha empresa sustentável: Cooperativa de reciclagem. *Cuiabá*.

[Silva et al. 2013] Silva, S. P., Goes, F. L., and Alvarez, A. R. (2013). Situação social das catadoras e dos catadores de material reciclável e reutilizável: Brasil. *IPEA*.

[Tavares et al. 2009] Tavares, G., Zsigraiova, Z., Semiao, V., and Carvalho, M. G. (2009). Optimisation of MSW collection routes for minimum fuel consumption using 3D GIS modelling. *Waste Management*, 29(3):1176–1185.

OmpTracing: Easy Profiling of OpenMP Programs

Vitoria Pinho[1], **Hervé Yviquel**[1], **Marcio Machado Pereira**[1], **Guido Araujo**[1]

[1] Institute of Computing – University of Campinas (UNICAMP)
Campinas – SP – Brazil

ra188511@students.ic.unicamp.br, herve.yviquel@ic.unicamp.br

mpereira@ic.unicamp.br, guido@ic.unicamp.br

Abstract. *One of the greatest challenges of modern computing is the development of software for parallel execution. To address such challenge, programmers use profiling tools to record relevant operations. Profilers can be used to analyze the execution of the application as they enable the programmer to check its performance hot spots and sources of overhead. This paper introduces the OmpTracing library, a lightweight tool that eases the task of profiling OpenMP based applications without the need to inject profiling code into the program. OmpTracing leverages on OMPT, an application programming interface that provides an introspection mechanism of the OpenMP runtime, and that enables to capture execution details of the parallelized application.*

Resumo. *Um dos maiores desafios da computação moderna é o desenvolvimento de aplicações para execução paralela. Diante desse desafio, programadores usam ferramentas de perfilamento para registrar operações relevantes. Estas ferramentas podem ser utilizadas para analisar a execução da aplicação, já que permitem que o programador indentifique pontos de ganho de performance e fontes de lentidão. Este artigo introduz a biblioteca OmpTracing, uma ferramenta leve que facilita a tarefa de perfilar aplicações OpenMP sem a necessidade de injetar código no programa. A ferramenta OmpTracing utiliza uma interface denominada OMPT, que fornece um mecanismo de análise do OpenMP e que permite capturar detalhes de execução da aplicação paralela.*

1. Introduction

Multicore and heterogeneous architectures have greatly increased the complexity of developing efficient programs. With that, many parallel programming models started to provide tools to assist the developer in the performance analysis and debugging of their programs. To enable this, OpenMP [Consortium et al. 2018] proposed OMPT [Eichenberger et al. 2013], an Application Programming Interface (API) that helps with the construction of performance analysis tools. OMPT is part of the OpenMP standard and supported by the entire OpenMP implementation.

This paper describes OmpTracing[1], an OMPT based library that enables the profiling of OpenMP parallel applications. The main goal of the OmpTracing project was to develop a lightweight profiling tool that does not need to inject code into the application. OmpTracing extracts information about the OpenMP events and generates a timeline for the events from registered OMPT callbacks. Examples of these callbacks are the creation

[1]OmpTracing is freely available at `https://ompcluster.gitlab.io/omptracing/`

of OpenMP tasks, the creation and destruction of OpenMP threads, and the definition of dependences between OpenMP tasks. In addition, OMPT provides information to callbacks that is very helpful not just for profiling the code but also to increase the knowledge about how the parallel runtime works. OmpTracing makes use of the Chrome Trace Event Profiling Tool [chr 2017], which enables a timeline graphic interface. OmpTracing generates tracing data in a JSON format text file which stores OpenMP events. This file can be loaded in the browser extension to visualize the timeline. While the timeline is practical and simplifies the view of the execution order of events on multi-core processors, other events like task dependences, which are essential to guaranty the semantic of parallel programs, are not easy to handle. For this reason, OmpTracing also generates a graph in dot format to visualize the tasks, their dependences, and execution times.

The paper is organized as follows: Section 2 describes general design concepts of OmpTracing, its functionalities and how it was organized. Section 3 provides some graphics and analyzes the results for a set of applications compiled with OmpTracing. Finally, Section 4 concludes the paper.

2. Methodology

This section presents the methodology used to build OmpTracing. It describes its architecture and the details of the implementation approach. The section is divided in two parts. First, it presents the construction of the timeline using the Chrome Tracing extension (Section 2.1). Second it describes the generation of the task dependence graph as a dot file (Section 2.2). Figure 1 shows an overview of OmpTracing workflow. It displays the OmpTracing interaction with OMPT, an interface of OpenMP runtime, and its outputs: the task graph and the timeline which can be visualized using Chrome Tracing extension. Also, it shows that application binary calls routines from OpenMp runtime.

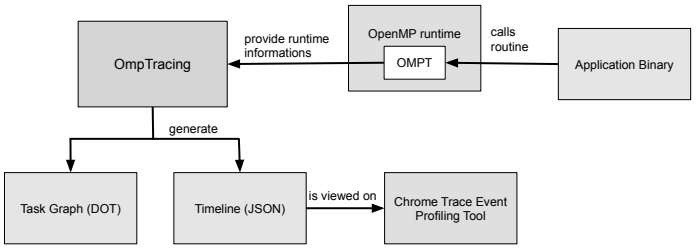


Figure 1. Overview of OmpTracing workflow

2.1. Tracing

OmpTracing leverages on the the Chrome's Trace Event Profiling Tool to produce a timeline of OpenMP events. This tool takes as input a JSON file containing a list of events, and uses a graphic viewer to display the corresponding event timeline. It was chosen because of its simplicity of use, accessibility, and facility to join events into a timeline, as successfully shown in the design of the Horovod [Sergeev and Balso 2018] tool. Using a graphical viewer to capture large traces of execution considerably facilitates the visualization of the parallelism, while reduces the complexity of the profile interpretation.

An OmpTracing timeline displays events, categories (that separate events), and extra information about a selected event. Figure 2 shows a timeline example of a countsort OpenMP application running with 2 threads. The registered events are the horizontal

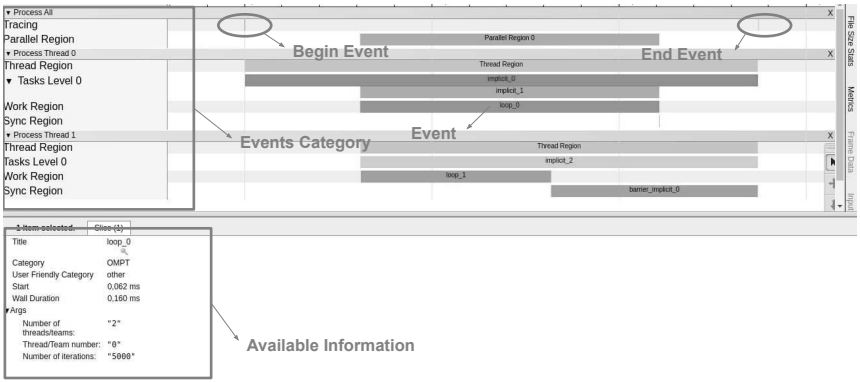


Figure 2. Tracing of the count-sort application.

colored bars. The left panel separates the events by their categories such as parallel region, thread region, and others. Each event has its category or subcategory that composes the event name. For example, the `implicit\_2` event in the figure has `Tasks Level 0` category (left panel) and implicit subcategory with id equals 2 (it's event name). The number in the task level category on the left panel represents the level of the task parent tree. When an event is clicked, all the available information is presented in the bottom section. The categories are divided by threads (`Process Thread` in the left panel) and the general categories are aligned in `Process All`. The tracing category registers the beginning and the end of the application. In the figure, the `begin` and the `end` event is circled.

OmpTracing uses the Pthread library to create a thread dedicated to saving the events into the JSON file by consuming a linked list. If those events were written directly when executing the OMPT callback, the thread could be blocked until finish writing each event, what could impact the execution time of the application. With all callbacks registered, a timeline created with OmpTracing can easily present a large amount of information, even with the simplest OpenMP program. Due to this, we introduced a configuration file which allows the user to select the specific callbacks he or she is interested to trace. There are several callbacks that can be registered by OmpTracing, each of them with specific OMPT information. The following callbacks can be registered: parallel regions, thread regions, work regions, master regions, and sync regions. Also, the beginning/ending of tasks as well as their dependences.

2.2. Task Graph

The programmer uses OpenMP clauses `depend(in: )` and `depend(out: )` to define the dependences between program tasks. The task graph is dynamically created by the OpenMP runtime during program execution, making it quite challenging to debug. Given the difficulty to visualize the task dependences through the timeline, OmpTracing produces a `DOT` file to present task dependences as a Directed Acyclic Graph (DAG). Figure 5 shows a graph example of a simple application that generated seven tasks. Each task is named by a unique identifier and presented with its timing information: starting time, ending time, and elapsed time. Dependences between tasks are represented by arrows. In our example, task 2 is dependent on tasks 0 and 1, and task 15 is dependent on task 13 and 14. All the other tasks are not dependent on any task.

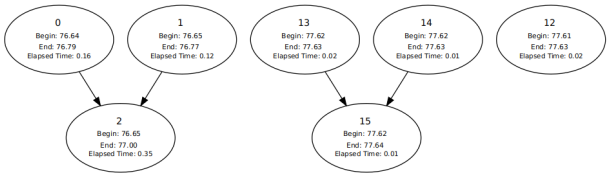


Figure 3. Example of the OmpTracing task graph.

The graph expresses all intended dependences between a predecessor task and a dependent task, therefore it can be used to verify if the task dependences specified by the programmer are correct. OpenMP programs that use the asynchronous task engine can quickly have a very complex dependency model, where race conditions can occur if the dependences are not specified correctly. The task dependency graph generated by OmpTracing shows the dependences that the OpenMP runtime library is generating for the application, facilitating the verification of the correctness of the program, difficult to discover through only functional tests.

3. Experimental Results

This section presents an experimental evaluation of the OmpTracing tool. The experiments were conducted on a machine with an Intel i5-8265U processor containing 4 cores with hyper-threading enabled. The operating system is Ubuntu 18.04.3 LTS. All applications were compiled using the Clang compiler version 10.0. OmpTracing has only been tested with the Clang compiler since the last GCC release version (GCC 10.1) does yet not support OMPT. To evaluate OmpTracing, two test scenarios were built. The first scenario (section 3.1), simpler, aims at exploring the OpenMP runtime and the various callbacks exposed by the OMPT interface. The second scenario (section 3.2), more realistic, uses a parameterized benchmark that explores a variety of patterns in parallel applications.

3.1. First Scenario

In the first scenario, OmpTracing was used to profile a simple count-sort parallel application designed to explore a parallel execution based on the OpenMP runtime library. Figure 2 shows the resulting trace in the Chrome GUI. The count-sort application executes a loop with 5000 iterations, and implements a `parallel for` directive to distribute the computation between two threads. An implicit task is generated by each thread, and is synchronized with the other using implicit barriers at the end of the parallel region. Thread 0 is the initial thread and thread 1 is a worker. Note that this information is not present in the figure because the threads are not selected in the Chrome GUI. In Fig. 2, the loop_0 section was selected and the number of iterations was showed in the bottom of the display. A work-sharing region has no barrier on entry, however, an implied barrier exists at the end, unless a `nowait` clause is specified. These barriers are displayed in the sync region category and identified as implicit barriers.

3.2. Task Bench Scenario

To evaluate how does OmpTracing performs with a more realistic benchmark, the Task Bench [Slaughter et al. 2019] benchmark was chosen. Task Bench is a parameterized

benchmark, designed to explore the performance of parallel programming systems in various application scenarios. Task Bench was implemented in 17 programming systems, but we only used it with OpenMP. It allows a wide variety of dependence patterns to be implemented, such as: trivial, no_comm, stencil_1d, and others. Tasks run a configurable set of kernels, which allow execution to be: compute-bound, memory-bound, communication-bound, or runtime overhead-bound (with empty tasks). In our Task Bench setup all tests used busy_wait kernels with 1 billion iterations and 4 worker units. The parameters width and steps represent, respectively, the degree of parallelism and the number of time steps.

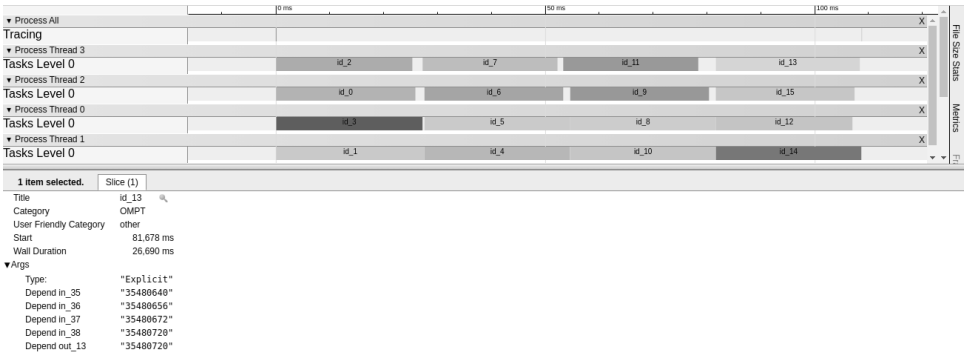


Figure 4. Tracing of Task Bench with stencil dependences

Figure 3.2 shows the timeline for the chosen Task Bench stencil_1d dependence pattern runs in parallel with 16 task events, each task being executed after the previous one. The task labeled with id_13 was clicked, and thus the task information is presented at the bottom of the display, showing the task type (explicit task) as well all depend clause events of this task. Each depend clause event has a unique id (the number on the depend in/out label) and its value is the unique id of the dependence data.

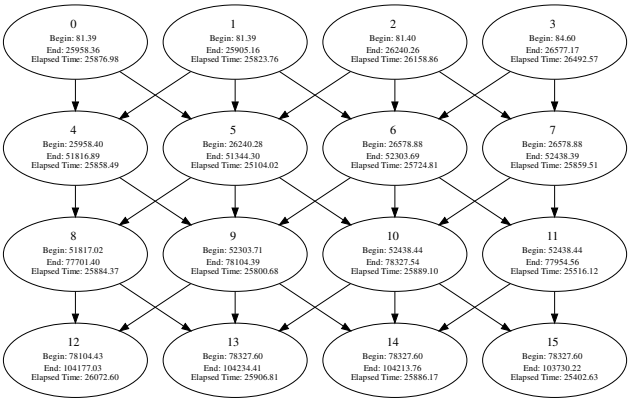


Figure 5. Task-dependence graph of Task Bench with Stencil_1d dependence pattern.

Figure 5 shows the generated task-dependence graph. The result shows how OmpTracing is able to graphically expose the dependences of task-based OpenMP programs to developers. The impact of OmpTracing on the execution time of Task Bench is presented in Table 1. The execution times reported in the table correspond to the average

total tasks	time [s]	OmpTracing time [s]	overhead (%)
16	18,56	19,36	4,3
64	78,69	81,69	3,8
256	301,54	320,25	6,2

Table 1. Compare Task Bench time with OmpTracing.

of 5 measures and the variation represents the overhead (in percent) to the execution time when profiling with OmpTracing. The results show that OmpTracing adds only a small overhead to the execution time of the application: at most 7% for 256 tasks executed in about 5 minutes.

4. Conclusion

Manual optimization of parallel applications has become increasingly difficult due to the growing complexity of applications. Automated optimization strategies have achieved good results. However, the use of tools that assist the optimization process becomes more and more essential. Profilers aim at solving this problem by providing metrics for optimization. OmpTracing has been added to this list of tools, providing a simple graphical interface to assist the developer in identifying the bottlenecks of the application without the need for additional code instrumentation. OmpTracing associates the execution time to the corresponding OpenMP tasks of the parallel application, thus allowing the developer to see where and how the execution time is spent. The utilization of OMPT interface permits OmpTracing to profile OpenMP application with limited overhead. The OMPT callback registration allows programmers to explore the events executing within the OpenMP runtime library and reduces the complexity of building a first-party tool. The results show that OmpTracing is a powerful profiling and tracing analysis tool and an effective manner to expose the task dependences of an OpenMP application. As future work we plan to support the tracing of target device offloading, after which OmpTracing will be published officially as an LLVM project. Moreover, we plan to test it with larger parallel applications to search for bugs and inconsistencies, and to verify its effectiveness.

References

(2017). The Trace Event Profiling Tool (about:tracing). (Date last accessed 1-July-2020).

Consortium, O. et al. (2018). Openmp specification version 5.0. Technical report, Technical Report. OpenMP Consortium. https://www. openmp. org/wp-content âl.

Eichenberger, A. E., Mellor-Crummey, J., Schulz, M., Wong, M., Copty, N., Dietrich, R., Liu, X., Loh, E., and Lorenz, D. (2013). Ompt: An openmp tools application programming interface for performance analysis. In *International Workshop on OpenMP*, pages 171–185. Springer.

Sergeev, A. and Balso, M. D. (2018). Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*.

Slaughter, E., Wu, W., Fu, Y., Brandenburg, L., Garcia, N., Kautz, W., Marx, E., Morris, K. S., Lee, W., Cao, Q., et al. (2019). Task Bench: A Parameterized Benchmark for Evaluating Parallel Runtime Performance. *arXiv preprint arXiv:1908.05790*.

Sistema operacional ONOS aplicado à SDN/NFV

Fernando Henrique Santorsula[1]

[1]Universidade Estadual de Campinas - (UNICAMP)
Av. Albert Einstein, Nº 400 - Cidade Universitária, Campinas - SP, 13083-852

[2]Faculdade de Engenharia Elétrica e de Computação - (FEEC)
f208918@dac.unicamp.br

Abstract. *This article will describe the entire history, installation, connectivity tests and features of the ONOS network operating system (Open Network Operating System), which is an extremely robust and versatile SDN controller, used by several companies and educational institutions around the world , working for new projects and research, still in this article, concepts and characteristics about SDN and NFV will be addressed.*

Resumo. *Este artigo irá descrever toda a história, instalação, testes de conectividade e recursos do sistema operacional de rede ONOS (Open Network Operating System), que é um controlador SDN extremamente robusto e versátil, utilizado por diversas empresas e instituições de ensino ao redor do mundo, atuando para novos projetos e pesquisas, ainda neste artigo, será abordado conceitos e características sobre SDN e NFV.*

1. Introdução ao ONOS

O projeto do sistema ONOS foi iniciado em 2012 pela liderança de Pankaj Berde (arquiteto de software do Open Networking Lab), o sistema ONOS foi oficialmente lançado em 2014 pelo Open Networking Lab, em parceria com algumas grandes empresas, tais como: AT&T, Cisco, NEC, Verison, NTT Communications, entre outras. O ONOS nasceu totalmente com seu código aberto e foi liberado para a The Linux Foundation, que se juntou ao projeto, desenvolvendo melhorias e correções.

O ONOS - (Open Network Operating System), é um sistema distribuído ou seja! uma coleção de dispositivos autônomos conectados por uma rede de comunicação, o ONOS foi desenvolvido para atuar como controlador SDN (Software Defined Networks), o ONOS foi projetado especificamente para atender a escalabilidade e alta disponibilidade de ambientes de redes SDN.

O ONOS se comporta como um sistema operacional de rede, fazendo a separação do plano de controle e dados, podendo ser utilizado em redes de longa distância e para provedores de serviços que são transportados em grandes redes. O ONOS é um software totalmente open source, mantido pela The Linux Foundation.

2. Características sobre a instalação do ONOS

O ONOS pode ser facilmente instalado e utilizado na arquitetura x86, entre outras, em especial em switch’s white box, atuando na comutação de camada 2 e camada 3, em ambientes pequenos e grandes como um data center, o ONOS se destaca com o uso de hardware "aberto", os mais utilizados são os da empresa Lanner que pode contar com vários modelos de switch, roteadores, servidores, podendo ter interfaces de

1 Gbe, 10 GbE e 40 GbE, aplicado para alta disponibilidade, alto desempenho e excelente custo-benefício, incluindo soluções de sobreposição de virtualização de rede que funcionam extremamente bem com controladores SDN como ONOS, para ter êxito na instalação, basta seguir os requisitos que estão presentes na tabela 1º

Sistema operacional utilizado: Ubuntu Server LTS – 16.04
Configuração de Hardware:
Processador Intel Core i5
2GB de memória RAM
1 NIC (Modo Brigde)
50GB de HD
Plataforma de instalação: Virtual (Oracle VirtualBOX – Versão: 5.2.42)

Tabela 1º

3. Procedimento de instalação

a-) Atualizar o sistema, seus pacotes e reiniciar o sistema:

apt update && apt upgrade -y && reboot

b-) Instalar os pacotes auxiliares: Nmap (Snnifer), Editor de textos (VIM), Ferramentas de Redes (Net-tools), Monitor de Sistema (Htop):

apt install nmap vim net-tools htop -y

c-) Instale os pacotes que serão utilizados pelo ONOS

apt-get update && apt-get install maven git openjdk-8-jre openjdk-8-jdk unzip -y

d-) Faça o download da versão 1.12.0 do ONOS e depois descompacte o arquivo:

onos-1.12.0.tar.gz

wget https://itfusion.com.br/onos/onos-1.12.0.tar.gz

tar zxf onos-1.12.0.tar.gz

e-) Acesse o diretório bin do ONOS e execute os seguintes comandos:

cd onos-2.0.0/bin/

export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64

./onos-service

f-) Após abrir o terminal do ONOS, execute o comando abaixo para ativar as aplicações:

apps -a -s
app activate org.onosproject.fwd
app activate org.onosproject.openflow

g-) Acesse o ONOS pela interface web com seu determinado endereço de IP, exemplo:
http://SEU-IP:8181/onos/ui/login.html
http://192.168.2.111:8181/onos/ui/login.html

Observações:
Neste artigo, o endereço de rede do servidor ONOS e Mininet são:
ONOS: 192.168.2.111/24
Mininet: 192.168.2.110/24

Este usuário e senha são definidos de forma default, após a instalação do ONOS.
Usuário: onos
Senha: rocks

Observação 1º:

Para testes de conectividade, no próximo tópico **h-)** será necessário ter uma OUTRA máquina virtual ou física com o Mininet instalado, caso não tenha o Mininet instalado, basta acessar o link abaixo, onde irá conter um tutorial de instalação do Mininet, para concluir os testes práticos com o controlador SDN "ONOS."

Link de instalação do Mininet:

http://fhs.pro.br/?page_id=2816

h-) Acesse a máquina (Mininet) – 192.168.2.110 e execute os comandos a seguir:

sudo mn –controller=remove,ip=192.168.2.111

h1 ping h2 -c 10

h2 ping h1 -c 10

Observação 2º:

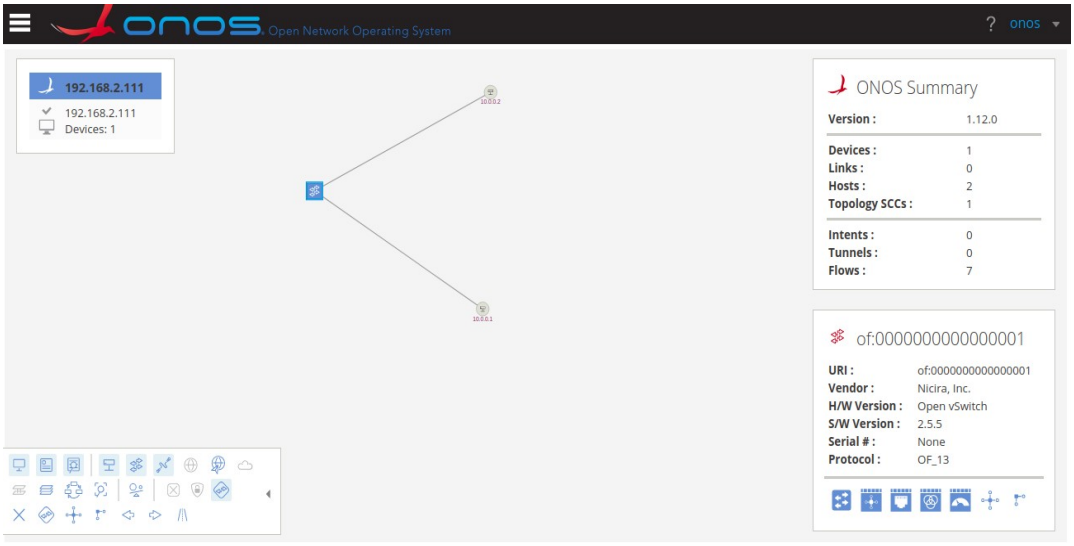
Cada comando deverá ser executado de forma única e pressionando ENTER para confirmar cada operação, conforme as próximas ilustrações.

```
root@mininet:~#  sudo mn --controller=remote,ip=192.168.2.111
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

```
mininet> h1 ping h2 -c 10
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=9.11 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.548 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.097 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.097 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.084 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.098 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.089 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.103 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.091 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9000ms
rtt min/avg/max/mdev = 0.075/1.039/9.115/2.695 ms
mininet> h2 ping h1 -c 10
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.051 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.116 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.091 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.090 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.100 ms
64 bytes from 10.0.0.1: icmp_seq=6 ttl=64 time=0.091 ms
64 bytes from 10.0.0.1: icmp_seq=7 ttl=64 time=0.096 ms
64 bytes from 10.0.0.1: icmp_seq=8 ttl=64 time=0.100 ms
64 bytes from 10.0.0.1: icmp_seq=9 ttl=64 time=0.121 ms
64 bytes from 10.0.0.1: icmp_seq=10 ttl=64 time=0.092 ms

--- 10.0.0.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9101ms
rtt min/avg/max/mdev = 0.051/0.094/0.121/0.021 ms
mininet>
```



4. Aplicações do ONOS

O ONOS possui inúmeras aplicações que podem ser utilizadas para simular novas topologias de redes que utilizam a arquitetura SDN, as mesmas iremos abordar mais adiante neste artigo, para ficar mais simples a compreensão, ilustramos um ambiente corporativo, onde possui uma solução de Segurança da Informação, mais precisamente um “Firewall”, onde o mesmo será ilustrado no modo “tradicional” e no modo “controlador”, utilizando o ONOS, vejamos:

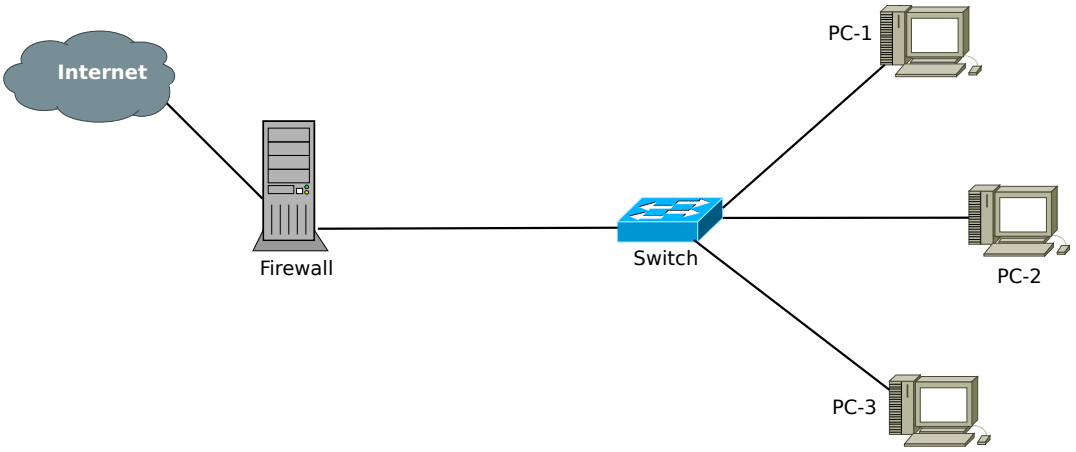


Figura 1

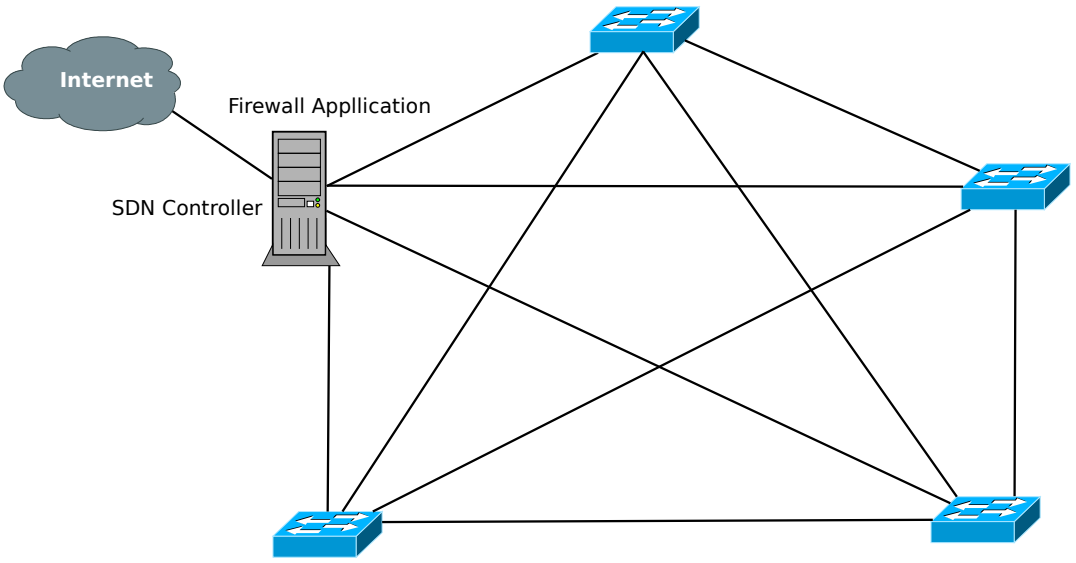
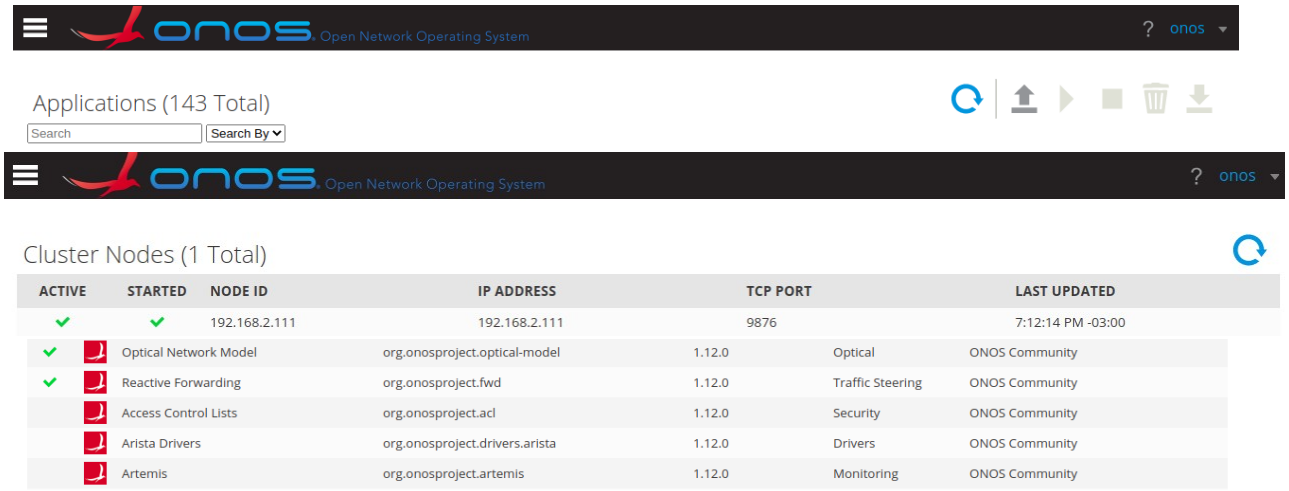


Figura 2

5. Aplicações

Ao acessar a interface web do ONOS, podemos ver as inúmeras aplicações que podemos utilizar com este magnifico controlador, na versão versão do ONOS: 1.12.0, a mesma possui 143 aplicações, conforme podemos ver na figura 3, onde as que estão habilitadas em verde, estão ativos no ONOS.

Figura 3



O ONOS por padrão possui inúmeros recursos de fácil acesso, auxiliando o gestor da aplicação fazer configurações rápidas e visualizar recursos em atividade, conforme a ilustração a seguir, podemos visualizar o nó principal e ativo no ONOS.

Figura 4

Nesta área do controlador, podemos visualizar de forma detalhada a quantidade de dispositivos que temos em no controlador ONOS, em nosso ambiente temos um equipamento (Switch) do fabricante Nicira, Inc. em operação:

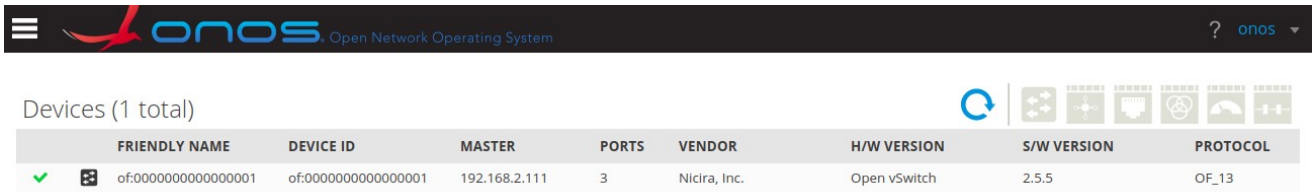


Figura 5

Nesta outra área do controlador, podemos visualizar de forma detalhada a quantidade de hosts que temos conectados em nosso controlador, neste caso temos 2 equipamentos conectados e que testamos sua conectividade com o Mininet:

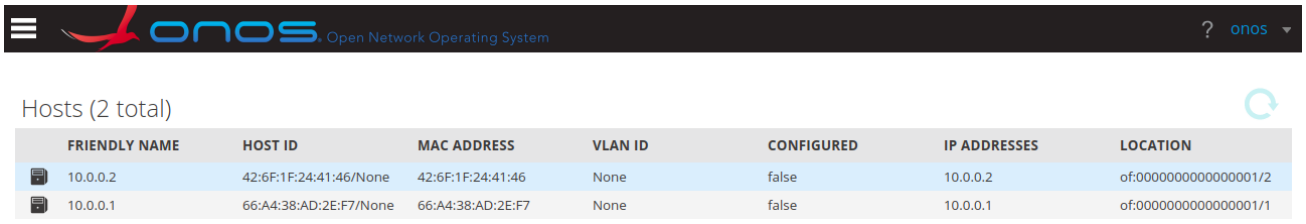


Figura 6

6. Hardware Open Source

O ONOS possui suporte nativo à vários fabricantes de hardware aberto, separamos um modelo especifico, onde o ONOS pode ser facilmente instalado, este equipamento é desenvolvido pela empresa Lanner, que é uma das referências mundiais em hardware aberto, veja um exemplo ilustrado de equipamentos da empresa Lanner, mas figuras 7 e 8.

Figura 7

Figura 8

7. Introdução a SDN

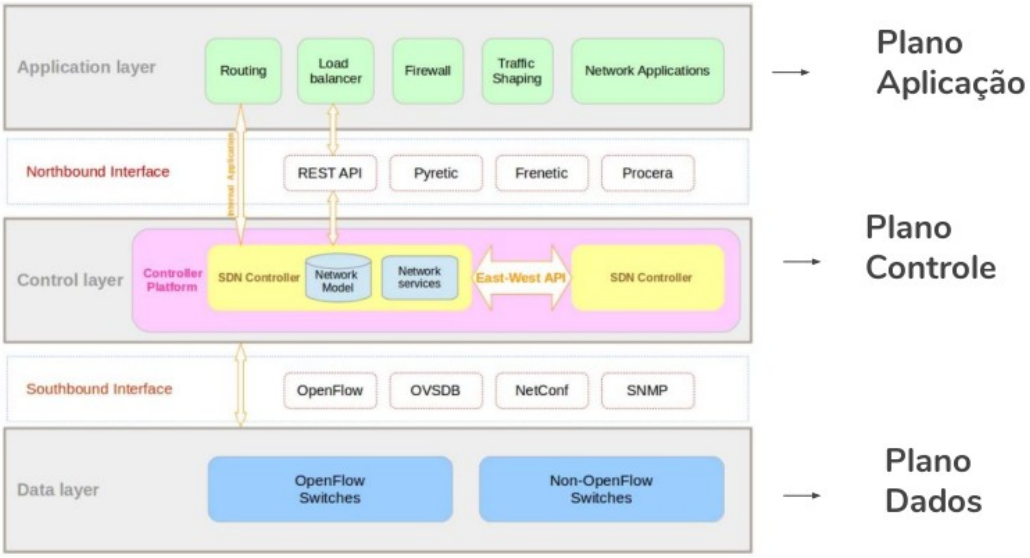
Nas atuais redes de computadores empresariais, instituições sem fins lucrativos e afins, possui a finalidade de compartilhar recursos, transportar informações de uma origem até um destino, sendo assim, as redes computacionais, possuem camadas embutidas nos dispositivos de rede, com a finalidade de melhorar a flexibilidade, inovação e evolução tecnológica, entre outros fatores positivos para o negócio ou pesquisa.

Porém esses dispositivos possuem configurações que devem ser feitas manualmente por profissionais especializados, utilizando um software do fabricante, ou seja! Uma camada de controle decide o que fazer com os pacotes de informação, gestão de tráfego e as demais decisões à ser tomada em determinada rede ou conjunto de redes computacionais.

O SDN (Software Defined Networking), possui um paradigma diferente do tradicional, onde existe uma separação da camada de controle, da camada de dados, melhorando os recursos de uma ou mais redes, simplificando o gerenciamento, reduzindo os custos de operação, melhorando a inovação e evolução tecnológica.

A arquitetura do SDN fica em um nível mais baixo, que é representada pela infraestrutura ou camada de dados, onde é nesta camada que os dispositivos encaminham o tráfego baseado nas decisões da camada de controle. A comunicação de

dados e a camada de controle pode ser utilizada por vários protocolos, tais como: OpenFlow, SNMP, BGP, NetConf entre outros, na figura 9 podemos compreender melhor o novo conceito de redes definidas por software.



Imagens do artigo: Distributed SDN Control: Survey, Taxonomy, and Challenges
IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 20, NO. 1, FIRST QUARTER 2018

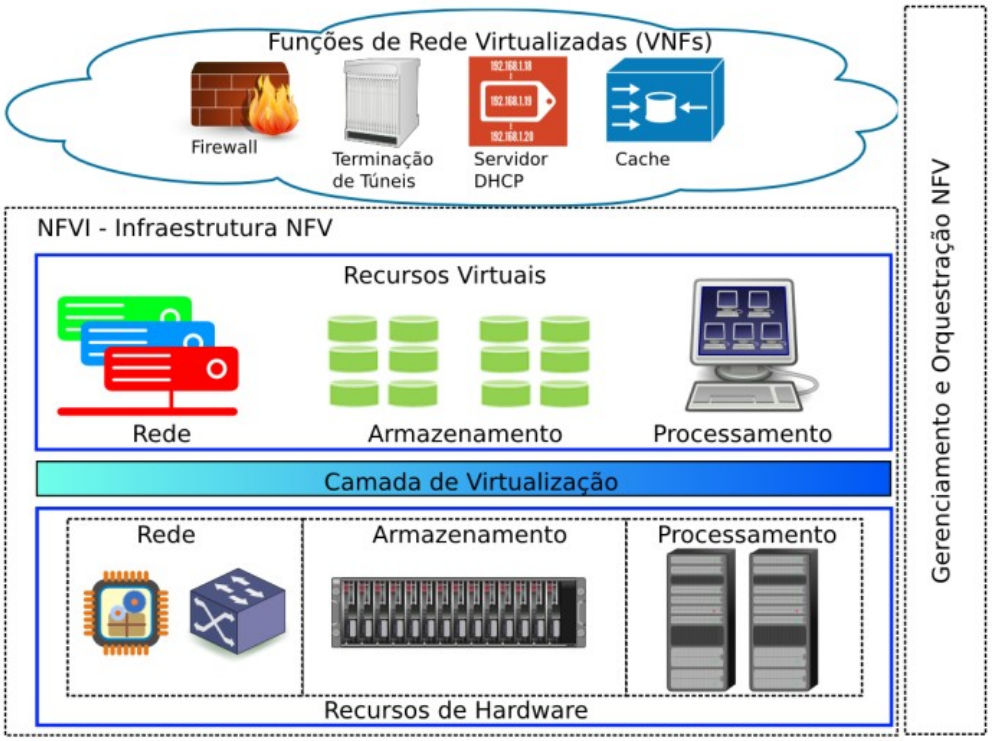
Figura 9

8. Introdução a NFV

A abordagem técnica sobre a NFV (Network Functions Virtualization), pode ser definida como a substituição do hardware dedicado (switch, roteadores, balanceadores de carga, entre outros equipamentos), que possuem alto custo e um software proprietário para fazer sua gestão e administração, com o NFV é possível fazer o que chamamos de "orquestração" de vários dispositivos em apenas um equipamento de hardware robusto e apropriado para esta finalidade.

O NFV é executada mais especificamente utilizando um sistema operacional que coordena dispositivos virtuais, trabalhando em modo "produção" em uma determinada rede ou várias redes de pequeno até grande porte. Podemos referenciar as máquinas virtuais para ter uma melhor compreensão, porém neste caso, o NFV faz a virtualização de equipamentos de rede, vejamos na figura 10, o modo tradicional sem o uso da tecnologia NFV e com o uso da tecnologia NFV.

Figura 10



9. Conclusão

Em outras palavras, é possível que um ISP (Internet Solution Provider), possa construir uma solução SDN/NFV em um ambiente real. ONOS é implantado como um serviço em um cluster de servidores, onde o controlador ONOS é executado em cada servidor.

A simetria de implantação é uma consideração de design importante, pois permite um failover rápido no caso de uma falha do servidor ONOS. O operador de rede pode adicionar servidores de forma incremental, sem interrupção, conforme necessário para capacidade adicional do plano de controle.

Várias instâncias do ONOS podem trabalham juntas, para criar o que parece para o resto da rede e aplicativos como uma plataforma única. Os aplicativos e dispositivos de rede não precisam saber se estão trabalhando com uma única instância ou com várias instâncias do ONOS. Este recurso torna o ONOS escalonável - pode-se escalar a capacidade do ONOS perfeitamente. É o Distributed Core que faz o trabalho pesado para concretizar esses recursos, esta é a eficiência do controlador ONOS e o futuro das redes SDN aplicado ao negócio e a pesquisa cientifica.

10. Referências

He, K., Khalid, J., Gember-Jacobson, A., Das, S., Prakash, C., Akella, A., Li, L. E., and Thottan, M. (2015). Measuring control plane latency in sdn-enabled switches. In Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, page 25. ACM. Jain, R. and Paul, S. (2013). Network virtualization and software defined networking for cloud computing: a survey. Communications Magazine, IEEE, 51(11):24–31. Kim, H. and Feamster, N. (2013). Improving network

management with software defined networking. Communications Magazine, IEEE, 51(2):114–119.

Kreutz, D., Ramos, F. M., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky,

S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. pro-

ceedings of the IEEE, 103(1):14–76. Lantz, B., Heller, B., and McKeown, N. (2010). A network in a laptop: Rapid prototyping for software-defined networks. In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Hotnets-IX, pages 19:1–19:6, New York, NY, USA. ACM.

Akyildiz, I. F., Lee, A., Wang, P., Luo, M., and Chou, W. (2014). A roadmap for traffic

engineering in sdn-openflow networks. Computer Networks, 71:1–30. Benson, T., Akella, A., and Maltz, D. A. (2009). Unraveling the complexity of network

management. In NSDI, pages 335–348. Bifulco, R. and Matsiuk, A. (2015). Towards scalable sdn switches: Enabling faster flow table entries installation. In Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, pages 343–344. ACM. Dusi, M., Bifulco, R., Gringoli, F., and Schneider, F. (2014). Reactive logic in software- defined networking: Measuring flow-table requirements. In Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International, pages 340–345. IEEE.

Federated Learning para estimativa de tráfego de veículos em tempo real

Matteus V. S. Silva[1], **Felipe C. Bertocco, Gustavo Garcia**[1], **Luiz F. Bittencourt**[1], **Adín Ramirez Rivera**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (Unicamp)
Av. Albert Einstein, 1251 – Campinas – SP – Brasil

{vargas.simao,felipecapiteli,ggarcpro}@gmail.com,{bit,adin}@ic.unicamp.br

Abstract. *Smart Vehicles are an example of a"Smart Environment". Cloud Computing, with Machine Learning, is used to process data in this context, but it is estimated that, in the future, each vehicle will generate very large datasets that will cause strong pressure on the communication infrastructure, in addition to high latency. To contour these drawbacks, this paper presents BRUSILI, which uses Federated Learning to estimate the level of traffic congestion locally, restricting the communications to learning data exchange between vehicles. Preliminary deployment and evaluation show that after ten rounds of communication to exchange knowledge, vehicles are able to estimate the traffic levels on their road with an average of 74% accuracy.*

Resumo. *Veículos Inteligentes são um exemplo de composição de um "Ambiente Inteligente". A Computação em Nuvem, com Machine Learning, é usada para processar dados nesse contexto, mas estima-se que, futuramente, cada veículo gerará uma conjunto de dados muito grande, o que causará uma forte pressão na infraestrutura de comunicação, além de alta latência. Para contornar tais limitações, este trabalho apresenta o BRUSILI, que usa Federated Learning para estimar o nível de congestionamento de sua via localmente, restringindo a comunicação a apenas trocas de dados de aprendizado entre os veículos. Uma implementação e avaliação preliminares apontam que após dez rodadas de comunicação para troca de conhecimento, os veículos conseguem estimar com 74% de precisão média os níveis de trânsito em sua via.*

1. Introdução

A ampla difusão dos sensores e dispositivos da *Internet of Things* (IoT), unidos à Inteligência Artificial (IA), tem criado os chamados "Ambientes Inteligentes" [Valerio et al. 2017]. Os Veículos Inteligentes são exemplos desse ambiente. Segundo a Intel, cada veículo gerará cerca de 4.000 GB de dados por dia, o que é equivalente a dados móveis gerados por quase 3.000 pessoas [Sensors 2017]. Isso significa pressão sem precedentes na infraestrutura de comunicação, armazenamento e computação.

Há propostas com Computação em Nuvem [Zhang et al. 2010] para oferecer suporte a veículos inteligentes com recursos de *Machine Learning* (ML) [Kehoe et al. 2015]. A longo prazo, isso traz problemas como alta latência, proteção frágil de privacidade e transmissão massiva de dados com alto consumo de largura de banda, além dos *Data Centers* em Nuvem estarem muitas vezes geograficamente

distantes do usuários finais [Barik et al. 2020]. Portanto, processamento e confiabilidade em tempo real não podem ser garantidos.

Já com a *Edge Computing*, é possível aplicar essa análise de dados localmente com ML, o mais próximo possível dos dispositivos finais [Valerio et al. 2017]. Partindo desse princípio, em trazer a inteligência direto nos veículos, garantindo a baixa latência, pouco uso da rede e processamento em tempo real, este artigo apresenta o BRUSILI (*BRoadly connected vehicle Units Share their Individual Learning Interactively*).

O BRUSILI aplica o método de aprendizado distribuído e colaborativo, *Federated Learning* (FL), fazendo com que cada veículo consiga utilizar o seu próprio modelo de aprendizado local para estimar o nível de congestionamento na sua via, para então unir o seu conhecimento com os demais veículos.

Uma implementação preliminar foi realizada com a criação de um *setup* da rede e da aplicação em simulador, onde as etapas do processo de aprendizado foram avaliadas. Como contribuição, além dos pontos de melhoria no uso da rede supracitados, busca-se trazer serviços outrora executados na Nuvem ou na *Fog* diretamente para os veículos da rede em uma estratégia distribuída, mas visando manter a eficiência, como a acurácia da aplicação, em níveis similares a infraestruturas centralizadas.

2. A aplicação

O BRUSILI é simulado com o uso da pilha de ferramentas OMNET++, SUMO e *Veins*. O OMNET++ é usado para a modelagem e programação da rede, o SUMO fornece o mapa e a trajetória dos veículos irão atuar e o *Veins* conecta o SUMO e o OMNET++, permitindo a modelagem e programação de redes veiculares e implementa o protocolo de redes veiculares $802.11p$. Durante a simulação, *scripts* de FL, desenvolvidos com o *framework Tensorflow*, são executados via chamadas de sistema para tecer as operações de aprendizado (seção 2.2).

Por fim, o cenário onde o BRUSILI foi simulado é um trecho de $2.032\ km^2$ do centro da cidade de *Melbourne*, Austrália, com um total de 548 veículos. Isso torna a sua densidade alta e passível de congestionamentos.

Durante a simulação, o BRUSILI faz com que cada veículo, individualmente, consiga estimar o nível de congestionamento na sua via e possa maximizar o fluxo de tráfego por meio do compartilhamento desse conhecimento com os demais veículos, sem aumentar a sobrecarga de informações rede.

2.1. Inicialização da aplicação

Assim que um veículo adentra a rede, três operações são imediatamente executadas: (i) a requisição do modelo de aprendizado, (ii) a inscrição na tabela global e (iii) a instanciação do BDL.

Antes de descrever as operações, é preciso salientar que o modelo de aprendizado é elaborado na Nuvem e enviado à *Fog*. A Nuvem aqui é uma instância do *Google Colaboratory*, ambiente que fornece recursos de armazenamento, computação e processamento mais elevados.

Isto posto, na primeira operação, o veículo requisita uma cópia do modelo de aprendizado (seção 2.2) à *Fog*, representada aqui pela infraestrutura RSU (*RoadSide*

Identificação	Localização
Id	Via
	Logradouro
	Rota
	Velocidade
	Tipo de veículo

Tabela 1. Tabela de inscrição global. Cada veículo se identifica e fornece dados básicos de localização.

Nível de Congestionamento	Densidade	Velocidade
Livre	[29-37] vei/km	[48-81] km/h
Moderado	[37-50] vei/km	[24-64] km/h
Congestionado	>=50 vei/km	[0-40] km/h

Tabela 2. Modelo de congestionamento LOS F em detalhes

Unit), posicionada no centro do mapa para que possa ser usada na estimativa de tráfego. Na segunda operação, o veículo adiciona seus dados básicos de identificação e localização a uma tabela global (Tabela 1). Essa tabela controla quem está na região e será usada durante o processo de aprendizado.

Já a terceira operação instancia, em cada veículo, o seu respectivo BDL que corresponde a um arquivo *.csv* com os dados de identificação do veículo (*id*) e as informações da quantidade de veículos e velocidade média na via.

Essa estrutura do BDL é derivado do *Level-of-Service* (LOS) F [Rocha Filho et al. 2020], que representa uma métrica para classificar o fluxo de tráfego, definida pelo *Highway Capacity Manual* (HCM) [Council 2000]. Esse modelo de classificação considera que a velocidade e densidade de veículos na estrada indicam o nível de congestionamento (Tabela 2), a saber: (i) *Livre*, (ii) *Moderado* e (iii) *Congestionado*.

2.2. Funcionamento do aprendizado

O BRUSILI usa o método de aprendizado conhecido como *Federated Learning* (FL). O FL treina modelos de *Machine Learning* (ML) agregando uma grande quantidade de dados distribuídos em diferentes nós da borda rede. Trata-se de uma forma de aprendizado distribuído e colaborativo [McMahan and Ramage 2017].

O modelo de aprendizado corresponde a uma rede neural com duas entradas, correspondente aos valores de velocidade e densidade de veículos, seis camadas escondidas (*hidden layers*) com oito neurônios cada. Até esse ponto, os neurônios são ativados por *ReLU* [Russell and Norvig 2002]. A última camada possui um único neurônio que corresponde à saída, sendo essa ativada por *Sigmoid* [Santos et al. 2018]. Essa saída é um dos níveis de congestionamento indicados na da Tabela 2.

O modelo é compilado com o otimizador *Adam*, taxa de aprendizado de 0.01 e função de perda *Mean Squared Error*. Tanto os hiper-parâmetros quanto os seus respectivos valores foram determinados por meio do procedimento de *tuner* [Abadi et al. 2016], o qual testa diversas combinações a fim de encontrar a configuração ideal que possa atin-

gir a melhor acurácia possível. A construção do modelo de aprendizado e o *tuner* foram feitos na Nuvem. Por haver dados de entrada e rótulos de saída bem-definidos, o tipo de aprendizado usado aqui é o supervisionado.

O BRUSILI utiliza o FL da seguinte forma em cada veículo:

1. O veículo requisita da *Fog* uma cópia do modelo de aprendizado, transformando-o no seu *modelo local*. Hiper-parâmetros e pesos já vem pré-definidos, mas ainda não ajustados;
2. A cada 1 segundo de tempo de simulação, o veículo chama a operação *Handle Position Update*. Essa operação atualiza a tabela de inscrição com os novos dados de localização;
3. O veículo consulta a tabela e conta quantos veículos estão na mesma via e suas respectivas velocidades. Com isso, ele incrementa o BDL;
4. Assim que o BDL é atualizado, imediatamente é feito o ajuste (*fit*) do modelo local. Cada veículo faz essa operação individualmente. Essa operação ajusta os pesos dos modelos locais;
5. A cada 25 segundos de tempo de simulação é chamada a *agregação*. Por critério, são escolhidos os modelos cujos os BDL's possuem as maiores quantidades de registro. Esse critério pressupõe que tais veículos também possuem maior variabilidade, ou seja, passaram pelas situações de tráfego descritas na tabela 2;
6. A agregação é feita na *Fog* da seguinte forma:
   (a) A *Fog* solicita os modelos locais dos veículos, conforme o critério em 5;
   (b) É executado o algoritmo FedAvg (*Federated Averaging)* [McMahan and Ramage 2017]. Dos modelos locais selecionados, o FedAvg tira a média dos pesos, gera um novo modelo global e o reenvia aos veículos na rede, reiniciando todo o processo;

O FL permite então que se opere com os dados dos veículos na rede, sem a necessidade de compartilhá-los com outra infraestrutura ou outros veículos, de forma colaborativa e distribuída.

3. Resultados preliminares

As avaliações preliminares realizadas levam em questão os seguintes aspectos:

- Cada veículo já possui um histórico de navegação que compõe os primeiros registros do seu BDL [McMahan and Ramage 2017]. Para simular esse histórico foram distribuídas partes do *Dataset* de Hinnerup, Dinamarca [Tönjes et al. 2014]. Esse *dataset* é elaborado de modo similar a Tabela LOS F;
- A acurácia dos modelos locais a cada agregação e a acurácia global;

Os seguintes parâmetros foram utilizados na avaliação preliminar:

1. O tempo de simulação foi de 250 segundos. A cada 25 segundos, a agregação é realizada, totalizando 10 agregações;
2. Cada modelo local é ajustado com 500 *epochs*, com *EarlyStopping* ativado e monitorando a acurácia;
3. O histórico de cada modelo local foi catalogado a fim de checar se as agregações estavam melhorando a acurácia;
4. Após as agregações, avaliou-se a acurácia global;

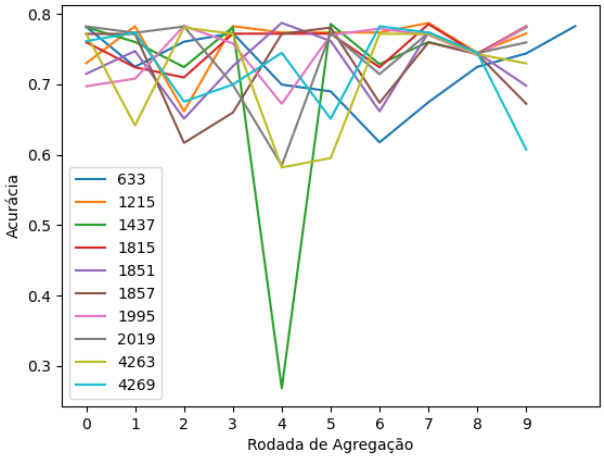


Figura 1. Gráfico geral que representa o desempenho dos veículos a cada rodada de agregação. A legenda indica o *id* de cada veículo.

Com o *EarlyStopping* ativado, os modelos locais foram capazes de serem treinados com 1% do valor definido para as *epochs*. No caso, apenas cinco *epochs*. Nesse teste, foram agregados apenas onze maiores modelos locais no total.

De modo geral, já na primeira agregação (rodada 0) os valores de acurácia ficaram entre 0.7 e 0.8. O padrão que se seguiu para a maioria dos modelos foi que a partir da rodada 1 até a rodada 6 ocorreram muitas variações, chegando até a 0.3 de acurácia em um dos modelos em uma rodada específica.

Analisando apenas o *dataset* que alcançou a pior acurácia em uma das rodadas (Figura 1, *id* 1437), notamos que o veículo vivenciou apenas uma única situação de trânsito, o *Congested*. Isso indica que a política de seleção dos *datasets* pela quantidade de registros pode ser insuficiente. É necessário selecionar mais veículos que passaram por situações diferenciadas dentro da simulação. Possivelmente isso também explica a instabilidade dos demais veículos nas rodadas do meio, como mostra o gráfico da Figura 1.

Dos modelos selecionados, apenas 36.3% apresentaram queda de valores ao término de todas as rodadas. Usualmente, a rodada 9, por mais que não alcance o maior valor de todos, é aquela que apresenta apenas crescimento. Ao término de todas as rodadas de agregação, o modelo global atingiu uma média de 0.74 de acurácia. Isso significa que, ao término das rodadas de agregação, o modelo que irá vigorar possui essa média de acurácia.

4. Conclusão

A estrutura da aplicação está bem definida no que concerne as operações a serem executadas, estrutura e funcionamento da rede, e como o aprendizado se encaixa nesse cenário. Os primeiros testes são considerados interessantes no sentido de que o aprendizado colaborativo e distribuído com FL é factível. Por mais que demonstre maior complexidade na sua implementação, isso é compensado pela redução de tráfego de dados pela rede,

apenas modelos de aprendizado, além da infraestrutura de suporte na *Fog* ser utilizado poucas vezes, o que torna os veículos mais independentes.

Os testes realizados são preliminares, mas úteis para entender o comportamento do FL e identificar potenciais pontos de melhoria. Como já mencionado na seção 3, durante algumas rodadas de agregação, os valores de acurácia dos veículos variam muito. Isso pode ser problemático nos momentos de predição. Apesar disso, a acurácia ao final de 0.74 pode ser considerada razoável.

Os próximos passos são os ajustes na política de seleção dos BDL's, o tempo entre uma agregação e outra, e quantas vezes a agregação deve ser realizada durante a simulação. Outra questão que deve ser avaliada é o uso de histórico para compor o BDL inicial contra a possibilidade de instanciar um novo assim que inicia a simulação.

Referências

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Barik, R. K., Priyadarshini, R., Lenka, R. K., Dubey, H., and Mankodiya, K. (2020). Fog computing architecture for scalable processing of geospatial big data. *International Journal of Applied Geospatial Research (IJAGR)*, 11(1):1–20.

Council, T. R. B. N. R. (2000). *Highway capacity manual*. TRB Business Office.

Kehoe, B., Patil, S., Abbeel, P., and Goldberg, K. (2015). A survey of research on cloud robotics and automation. *IEEE Transactions on automation science and engineering*, 12(2):398–409.

McMahan, B. and Ramage, D. (2017). Federated learning: Collaborative machine learning without centralized training data. *Google Research Blog*, 3.

Rocha Filho, G. P., Meneguette, R. I., Neto, J. R. T., Valejo, A., Weigang, L., Ueyama, J., Pessin, G., and Villas, L. A. (2020). Enhancing intelligence in traffic management systems to aid in vehicle traffic congestion problems in smart cities. *Ad Hoc Networks*, 107:102265.

Russell, S. and Norvig, P. (2002). Artificial intelligence: a modern approach.

Santos, M. R. M. d. et al. (2018). Identificação de textos em imagens captcha utilizando conceitos de aprendizado de máquina e redes neurais convolucionais.

Sensors, A. (2017). Electronics expo 2017. *Detroit, USA (14–15 June 2017)*.

Tönjes, R., Barnaghi, P., Ali, M., Mileo, A., Hauswirth, M., Ganz, F., Ganea, S., Kjærgaard, B., Kuemper, D., Nechifor, S., et al. (2014). Real time iot stream processing and large-scale data analytics for smart city applications. In *poster session, European Conference on Networks and Communications*. sn.

Valerio, L., Passarella, A., and Conti, M. (2017). A communication efficient distributed learning framework for smart environments. *Pervasive and Mobile Computing*, 41:46–68.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18.

Analysis of Solutions and Datasets for the Problem of Identifying Humpback Whales by Their Tails

Henrique da Fonseca Simões[1], **João Meidanis**[1]

[1]Institute of Computing – University of Campinas (UNICAMP)
Campinas – SP – Brazil

henrique.simoes@students.ic.unicamp.br, meidanis@ic.unicamp.br

Abstract. *Humpback whales (Megaptera novaeangliae) can be identified by their flukes' shape, markings, and scars. This allows researchers to track the individuals, and better understand the specie population dynamics. In this sense, the Kaggle platform hosted a competition from Nov. 2018 to Feb. 2019, in which an automatic whale identification model should be developed. In this research project, we aimed at analyzing and reproducing the competition top solutions' results. We found that the second and third solutions are more reproducible than the first. In addition, corrections of issues found in the dataset seem to improve the performance of the more reproducible solutions.*

Resumo. *Baleias jubarte (Megaptera novaeangliae) podem ser identificadas pelo formato, marcas e cicatrizes de suas caudas. Isto permite que pesquisadores rastreiem as baleias e melhor entendam a dinâmica populacional da espécie. Nesse sentido, a plataforma Kaggle sediou uma competição de novembro de 2018 a fevereiro de 2019, cujo objetivo era a criação de um modelo de identificação automatizada de baleias. Neste trabalho, buscamos analisar e reproduzir os resultados dos primeiros colocados da competição. Descobrimos que a segunda e terceira soluções são mais reprodutíveis que a primeira. Além disso, a correção de problemas encontrados no conjunto de dados parece levar a uma melhor performance das soluções mais reprodutíveis.*

1. Introduction

Humpback whales (*Megaptera novaeangliae*) were predominantly explored by the fishing industry between the 1860s and the late 1900s. In 1955, the International Whale Commission banned commercial hunting, and two decades later the species was declared endangered under the United States Endangered Species Act of 1973. This helped some subpopulations to start recovering [Stevick et al. 2003]. However, for other subpopulations, in western North Pacific and eastern North Atlantic, we still have limited data available to document the recovery process [Thomas et al. 2016].

A methodology used to analyze humpback whale population trends is to identify the individuals by photographs. This is possible because shapes, markings, and scars in the whales' tails are different from one whale to another [Katona et al. 1979]. Nevertheless, much human effort is necessary to classify the whale images this way. Therefore, from November 2018 to February 2019, the *Kaggle* platform hosted a competition in which the participants were challenged to develop a solution for automatic identification of whales by photographs of their flukes.

This work aimed at reproducing the results of the top-3 solutions in the aforementioned competition. In addition, since there existed mislabeled data and examples with additional, excessive difficulty factors, we created a corrected dataset and evaluated the analyzed solutions with it. Finally, we tried to improve the best solution in our reproductions by adapting it to use two techniques. Our code and resources can be found at `https://github.com/henriquesimoes/humpback`.

2. Competition Solutions

Many solutions developed in the competition used Machine Learning or, more specifically, Deep Learning techniques. This was the case also for the top-3 solutions, which achieved around 0.97 Mean Average Precision at 5 (MAP@5). We present next an overview of each analyzed solution. All three solutions used a similar data treatment pipeline: application of several image transformations to implement data augmentation, and image cropping with bounding boxes to focus on the whale's flukes.

The first-placed solution, designed by Jian Qiao, Peiyuan Liao, Thomas Tilli and Yiheng Wang, consists of an end-to-end learning architecture that directly classifies the input images in one of the 5004 whale identities. In order to do so, SENet-154 [Hu et al. 2018] is used as a backbone network to generate feature maps, from which fluke global and local features are extracted using a strategy from the field of person re-identification [Sun et al. 2018]. These extractions are trained using metric learning in the form of Triplet Loss [Schroff et al. 2015].

The second solution, designed by Tao Shen, also consists in an end-to-end model to classify the input images by using deep neural networks as backbones: ResNet-101 [He et al. 2016], SE-ResNet-101 [Hu et al. 2018, He et al. 2016], and SE-ResNeXt-101 [Xie et al. 2017, Hu et al. 2018]. In total, 10 models were trained using these backbones, varying the image size and pseudo-labels, and then combined in an ensemble. To train the networks, many loss functions are used: Triplet Loss [Schroff et al. 2015], ArcFace Loss [Deng et al. 2019], CosFace Loss [Wang et al. 2018a, Wang et al. 2018b], and an adaptation of FocalLoss [Lin et al. 2017].

Designed by Jinmo Park, the third-placed solution used a different approach. A deep architecture, consisting of DensetNet-121 [Huang et al. 2017], is used to generate 512-D feature vectors (embeddings), capable of summarizing the whale fluke characteristics. This architecture is trained (for 500 epochs) also with the ArcFace strategy to increase its discriminatory power [Deng et al. 2019]. Upon inference, a class embedding is defined as the feature center of all image embeddings of each class in the training set. Then, each test image embedding is compared to each class embedding via cosine similarity. An ensemble of three models, which vary in learning rate schedule and label grouping, is the final classification.

In our experiments, we used a server with an Intel Xeon Silver 4110 Octa Core 2.10GHz processor, with 126 GB RAM, and configured with 3 NVIDIA GeForce RTX 2080 Ti GPUs, with 11 GB of memory GDDR6 each. We followed the instructions provided by the authors, except that we had to sometimes reduce batch sizes to fit our computational resources. In addition, we trained each solution only once, because ensemble training requires a long time. Table 1 shows our reproduction results.

Table 1. Reproduction results for the top-3 solutions.

	1^{st}	2^{nd}	3^{rd}
Original	0.97309	0.97208	0.97113
Reproduction	0.86513	0.96910	0.96889

3. Data Correction

Since testing set labels were not available, we used only the training data in our data-corrected experiments, splitting them into train and validation sets. We discovered mislabeled examples with the help of the Kaggle forum posts, where competitors shared their findings during the competition. Mislabeling consisted in:

1. sets of examples from the same identity with different labels;
2. identifiable whales classified as a previously unknown whale (*new whale*);
3. examples from different identities with the same label.

We found 121 pairs and 4 trios of classes potentially in the first item. Among these, we confirmed that 108 pairs and all trios were duplicated. The correction consisted in merging the sets of images, keeping a single label. In the second item, we found 336 images potentially mislabeled as *new whale*, and confirmed that 164 were in fact from other classes. In the third situation, we found 29 examples, which had their labels corrected.

Besides mislabeling, there existed examples in the dataset with factors that could hinder the identification of the fluke patterns. These factors included occlusion — due to water aspersion and part of the tail being underwater —, poor photograph angle — with distortion of the markings and fluke shape —, and poor image resolution — photograph taken from far away or blurred. We created a test set called *hard* with these examples.

We also found 66 examples which were not adequate for training or inference. Most of them had two photographs or multiple flukes in the same image. We decided not to include them in the corrected dataset, since fixing them was not always possible. We then used the remaining (*standard*) examples for training and testing the original solutions again and our adaptations (see Section 4).

In our experiments, we used two random splits of 20% testing and 80% training images from the corrected set (Table 2). Due to merged classes and modified labels, the total number of classes in the corrected dataset dropped to 4887 (plus the *new whale*). In addition, there were test image classes in the splits that did not appear in the corresponding training set. In these situations, we considered the *new whale* as the correct class in the evaluation.

To reevaluate the studied solutions, due to time constraints, we chose to train only single models (not ensembles). Therefore, since the first solution did not use ensembles, we followed the same procedure as specified by the authors. For the second solution, we used the best model configuration (when evaluated in isolation), which was the ResNet-101. For the third solution, we used the learning rate scheduler configuration most often used in the ensemble models, training for 300 epochs. Reevaluation results are shown in Table 3.

Table 2. Number of classes, identifiable examples and *new whale* examples in the training and testing splits and in the *hard* set. Number of classes means the number of identities which had at least one example in the corresponding set.

	Split 1		Split 2		Hard set
	Train	Test	Train	Test	Test
Classes	3965	1747	3927	1805	1441
Identifiable examples	11161	2792	11139	2814	1869
New whale examples	4813	1201	4835	1179	3459
Total examples	15974	3993	15974	3993	5328

Table 3. Reevaluation results for the top-3 solutions using the corrected dataset. The reported values are the mean of the results of the splits and their standard error.

Solution	Standard set	Hard set
1^{st}	0.72 ± 0.02	0.826 ± 0.002
2^{nd}	0.984 ± 0.001	0.930 ± 0.008
3^{rd}	0.982 ± 0.005	0.93 ± 0.01

4. Adaptations

The second solution showed slightly better scores in our experiments, besides having more stable results when compared to the first solution. We thus tried to improve it even further by using: Contrast Normalization [Goodfellow et al. 2016, Section 12.2.1.1] and Stochastic Weight Averaging (SWA) [Izmailov et al. 2018].

Contrast Normalization is a strategy that (safely) removes a source of variation that a neural network would have to become invariant to. We tested two types of normalization: Global Contrast Normalization (GCN) and Local Contrast Normalization (LCN). In GCN, standard deviation and mean of the image pixels are computed over the entire image at once, and then used to normalize it. On the other hand, in LCN, a pixel value is updated only with the information of its surrounding pixels (*i.e.* in a small window).

In our experiments, we used the simplest formulation for GCN, where no bias term is added and the scaling factor is 1. For LCN, we used a 9×9 window (with padding), bias term $\lambda = 1$, and scaling factor $s = 1$. Since the original solution already normalized the input images using the image statistics of the ImageNet dataset [Russakovsky et al. 2015], we also tested it without any normalization. Table 4 shows the results for each approach. As can be seen, no significant improvements resulted.

SWA is a technique that assembles several models during the converge procedure by averaging their parameters in all layers of the network. Izmailov and colleagues show that an improvement on generalization can be achieved using SWA with constant learning rate and Stochastic Gradient Descent (SGD), with different neural networks [Izmailov et al. 2018]. We applied SWA from the 75^{th} to the 100^{th} epoch with learning rate $\epsilon = 10^{-4}$. Moreover, we used *cosine annealing* for 10 epochs. The results are shown in Table 4. No noticeable gain showed up in the standard set. In the hard set, the average improvement is within the error margins.

Table 4. Contrast Normalization and SWA results in the corrected dataset. The reported values are the mean of the results of the splits and their standard error.

Solution	Standard set	Hard set
2^{nd}	0.984 ± 0.001	0.930 ± 0.008
No CN	0.9847 ± 0.0005	0.9346 ± 0.0005
GCN	0.9837 ± 0.0004	0.928 ± 0.005
LCN	0.9829 ± 0.0006	0.922 ± 0.005
SWA	0.983 ± 0.001	0.936 ± 0.002

5. Conclusion

We analyzed the top-3 solutions in the Kaggle competition on humpback whale identification and successfully reproduced the second and third solution results. The first solution showed to be more difficult to reproduce, at least with our hardware configuration and experience with the model training.

Moreover, there existed mislabeling issues in the dataset, in addition to difficult examples — in which the difficulty relied on other aspects besides fluke pattern identification, such as occlusion. We reevaluated the solutions using a corrected set and found out that an improvement of around 0.01 over the original dataset results could be achieved, even using only single models (not ensembles) for both the second and third solutions. This indicates that the solution improvements were limited by the dataset intrinsic issues, and possibly not by the developed architectures.

Finally, we tested the incorporation of Contrast Normalization and Stochastic Weight Averaging approaches in the second solution. However, none of the strategies led to significant improvements.

Acknowledgments

This work was supported[1] by the São Paulo Research Foundation (FAPESP) under grant no. 2019/11386-3. We also appreciate the availability of the computational resources, described in Section 2, acquired through FAPESP support under grant no. 2018/00031-7, and maintained by the Institute of Computing/UNICAMP. We also thank the Student Support Service (SAE)/UNICAMP for the support given.

References

Deng, J., Guo, J., Xue, N., and Zafeiriou, S. (2019). Arcface: Additive angular margin loss for deep face recognition. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4690–4699.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

[1]Opinions, hypothesis and conclusions, or recommendations expressed in this material are responsibility of the authors, and do not necessarily reflect FAPESP's point of view.

Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7132–7141.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708.

Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., and Wilson, A. G. (2018). Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*.

Katona, S., Baxter, B., Brazier, O., Kraus, S., Perkins, J., and Whitehead, H. (1979). Identification of humpback whales by fluke photographs. In *Behavior of marine animals*, pages 33–44. Springer.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). Focal loss for dense object detection. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2980–2988.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). ImageNet large scale visual recognition challenge. *International Journal on Computer Vision*, 115(3):211–252.

Schroff, F., Kalenichenko, D., and Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823.

Stevick, P. T., Allen, J., Clapham, P. J., Friday, N., Katona, S. K., Larsen, F., Lien, J., Mattila, D. K., Palsbøll, P. J., Sigurjónsson, J., Smith, T. D., Øien, N., and Hammond, P. S. (2003). North Atlantic humpback whale abundance and rate of increase four decades after protection from whaling. *Marine Ecology Progress Series*, 258:263–273.

Sun, Y., Zheng, L., Yang, Y., Tian, Q., and Wang, S. (2018). Beyond part models: Person retrieval with refined part pooling (and a strong convolutional baseline). In *European Conference on Computer Vision (ECCV)*, pages 480–496.

Thomas, P. O., Reeves, R. R., and Brownell Jr, R. L. (2016). Status of the world's baleen whales. *Marine Mammal Science*, 32(2):682–734.

Wang, F., Cheng, J., Liu, W., and Liu, H. (2018a). Additive Margin Softmax for Face Verification. *IEEE Signal Processing Letters*, 25(7):926–930.

Wang, H., Wang, Y., Zhou, Z., Ji, X., Gong, D., Zhou, J., Li, Z., and Liu, W. (2018b). Cosface: Large margin cosine loss for deep face recognition. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5265–5274.

Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. (2017). Aggregated residual transformations for deep neural networks. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1492–1500.

Caracterização de Reticulados para Modelos Criptográficos Pós-Quânticos

Tomás S. R. Silva[1], Ricardo Dahab[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP).

Abstract. *Secure communication has been a persistent issue throughout human history. In fact, the constant development of measures and countermeasures related to the establishment of safe methods of communication over the years is remarkable. Currently, the imminent appearance of a sufficiently powerful quantum computer is a threat to current cryptographic methods. Thus, a class of post-quantum cryptographic schemes arises, aiming to suppress such a threat. Within these schemes, lattices appear with great protagonism. Our objective in this work is, therefore, to investigate certain mathematical properties of the lattice structure.*

Resumo. *Comunicação segura é um assunto persistente ao longo da história da humanidade. De fato, é notável o constante desenvolvimento de medidas e contramedidas relacionadas ao estabelecimento de métodos seguros de comunicação no decorrer dos anos. Atualmente, o surgimento iminente de um computador quântico suficientemente potente é uma ameaça para os métodos criptográficos em vigência. Assim, surge uma classe de esquemas criptográficos pós-quânticos, visando suprimir tal ameaça. Dentro desses esquemas, figuram com grande protagonismo os reticulados. Nosso objetivo nesse trabalho é, portanto, investigar certas propriedades matemáticas da estrutura dos reticulados.*

1. Introdução

O próximo grande salto na capacidade de transmitir e processar informações poderá vir com o surgimento de um computador quântico suficientemente potente (Yanofsky 2007), i.e. capaz de executar para tamanhos grandes de entrada rotinas tais quais o algoritmo de Shor (Shor 1997) para fatoração de números, ou o algoritmo de Grover (Grover 1996) para buscas lineares. Um computador quântico capaz de executar tais algoritmos põe em cheque as técnicas clássicas de criptografia, uma vez que estas se baseiam em problemas que se tornariam tratáveis em tempo hábil por um computador quântico dadas as dimensões atuais de chaves, como o problema da fatoração de inteiros ou o problema do logaritmo discreto, como é bem descrito por Coron e Wege em (Jean-Sebastien Coron (Gemplus), Benne de Weger (TUE) 2007).

Dessa forma, o uso de algoritmos clássicos da criptografia moderna precisa ser revisado. Algoritmos como o de Diffie-Hellman (Diffie and Hellman 1976) para troca de chaves e o RSA (Rivest et al. 1978) para criptografia de chave pública, por

exemplo, têm sua segurança garantida em grande parte pela dificuldade do problema do logaritmo discreto e o problema da fatoração em primos, que podem ser atacados em tempo hábil pelo algoritmo de Shor(Shor 1997) dadas as dimensões atuais de chave. Já o algoritmo empregado pela cifra Rijndael, que é utilizada no AES (information processing standards publication 197 2001) para criptografia de chave privada, poderia ser atacado com uma busca linear intensiva, performada por exemplo pelo algoritmo de Grover (Grover 1996) dados os tamanhos vigentes de chave. Nota-se, portanto, que o advento eventual de um computador quântico suficientemente potente motiva a adaptação das primitivas criptográficas que garantem a segurança de uma cifra, seja em relação à dimensão ou à essência de tais primitivas.

Duas alternativas surgem para enfrentar um possível advento de tal computador quântico: a primeira alternativa é aumentarmos as dimensões das chaves utilizadas nos esquemas modernos, o que seria computacionalmente custoso e, de certa forma, ineficiente dadas as limitações dos computadores modernos. A segunda alternativa é desenvolver novas técnicas de criptografia que consideram em sua elaboração problemas intratáveis até mesmo por algoritmos quânticos dadas certas dimensões de chaves. A classe de esquemas criptográficos que tem sido desenvolvidos para enfrentar ataques desse futuro computador quântico compõem a chamada *Criptografia Pós-Quântica*.

A preocupação com o surgimento iminente de um computador quântico pode ser ratificada pelo concurso lançado em 2017 pelo "NIST - National Institute of Standards and Technology", o instituto de padrões e tecnologia dos Estados Unidos (NIST 2017), que solicitou submissões de esquemas criptográficos pós-quânticos. Atualmente o concurso encontra-se na terceira rodada de avaliação (NIST 2020), com finalistas divulgados em Setembro de 2020. Nessa terceira rodada do concurso, figuram entre os finalistas e candidatos alternativos 15 esquemas criptográficos pós-quânticos, tanto de encriptação/KEM quanto de assinatura digital. Uma distribuição geral desses esquemas pode ser vista nos diagramas a seguir.

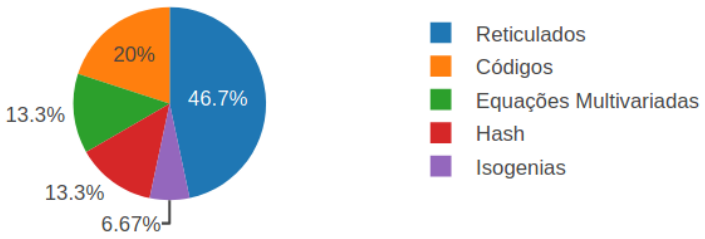


Figura 1. Distribuição dos esquemas na terceira rodada do concurso NIST(NIST 2020) por área dos problemas.

Podemos observar do concurso do NIST (NIST 2017) que esquemas criptográficos baseados em reticulados compõe uma das mais promissoras apostas na construção de esquemas criptográficos pós-quânticos. Dessa forma, nosso objetivo con-

siste em analisar certas propriedades matemáticas da estrutura dos reticulados cuja escolha tem potencial de torná-los mais efetivos em aplicações criptográficas.

2. Preliminares

Definição 2.1 (Reticulado)**.** *Um reticulado Λ é um subgrupo aditivo discreto do Espaço Euclidiano n-dimensional, $\mathbb{R}^n$, isto é, $\Lambda \subset \mathbb{R}^n$. Uma base B de um reticulado Λ é um conjunto maximal de k vetores linearmente independentes $\{b_1, b_2, ..., b_k\} \in \Lambda$, de tamanho n com $k \leq n$. O posto do reticulado resultante $\Lambda(B)$ é k. Quando $k = n$, diz-se que o reticulado $\Lambda(B)$ tem posto completo. Ademais, um reticulado admite uma representação matricial dada por*

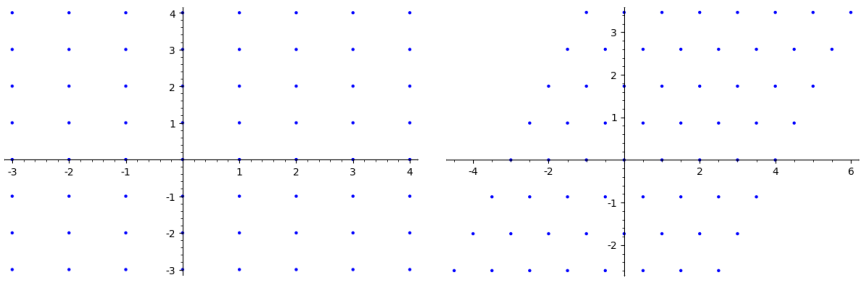
$$\Lambda(B) = \{Bx : x \in \mathbb{Z}^n\}$$

Sendo x um vetor qualquer de inteiros de tamanho n e B é a chama de matriz geradora de Λ, formada pelos vetores da base do reticulado

$$B = \begin{pmatrix} -b_1- \\ -b_2- \\ \cdots \\ -b_k- \end{pmatrix}$$

Note que podemos construir infinitas bases para um reticulado em posse de uma matriz B. Seja U uma matriz tal que $\det(U) = \pm 1$ uma matriz de coeficientes inteiros $n \times k$, temos que $B' = BU$ também será base do mesmo reticulado Λ.

A seguir, ilustramos dois exemplos de reticulados de posto completo construídos em $\mathbb{R}^2$: o reticulado inteiro $\mathbb{Z}^2$, cujas bases são $\{(1,0),(0,1)\}$; e o reticulado hexagonal, cujas bases são $\{(1,0),(\sqrt{3}/2,1/2)\}$.



(a) $\Lambda = \langle (1,0),(0,1) \rangle$ (b) $\Lambda = \langle (1,0),(\sqrt{3}/2,1/2) \rangle$

Figura 2. Reticulados em $\mathbb{R}^2$

Definição 2.2 (Paralelepípedo Fundamental)**.** *Dado um reticulado Λ e uma de suas bases B, chama-se de paralelepípedo fundamental do reticulado a região definida por*

$$\mathcal{P}(B) = \theta_1 b_1 + ... + \theta_n b_n, \theta_i \in (0,1) \forall i \in (1,n)$$

.

Definição 2.3 (Volume do Reticulado)**.** *Dado um reticulado Λ e um de seus paralelepípedos fundamentais $\mathcal{P}(B)$, o volume dessa região define o que chamamos de volume do reticulado, dado por*

$$Vol(\Lambda) = Vol(\mathcal{P}(B)) = \sqrt{|\det(B.B^t)|}$$

.

Note que para um mesmo reticulado Λ, temos infinitas bases e, consequentemente, infinitos paralelepípedos fundamentais. Porém, para um mesmo reticulado Λ o seu volume é constante, independentemente da escolha da base. Na figura a seguir, ilustramos o reticulado hexagonal em $\mathbb{R}^2$ e dois de seus paralelepípedos fundamentais: um considerando $B_1 = \{(1,0), (\sqrt{3}/2, 1/2)\}$ e outro considerando $B_2 = \{(0,\sqrt{3}), (1/2, \sqrt{3}/2)\}$. Note que tando a região delimitada em verde quanto a região delimitada em rosa tem o mesmo volume, que é igual ao volume do reticulado hexagonal, i.e., $\text{Vol}(\Lambda(B_1)) = \text{Vol}(\Lambda(B_2)) = \frac{1}{2}$.

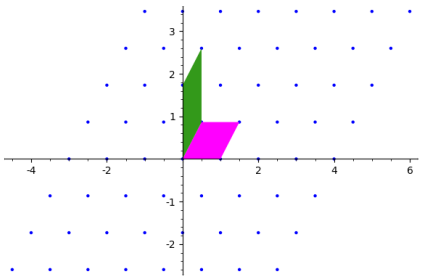


Figura 3. Reticulado Hexagonal e dois de seus paralelepípedos fundamentais.

Definição 2.4 (Distância Mínima)**.** *Dado um reticulado Λ, define-se sua distância mínima como sendo o valor da norma euclidiana do seu menor vetor não nulo*

$$\lambda(\Lambda) = \lambda = min||v||, v \in \Lambda \setminus \{0\}$$

Equivalentemente, uma vez que todo reticulado é um grupo aditivo discreto, podemos escrever que

$$\lambda(\Lambda) = min||x - y||; x, y \in \Lambda$$

Ademais, um problema bastante interessante que motiva o estudo de reticulados é o problema do empacotamento de bolas n-dimensionais. Uma das questões lançadas é como podemos arranjar bolas n-dimensionais de mesmo raio de forma a preencher o maior espaço possível, i.e., qual configuração das bolas nos fornece uma maior densidade. A forma de utilizar reticulados para resolvermos esse problema é colocando os centros das bolas em pontos do reticulados, e analisar as relações volumétricas decorrentes. O exemplo abaixo ilustra o empacotamento de bolas de raio $r = 1/2$ no reticulado inteiro, apresentado previamente na figura 2(a).

É evidente que o raio máximo que bolas n-dimensionais centradas em pontos de um reticulado Λ podem assumir para serem disjuntas duas a duas é $r < \lambda/2$.

Definição 2.5 (Densidade de Empacotamento)**.** *Seja $\Lambda \subset \mathbb{R}^n$ um reticulado, e seja $B_r(\boldsymbol{x})$ a bola centrada em $\boldsymbol{x}$ de raio r. Dizemos que a densidade de empacotamento de Λ é*

$$\Delta(\Lambda) = \frac{Vol(B_{\lambda/2}(0))}{Vol(\Lambda)}$$

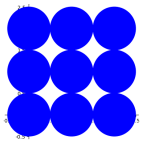


Figura 4. Empacotamento de bolas no reticulado inteiro.

É importante ressaltarmos que encontrar reticulados com densidade de empacotamento ótima em dada dimensão é um problema em aberto na maioria casos. Só são conhecidos tais reticulados com empacotamento ótimo nas dimensões 1 a 8 e 24, sendo que o trabalho que levou a solução do problema em 24 dimensões foi publicado apenas em 2017 (Cohn et al. 2017).

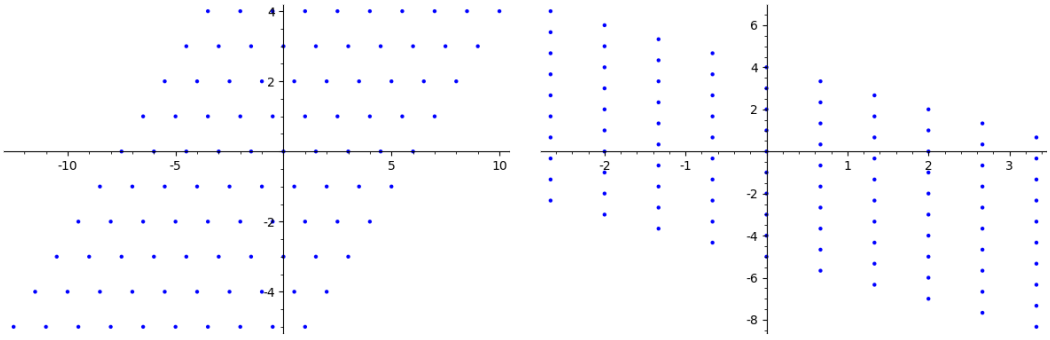
Outra noção bastante importante acerca dos reticulados é a de dualidade. Isso porque em certos aspectos a geometria do reticulado dual é equivalente à do reticulado original, o que pode facilitar algumas análises.

Definição 2.6 (Reticulado Dual)**.** *Dado um reticulado $\Lambda \subset \mathbb{R}^n$ de posto completo, seu reticulado dual é definido como*

$$\Lambda^* = \{x \in \mathbb{R}^n : \langle x, y \rangle \in \mathbb{Z}, \forall y \in \Lambda\}$$

Lemma 1. *Dada uma base B para o reticulado de posto completo $\Lambda \subset \mathbb{R}^n$, podemos construir uma base para seu dual Λ^* como sendo $B^* = (B^{-1})^T$*

Considere como exemplo a figura a seguir, onde são apresentados o reticulado $\Lambda = \langle (1,1), (3/2, 0) \rangle$ e seu reticulado dual $\Lambda^* = \langle (0,1), (-2/3, 2/3) \rangle$



(a) $\Lambda = \langle (1,1), (3/2, 0) \rangle$ (b) $\Lambda^* = \langle (0,1), (-2/3, 2/3) \rangle$

Figura 5. Um reticulado e seu dual em $\mathbb{R}^2$

A estrutura geométrica dos reticulados tem sido usada na construção de primitivas criptográficas desde o trabalho revolucionário de Ajtai em 1996 (Ajtai 1996). É provado neste trabalho que construções criptográficas sobre reticulados são tão difíceis de serem solucionadas quanto é difícil de resolver computacionalmente problemas NP-completos que exploram as características dos reticulados.

3. Resultados

Neste trabalho, investigamos principalmente o chamado *parâmetro de suavização*. Esses estudos foram em grande parte motivados pela dissertação de mestrado de F. Meneghetti

(MENEGHETTI 2020). Para além das definições preliminares outrora apresentadas na Seção 2, considere as seguintes definições iniciais, relacionadas à distribuição Gaussiana sobre o espaço real n-dimensional.

Definição 3.7 (Função Gaussiana)**.** *A função gaussiana parametrizada por $s \in \mathbb{R}_+^*$ é $\rho_s : \mathbb{R}^n \to \mathbb{R}$ dada por*

$$\rho_s(\boldsymbol{v}) = exp\left(-\pi\left(\frac{||\boldsymbol{v}||^2}{s^2}\right)\right)$$

Definição 3.8 (Massa Gaussiana)**.** *Seja $\Lambda \subset \mathbb{R}^n$ um reticulado de posto completo, e $\boldsymbol{c} \in \mathbb{R}$. A massa gaussiana de parâmetro $s \in \mathbb{R}_+^*$ é*

$$\rho_s(\Lambda + \boldsymbol{c}) = \sum_{\boldsymbol{v} \in (\Lambda + \boldsymbol{c})} \rho_s(\boldsymbol{v}) = \sum_{\boldsymbol{v} \in (\Lambda)} \rho_s(\boldsymbol{v} + \boldsymbol{c})$$

Com isso em vista, podemos definir o parâmetro de suavização para um reticulado $\Lambda \subset \mathbb{R}^n$.

Definição 3.9 (Parâmetro de Suavização)**.** *Seja $\Lambda \subset \mathbb{R}^n$ um reticulado de posto completo e Λ^* seu dual, e $\epsilon > 0$ um número real. Definimos o parâmetro de suavização de Λ como*

$$\eta_\epsilon(\Lambda) = \mathit{min}\{s > 0 | \rho_{1/s}(\Lambda^* \setminus \mathbf{0}) \leq \epsilon\}$$

Seja a *distância estatística* $\Delta(X, Y)$ entre duas variáveis aleatórias X e Y sobre um conjunto contável A definida como $\Delta(X, Y) = \frac{1}{2}\sum_{a \in A} |Pr(X = a) - Pr(Y = a)|$. Temos intuitivamente que dado um ponto aleatório de Λ, se acrescentarmos a esse ponto um ruído gaussiano de raio $\eta_\epsilon(\Lambda)$, a distribuição resultante será $\epsilon/2$ distante estatisticamente da distribuição uniforme sobre $\mathbb{R}^n$. Alguns limitantes conhecidos para o parâmetro de suavização são apresentados no teorema abaixo.

Teorema 3.1 ((Micciancio and Regev 2004))**.** *Para $\Lambda \in \mathbb{R}^n$ temos que*

1. $\eta_\epsilon(\Lambda) \leq \frac{\sqrt{n}}{\lambda(\Lambda *)}$, $\epsilon = 2^{-n}$
2. $\eta_\epsilon(\Lambda) \leq \sqrt{\frac{ln(2n(1+1/\epsilon))}{\pi}}.\lambda_n(\Lambda)$, $\epsilon > 0$

Uma caracterização alternativa para o parâmetro de suavização pode ser dada a partir do conceito de sobreposição de bolas, que foi introduzida por C. Peikert et al em (Chung et al. 2014).

Definição 3.10 ((Chung et al. 2014)Sobreposição de Bolas)**.** *Seja $r > 0$ e $\Lambda \subset \mathbb{R}^n$ um reticulado de posto completo. Definimos*

$$\mathit{Overlap}(\Lambda, r) := \frac{\mathit{Vol}(\cup_{\boldsymbol{v} \in \Lambda \setminus \mathbf{0}}(B_r(\mathbf{0}) \cap B_r(\boldsymbol{v}))}{\mathit{Vol}(B_r(\mathbf{0}))}$$

De forma mais direta, podemos dizer que o parâmetro Overlap(Λ, r) de um reticulado Λ representa a proporção de volume das bolas de raio r centradas em pontos de $\Lambda \setminus 0$ que intersectam a bola de raio r centrada na origem. Alguns limitantes para o parâmetro Overlap(Λ, r) são fornecidos no teorema a seguir.

Teorema 3.2 ((Chung et al. 2014)). *Seja $\Lambda \subset \mathbb{R}^n$ um reticulado de posto completo. Para $\epsilon \in (2^{o(-n)}, 1/3)$ e $r_\epsilon = \sqrt{\frac{n}{2\pi}}\frac{1}{2\eta_\epsilon(\Lambda^*)}$, temos:*

1. *Para $0 \leq r \leq r_\epsilon$ temos que Overlap$(\Lambda, r) \leq 2\epsilon$*
2. *Para todo $r \geq 2r_\epsilon(1+\delta)$, onde $\delta = \sqrt{\frac{3}{2n}ln\frac{4}{\epsilon}}$, temos que Overlap$(\Lambda, r) \geq \epsilon/2$*

Seja $\Lambda \subset \mathbb{R}^n$ um reticulado de posto completo. Sabemos da definição 2.5 que a densidade de empacotamento do reticulado é o valor $\Delta(\Lambda) = \frac{\text{Vol}(B_{\lambda/2}(\mathbf{0}))}{\text{Vol}(\Lambda)}$

Considere uma bola de raio $r > 0$ arbitrário centrada na origem. Podemos escrever seu volume como

$$\text{Vol}(B_r(\mathbf{0})) = t(r)\Delta(\Lambda)\text{Vol}(\Lambda) \tag{1}$$

Sendo que

$$t(r) = \frac{\text{Vol}(B_r(\mathbf{0}))}{\text{Vol}(B_{\lambda/2}(\mathbf{0}))} = \left(\frac{r}{\lambda/2}\right)^n = \left(\frac{2r}{\lambda}\right)^n \tag{2}$$

Assim, das equações (1) e (2), podemos escrever que

$$\text{Vol}(B_r(\mathbf{0})) = \left(\frac{2r}{\lambda}\right)^n \Delta(\Lambda)\text{Vol}(\Lambda) \tag{3}$$

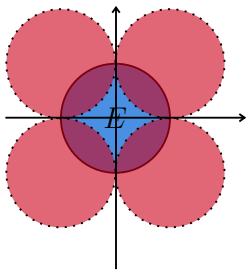


Figura 6. Introdução da região E

Considere agora que o volume entre a intersecção de $B_r(\mathbf{0})$ com todas as bolas de raio r centradas em pontos de Λ pode ser calculada da seguinte forma:

$$\text{Vol}(\cup_{\boldsymbol{v}\in\Lambda\setminus\mathbf{0}}(B_r(\mathbf{0}) \cap B_r(\boldsymbol{v})) = \text{Vol}(B_r(\mathbf{0})) - Vol(E) \tag{4}$$

Sendo que E é a região complementar das intersecções das bolas de raio r centradas em pontos de $\Lambda \setminus 0$ em relação a bola de raio r centrada na origem, como é mostrado na Figura 6. Considere agora que $\boldsymbol{x} \in \Lambda$ é tal que $||\boldsymbol{x}|| = \lambda(\Lambda)$, isto é, escolhemos $\boldsymbol{x}$ tal que este seja um vetor de norma mínima em Λ.

Note que para qualquer vetor do espaço em $\cup_{\boldsymbol{v}\in\Lambda\setminus\mathbf{0}}B_r(\boldsymbol{v})$, aquele que mais se aproxima da origem tem norma igual à $(\lambda - r)$. Assim, a maior bola centrada na origem que é disjunta de $\cup_{\boldsymbol{v}\in\Lambda\setminus\mathbf{0}}B_r(\boldsymbol{v})$ é a bola $B_{\lambda-r}(\mathbf{0})$. A situação descrita pode ser bem visualizada na Figura 7.

Assim, deduz-se a seguinte desigualdade:

$$Vol(E) > Vol(B_{\lambda-r}(\mathbf{0})) \tag{5}$$

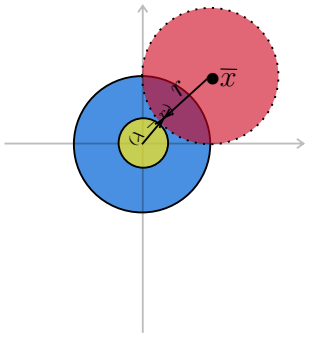


Figura 7. Ilustração do vetor de norma máxima em $\cup_{\boldsymbol{v}\in\Lambda\setminus\mathbf{0}}B_r(\boldsymbol{v})$

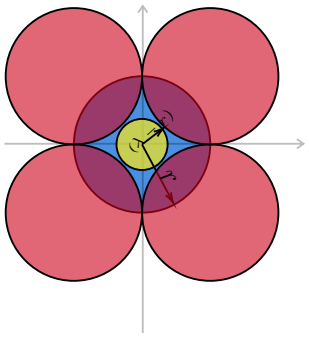


Figura 8. Ilustração da desigualdade 5

Portanto, das equações (4) e (5), segue que:

$$\text{Vol}(\cup_{\boldsymbol{v}\in\Lambda\setminus\mathbf{0}}(B_r(\mathbf{0})\cap B_r(\boldsymbol{v}))=$$
$$=\text{Vol}(B_r(\mathbf{0}))-\text{Vol}(E)<\text{Vol}(B_r(\mathbf{0}))-\text{Vol}(B_{(\lambda-r)}(\mathbf{0}))$$

Ou seja, podemos afirmar que

$$\text{Vol}(\cup_{\boldsymbol{v}\in\Lambda\setminus\mathbf{0}}(B_r(\mathbf{0})\cap B_r(\boldsymbol{v}))<\text{Vol}(B_r(\mathbf{0}))-\text{Vol}(B_{(\lambda-r)}(\mathbf{0})) \tag{6}$$

Ademais, unindo-se as equações (3) e (6), e da definição de sobreposição de bolas apresentada em 3.10, segue imediatamente que que

$$\text{Overlap}(\Lambda,r)=\frac{\text{Vol}(\cup_{\boldsymbol{v}\in\Lambda\setminus\mathbf{0}}(B_r(\mathbf{0})\cap B_r(\boldsymbol{v}))}{\text{Vol}(B_r(\mathbf{0}))}=$$
$$=\frac{\text{Vol}(\cup_{\boldsymbol{v}\in\Lambda\setminus\mathbf{0}}(B_r(\mathbf{0})\cap B_r(\boldsymbol{v}))}{(2r/\lambda)^n\Delta(\Lambda)\text{Vol}(\Lambda)}<\frac{\text{Vol}(B_r(\mathbf{0}))-\text{Vol}(B_{(\lambda-r)}(\mathbf{0}))}{(2r/\lambda)^n\Delta(\Lambda)\text{Vol}(\Lambda)}$$

Logo,

$$\text{Overlap}(\Lambda,r)<\frac{\text{Vol}(B_r(\mathbf{0}))-\text{Vol}(B_{(\lambda-r)}(\mathbf{0}))}{(2r/\lambda)^n\Delta(\Lambda)\text{Vol}(\Lambda)} \tag{7}$$

Assim, recapitulando o Teorema 3.2, temos que para $\epsilon\in(2^{o(-n)},1/3)$ e $r_\epsilon=\sqrt{\frac{n}{2\pi}}\frac{1}{2\eta_\epsilon(\Lambda^*)}$, temos que se $r\geq 2r_\epsilon(1+\delta)$, onde $\delta=\sqrt{\frac{3}{2n}ln\frac{4}{\epsilon}}$, segue que

$$\text{Overlap}(\Lambda, r) \geq \epsilon/2 \tag{8}$$

Assim, das equações (7) e (8), tem-se que

$$\frac{\epsilon}{2} \leq \text{Overlap}(\Lambda, r) < \frac{\text{Vol}(B_r(\mathbf{0})) - \text{Vol}(B_{(\lambda - r)}(\mathbf{0}))}{(2r/\lambda)^n \Delta(\Lambda)\text{Vol}(\Lambda)} \tag{9}$$

Finalmente, da equação 9, lançamos a seguinte proposição em forma de lema

Lemma 2. *Para* $\epsilon \in (2^{o(-n)}, 1/3)$ *e* $r_\epsilon = \sqrt{\frac{n}{2\pi}} \frac{1}{2\eta_\epsilon(\Lambda^*)}$, *se* $r \geq 2r_\epsilon(1+\delta)$, *onde* $\delta = \sqrt{\frac{3}{2n} ln \frac{4}{\epsilon}}$, *temos que*

$$\boxed{\frac{\epsilon}{2} < \frac{Vol(B_r(\mathbf{0})) - Vol(B_{(\lambda - r)}(\mathbf{0}))}{(2r/\lambda)^n \Delta(\Lambda) Vol(\Lambda)}} \tag{10}$$

4. Conclusão

A caracterização exata do que vem a ser um bom reticulado para criptografia ainda é um tema em aberto na área. Neste trabalho, propomos no Lemma 2 uma relação bem definida entre diversos parâmetros de um reticulado arbitrário. Surge, com isso, uma série de questões acerca da aplicação desse resultado na construção de esquemas criptográficos pós-quânticos baseados em reticulados mais seguros e eficientes, o que ainda é tema de pesquisa.

Referências

[Ajtai 1996] Ajtai, M. (1996). Generating Hard Instances of Lattice Problems (Extended Abstract). In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 99–108, New York, NY, USA. ACM.

[Chung et al. 2014] Chung, K.-M., Dadush, D., Liu, F.-H., and Peikert, C. (2014). On the lattice smoothing parameter problem.

[Cohn et al. 2017] Cohn, H., Kumar, A., Miller, S., Radchenko, D., and Viazovska, M. (2017). The sphere packing problem in dimension 24. *Annals of Mathematics*, 185(3):1017–1033.

[Diffie and Hellman 1976] Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654.

[Grover 1996] Grover, L. K. (1996). A fast quantum mechanical algorithm for database search.

[information processing standards publication 197 2001] information processing standards publication 197, F. (2001). Announcing the ADVANCED ENCRYPTION STANDARD (AES) . `https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf`.

[Jean-Sebastien Coron (Gemplus), Benne de Weger (TUE) 2007] Jean-Sebastien Coron (Gemplus), Benne de Weger (TUE) (2007). Hardness of the Main Computational Problems Used in Cryptography. *European Network of Excellence in Cryptology*.

[MENEGHETTI 2020] MENEGHETTI, F. C. C. (2020). *Reticulados: um estudo de alguns parâmetros relevantes para aplicações em criptografia.* dissertation, Universidade Estadual de Campinas.

[Micciancio and Regev 2004] Micciancio, D. and Regev, O. (2004). Worst-case to average-case reductions based on gaussian measures. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 372–381.

[NIST 2017] NIST, N. (2017). Post-Quantum Crypto Standardization. `http://csrc.nist.gov/groups/ST/post-quantum-crypto/cfp-announce-dec2016.html`.

[NIST 2020] NIST, N. (2020). Post-Quantum Crypto Standardization. `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions`.

[Rivest et al. 1978] Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126.

[Shor 1997] Shor, P. W. (1997). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.*, 26(5):1484–1509.

[Yanofsky 2007] Yanofsky, N. S. (2007). An introduction to quantum computing.

Aprendizado de Máquina para Desfragmentação Espectral em Redes Ópticas Elásticas com Multiplexação por Divisão Espacial

Silvana Trindade*

December 1, 2020

1 Introdução

Recentemente, avanços em comunicação óptica têm sido explorados para atender o crescimento exponencial do tráfego na Internet. Um destes avanços é a tecnologia de Redes Ópticas Elásticas (do inglês *Elastic Optical Netoworks*–EON), que permite a alocação de *slots* de frequência em uma granularidade muito fina — $12,5\ GHz$. Contudo, a crescente demanda por largura de banda pode não ser suportada por tecnologias que utilizam fibra com um único módulo (do inglês *Single-Mode Fiber*–SMF), chegando assim à exaustão em um curto período. Como resposta à esta limitação, a tecnologia de Multiplexação por Divisão Espacial (do inglês *Spatial Division Multiplexing*–SDM) vem sendo incorporada com EONs para aumentar a capacidade da rede (1). SDM utiliza canais espaciais e pode utilizar a tecnologia de Múltiplos Núcleos de Fibra (do inglês *Multi-Core Fiber*–MCF) onde cada núcleo é executado como um SMF.

Em redes EON-SDM, para estabelecer um caminho de luz, um conjunto de *slots* (contínuos e contíguos) precisa ser alocado em todos os enlaces que formam um caminho de luz. Na literatura este problema é denominado Roteamento, Formato de Modulação, Núcleo, e Alocação de Espectro (do inglês *Routing, Modulation Format, Core, and Spectrum Allocation*–RMCSA). O RMCSA é dividido nas seguintes etapas: busca por uma rota, escolha do formato de modulação, cálculo do número de *slots* necessários com base no formato de modulação, e alocação do espectro nos enlaces que compõe a rota escolhida.

A alocação e desalocação de caminhos de luz resultam na fragmentação espectral, que é o estado onde existem recursos espectrais disponíveis para satisfazer uma requisição que chega, mas os recursos espectrais não são contínuos ou contíguos, e portanto, não é possível estabelecer o caminho de luz. A fragmentação espectral pode aumentar o bloqueio de novas requisições, em específico quando a carga da rede aumenta (2).

A interferência sofrida entre os núcleos da fibra é denominada, do inglês, de *inter-core crosstalk*), e resulta no consumo de sinal entre núcleos adjacentes. O *crosstalk* é uma importante variável na alocação de espectro em MCFs devendo ser assim considerada. Se um valor de *crosstalk* alto ocorrer entre *slots* em núcleos adjacentes, a Qualidade de Transmissão poderá ser degradada ao ponto de bloquear *slots* adjacentes, impossibilitando que estes *slots* sejam usados para alocação de recursos, podendo resultar em uma maior fragmentação espectral.

Para tentar resolver o problema da fragmentação existem duas abordagens na literatura: proativa e reativa. Soluções proativas tentam reduzir e prevenir a ocorrência de fragmentação através da

*Instituto de Computação, UNICAMP, Campinas, SP. silvana@lrc.ic.unicamp.br

definição de critérios para seleção de rotas e *slots* de frequência, resultando no aumento da probabilidade de alocação de futuras requisições. Estas soluções podem tratar do problema da fragmentação para tráfego de baixa carga, entretanto atingem seu limite quando a rede fica mais sobrecarregada, devido à constante alocação e desalocação de caminhos de luz, que acaba aumentando o número de slots espalhados no espectro.

As soluções reativas (3; 4; 2) existentes focam na desfragmentação espectral; estas buscam reduzir a fragmentação através da realocação e/ou re-roteamento de um conjunto de caminhos de luz existentes. O re-roteamento estabelece os caminhos de luz existentes em rotas diferentes, e a desfragmentação sem re-roteamento busca somente reorganizar a alocação dos *slots* nas mesmas rotas. Soluções de desfragmentação reorganizam o espectro de tal forma que o número de *slots* contínuos e contíguos aumenta, resultando no aumento da probabilidade de novas requisições serem aceitas (5).

Grande parte das soluções existentes para desfragmentação em redes EON-SDM adaptam as soluções já existentes em redes ópticas SMFs (6; 7; 2) para MCFs. Entretanto, estas soluções apresentam problemas após a desfragmentação, já que o algoritmo de RMCSA continua aceitando requisições seguindo nenhum critério de redução de fragmentação. Sendo assim, a desfragmentação torna-se um processo constante, acarretando em um aumento nos custos da rede. Para mitigar este problema, o presente documento apresenta um solução de desfragmentação e um algoritmo de RMCSA baseado em fragmentação consciente, que trabalham de forma conjunto com o objetivo de reduzir a necessidade frequente de desfragmentação. Nesta proposta, portanto, as vantagens de abordagem proativas e reativas são unidas.

Este documento está organizado no seguinte formato. Primeiro, os algoritmos existentes de fragmentação consciente são descritos. O algoritmo CISUR e C-RMCSA são introduzidos na sequência. Finalmente, conclusões e sugestões para trabalhos futuros são apresentados.

2 Algoritmo CISUR

No algoritmo CISUR, a fragmentação espectral é monitorada, de forma que quando um conjunto de métricas atingem um certo limiar, a desfragmentação é executada. O algoritmo CISUR agrupa caminhos de luz em *clusters*, e os *clusters* são mapeados em núcleos. Assim, blocos pequenos de *slots* são reduzidos, deixando o espectro mais contínuo e contíguo, fazendo com que a probabilidade de requisições futuras serem alocadas aumente.

O CISUR utiliza o algoritmo de clusterização *K-Means*, que converge rapidamente e demonstrou ser mais apropriado em cenários com tráfego dinâmico. O coeficiente de silhueta é utilizado neste trabalho para determinar um valor ótimo de K, e é calculado usando a distância média intra-*cluster* e a distância média de *clusters* vizinhos para cada amostra. A média deste coeficiente para cada uma das amostras é computada e usada como métrica de avaliação do número de *clusters* (valor de K). As informações geradas são disponibilizadas para o algoritmo C-RMCSA após a desfragmentação. Novos caminhos de luz são atribuídos aos *clusters* baseado na similaridade com as características daqueles *clusters*.

2.1 Extração de Características

Foram realizadas simulações com 10^3 requisições, utilizando as topologias CHNNET e ARPANET com o objetivo de produzir um conjunto de dados para seleção de características relevantes para o problema da fragmentação espectral. As topologias possuem diferentes números de enlaces, nós, e grau de conectividade, o que aumenta a diversidade dos dados gerados.

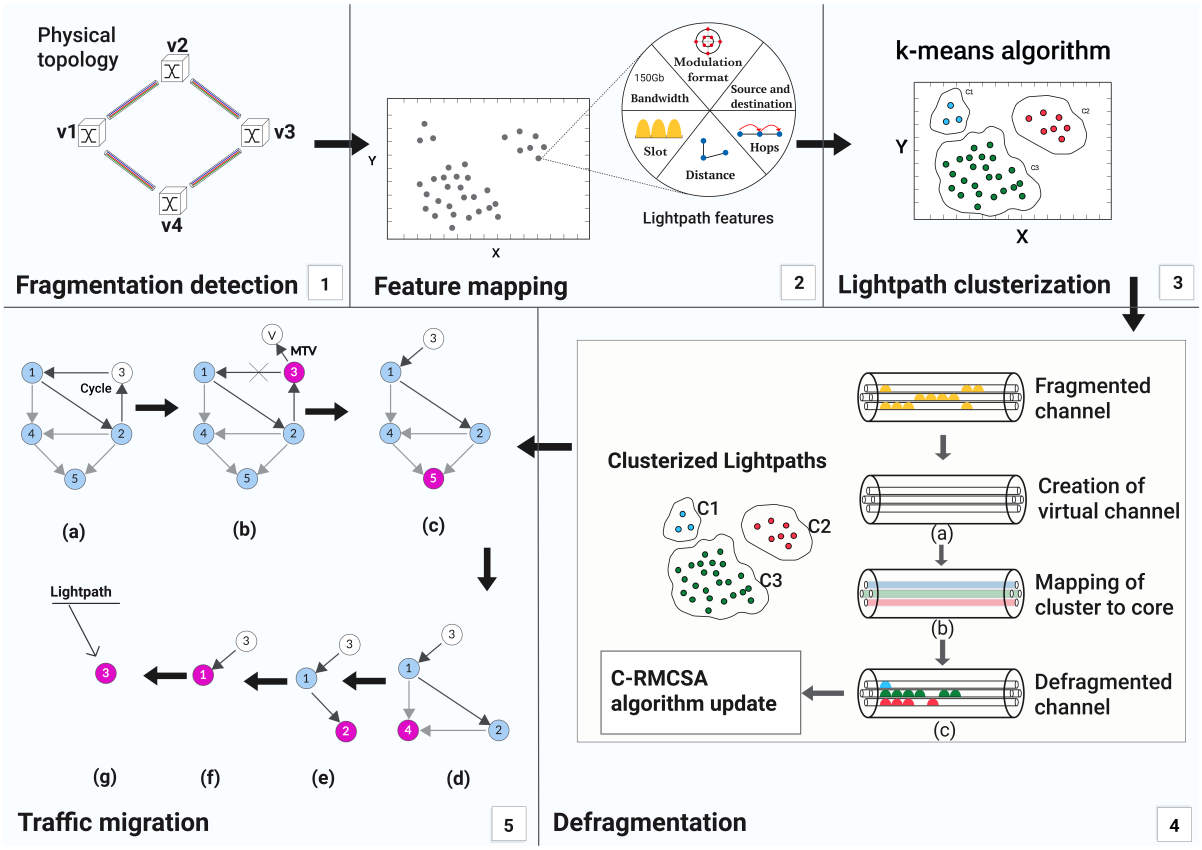


Figure 1: Representação do algoritmo CISUR.

A partir dos dados gerados, foi selecionado um conjunto de características que se correlacionam com a fragmentação espectral, ou seja, de alguma forma impactavam nas taxas de fragmentação. Os valores das características foram normalizados em um intervalo de 0 à 1 usando o método de normalização min-max.

Para definir o sub-conjunto de características a partir do conjunto descrito anteriormente, utilizou-se a técnica ANOVA (*ANalysis Of VAriance*). A técnica ANOVA compara a média do número de características e determina que médias são estatisticamente significativas distintas entre elas. Neste trabalho foi avaliado a média dos *clusters* derivada do ANOVA para cada dimensão a fim de estimar o quão distintos os *clusters* são. Como resultado, um conjunto menor de características foi selecionado, sendo elas: largura de banda, formato de modulação, tempo de chegada, tempo de vida da requisição, distância física, número de *slots*, e conjunto de enlaces que compõe a rota. Estas características foram usadas no algoritmo K-Menas para o agrupamento de caminhos de luz.

2.2 Desfragmentação

O processo de desfragmentação do algoritmo CISUR é dividido em cinco passos: identificação de fragmentação, extração de características das requisições, clusterização das requisições, rearranjo das requisições, e migração de tráfego. Estes passos estão ilustradas na Fig. 1, que descreve o procedimento da desfragmentação.

A necessidade de desfragmentação é identificada pelo monitoramento da taxa de fragmentação

da rede, da utilização do enlace, do número de requisições bloqueadas, e do número de caminhos de luz que foram desalocados. A taxa de fragmentação da rede é definida como a média entre o número de pequenos blocos de *slots* isolados e o número total de *slots* disponíveis para alocação (8). Estes blocos isolados não são contínuos no domínio espectral e nem alinhados ao longo das fibras, o que faz com que estes *slots* sejam dificilmente alocados. A utilização do enlace é observada para detectar se a capacidade da rede está próxima do seu limite pois neste estado a desfragmentação não será eficiente para disponibilizar recurso espectral, além de causar atraso por conta do processamento.

O CISUR inicialmente distribui caminhos de luz como pontos em um plano para o mapeamento dos *clusters*. O algoritmo de K-Means usa os valores das características dos caminhos de luz e executa os seguintes passos: 1. seleciona um k arbitrário para inicialização dos *clusters*, 2. computa as centroides dos k *clusters*, 3. associa cada caminho de luz com a centróide mais próxima, baseado na similaridade, e 4. verifica mudanças significativas nos *clusters*. Se estas mudanças não forem significativas, o processo de clusterização é finalizado, e as k centróides e caminhos de luz são associados à estes *clusters*.

Para determinar o valor de k, os seguintes critérios foram considerados: K deve ser menor que o número de núcleos e menor que o número de características, aumentando a eficiência da clusterização. Após a definição dos *clusters*, estes irão ser mapeados para os núcleos e os caminhos de luz serão distribuídos nestes *clusters*. O algoritmo CISUR ordena os núcleos baseado em critérios de redução de crosstalk entre núcleos adjacentes. Os *clusters* são atribuídos aos núcleos seguindo os critérios descritos em (8).

O algoritmo CISUR aloca os caminhos de luz em *clusters* de forma decrescente de acordo com a largura de banda requisitada. A política de atribuição de *slots* de frequência *First-Fit* (FF) é usada para rearranjar os caminhos de luz no espectro. A priorização dos *slots* e tamanho de requisições por largura de banda mostrou-se efetivo para diminuir a taxa de *crosstalk* e fragmentação comparada às abordagens existentes na literatura. Além disso, quando um *cluster* é mapeado para um conjunto de núcleos, ele poderá compartilhar recursos com outros *clusters*, seguindo duas políticas de alocação de espectro, a FF e Last-Fit (LF), que juntas permitem disponibilizar mais *slots* contíguos e contínuos no espectro.

O último passo da desfragmentação, quando os caminhos de luz já foram atribuídos para os *clusters*, então a interrupção de tráfego deve ser abordada usando a técnica *push-pull* (3). Caso os caminhos de luz não consigam ser atribuídos a mesma rota, então o re-roteamento destes é realiza. A interrupção de tráfego pode ocorrer neste processo, gerando impacto significativo no desempenho da rede, mas o re-roteamento é um processo raro no algoritmo CISUR.

Para tratar da interrupção de tráfego em caso de re-roteamento, foi utilizado o algoritmo *Move-To-Vacancy* (MTV) (4). Este algoritmo gera um grafo dirigido para identificar ciclos, onde os vértices são os caminhos de luz e as arestas são as dependências. Um ciclo neste grafo representa a possibilidade de interrupção de tráfego. O vértice mínimo *feedback* é identificado, e então um processo de busca por novos recursos é realizada, onde estes recursos são alocados temporariamente. Com o grafo livre de ciclos, o algoritmo então realiza a migração do restante das requisições, deixando para o final a restauração dos caminhos de luz que foram alocados temporariamente em outro local no espectro.

2.3 Algoritmo C-RMCSA

O algoritmo *Clusterization* RMCSA (C-RMCSA) determina se uma nova requisição será aceita ou não. O algoritmo é dividido em dois passos: roteamento e alocação de espectro. Quando uma requisição chega, o algoritmo C-RMCSA encontra um conjunto de caminhos mínimos candidatos usando o algoritmo de K-Caminhos Mínimos considerando um par de nós origem e destino. Na

sequência, para cada caminho candidato, o algoritmo tenta encontrar um conjunto de *slots* contíguos e contínuos para alocação.

Informações sobre ocupação espectral após a desfragmentação, tais como número de *clusters*, o mapeamento de *clusters* e núcleos, e as centróides de cada *cluster* são disponibilizadas para o algoritmo C-RMCSA. Com estas informações, o algoritmo estabelece futuras requisições de caminhos de luz, determinando em qual *cluster* cada requisição será alocada. Para fazer isso, as características das novas requisições são extraídas e normalizadas, e então usadas para determinar onde a requisição pode ser alocada no espectro. Basicamente, o algoritmo irá comparar as características da requisição que chegou com as características de cada *cluster* usando a métrica de distância de Manhattan. Para alocar a requisição dentro de um *cluster* o C-RMCSA utiliza a política de FF.

3 Resultados

A abordagem proposta neste documento foi comparada a dois algoritmos, o *First-Core First-Fit* (FCFF) (9) que é um algoritmo proativo, e o algoritmo *Defragmentation* (DF) (4) que é reativo. Para termos uma comparação justa com o CISUR, adicionamos o paradigma de formato de modulação em ambos algoritmos.

Todos algoritmos utilizados neste documento foram implementados no simulador FlexGrid, que está disponível em (10). Cada simulação recebeu 100.000 requisições, e a carga (Erlang) foi aumentada em etapas de 25 Erlangs para cada simulação de 0 à 500. O intervalo de confiança considerado foi de 95% usando vinte amostras em cada etapa. A largura de banda das requisições foi gerada uniformemente seguindo uma distribuição entre 25 e 400 *Gb*.

Para as simulações, a topologia NSFNET foi utilizada (8). Esta topologia possui 14 nós e 21 enlaces de fibra. A arquitetura de nó considerada neste trabalho está descrita em (11). Cada enlace de fibra é bidirecional e contem 7 núcleos, cada um com 320 *slots* de frequência e estes com espaçamento de 12,5 *Ghz*.

A taxa de bloqueio por largura de banda (do inglês *Bandwidth Blocking Ratio*–BBR) foi usada para avaliar o desempenho dos algoritmos. A taxa de fragmentação gerada pelo CISUR é 25% menor que a produzida pelos outros algoritmos. Tais resultados evidenciam a vantagem do uso de desfragmentação orientada a características. O algoritmo DF (4) executa a desfragmentação em média 330 vezes por simulação, enquanto o algoritmo CISUR executa no pior caso 100 vezes. Além disso, usando o CISUR, cada desfragmentação é realizada com 10 à 25% dos caminhos de luz, enquanto o DF executa o mesmo processo com um número fixo de caminhos de luz. Se tivéssemos 100,000 requisições, cada requisição com tempo de vida de 1 segundo, o algoritmo CISUR seria executado em média 100 a 250 vezes e a ocorrência com taxa de interrupção de tráfego seria de 1%. Além disso, o algoritmo CISUR não precisa de *transponders* adicionais para reconfigurar os caminhos de luz, aem adição de custo monetário.

A taxa de fragmentação produzida pelo algoritmo CISUR ocasionou em uma taxa baixa de BBR, sendo assim, mais caminhos de luz puderam ser aceitos. Os resultados mostraram que o algoritmo CISUR consegue diminuir os valores de BBR, inciando o bloqueio de requisições somente a partir de 200 erlangs, enquanto o algoritmo DF inicia o bloqueio com carga de 100 erlangs. Tais resultados foram possíveis devido ao uso de clusterização de caminhos de luz, mostrando que as características selecionas para a clusterização descrever efetivamente o cenário do problema da fragmentação.

4 Conclusões

Neste documento foi apresentado uma abordagem de desfragmentação baseada em aprendizado de máquina não-supervisionado com um algoritmo de RMCSA que reduz a fragmentação espectral. Estes algoritmos juntos demonstraram ser eficientes para o problema da fragmentação espectral, diminuindo a probabilidade de bloqueio de caminhos de luz.

4.1 Publicação

Machine Learning for Spectrum Defragmentation in Space-Division Multiplexing Elastic Optical Networks. Trindade, Silvana e Fonseca, Nelson L. S. da. IEEE Network, 2020.

References

[1] D. Richardson, J. Fini, and L. Nelson, "Space-division multiplexing in optical fibres," *Nature Photonics*, vol. 7, no. 5, p. 354, 2013.

[2] R. Luo, N. Hua, X. Zheng, and B. Zhou, "Fast parallel lightpath re-optimization for space-division multiplexing optical networks based on time synchronization," *Journal of Optical Communications and Networking*, vol. 10, no. 1, pp. A8–A19, 2018.

[3] F. Cugini, F. Paolucci, G. Meloni, G. Berrettini, M. Secondini, F. Fresi, N. Sambo, L. Poti, and P. Castoldi, "Push-pull defragmentation without traffic disruption in flexible grid optical networks," *Journal of Lightwave Technology*, vol. 31, no. 1, pp. 125–133, 2012.

[4] M. Zhang, W. Shi, L. Gong, W. Lu, and Z. Zhu, "Bandwidth defragmentation in dynamic elastic optical networks with minimum traffic disruptions," in *Communications (ICC), 2013 IEEE International Conference on*, pp. 3894–3898, IEEE, 2013.

[5] Y. Zhao, L. Hu, R. Zhu, X. Yu, X. Wang, and J. Zhang, "Crosstalk-aware spectrum defragmentation based on spectrum compactness in space division multiplexing enabled elastic optical networks with multicore fiber," *IEEE Access*, vol. 6, pp. 15346–15355, 2018.

[6] Y. Zhao and J. Zhang, "Crosstalk-aware cross-core virtual concatenation in spatial division multiplexing elastic optical networks," *Electronics Letters*, vol. 52, no. 20, pp. 1701–1703, 2016.

[7] H. Tode and Y. Hirota, "Routing, spectrum, and core and/or mode assignment on space-division multiplexing optical networks," *Journal of Optical Communications and Networking*, vol. 9, no. 1, pp. A99–A113, 2017.

[8] S. Trindade and N. L. S. da Fonseca, "Proactive fragmentation-aware routing, modulation format, core, and spectrum allocation in eon-sdm," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pp. 1–6, May 2019.

[9] M. Klinkowski, P. Lechowicz, and K. Walkowiak, "Survey of resource allocation schemes and algorithms in spectrally-spatially flexible optical networking," *Optical Switching and Networking*, 2017.

[10] S. Trindade, "Flexgridsim: Flexible grid optical network simulator," 2018.

[11] A. Muhammad, G. Zervas, and R. Forchheimer, "Resource allocation for space-division multiplexing: optical white box versus optical black box networking," *Journal of Lightwave Technology*, vol. 33, no. 23, pp. 4928–4941, 2015.

Counting Sorting Scenarios for the Rank Distance

João Paulo Pereira Zanetti[1] **, Lucas Peres Oliveira**[1] **,**
Leonid Chindelevitch[2] **, João Meidanis**[1]

[1] Institute of Computing – University of Campinas (Unicamp)
Av. Albert Einstein, 1251 – Campinas – SP – Brazil

[2]School of Public Health – Imperial College London
London SW7 2AZ, U.K.

joao.zanetti@ic.unicamp.br, l265193@dac.unicamp.br

lchindel@ic.ac.uk, meidanis@ic.unicamp.br

Abstract. *An important problem in genome comparison is to find a sequence of basic operations that transforms one genome into another, called an optimal sorting scenario. However, there is usually a large number of such scenarios, and a naïve algorithm is very likely to be biased towards a specific type of scenario, impairing its usefulness. In this paper, we study the solution space of the this problem and show how to count the number of optimal sorting scenarios between any two given genomes, under the rank distance.*

Resumo. *Um problema importante em comparação de genomas é encontrar uma sequência de operações básicas que transforma um genoma em outro, chamada de cenário de ordenação ótimo. Entretanto, geralmente existe um número enorme de tais cenários, e é muito provável que um algoritmo ingênuo seja enviesado em direção a um tipo específico de cenário, prejudicando sua utilidade. Neste trabalho, estudamos o espaço de soluções desse problema e mostramos como contar o número de cenários de ordenação ótimos entre dois genomas quaisquer, sob a distância de posto.*

1. Introduction

At a high level of abstraction, genomes can be represented as a list of blocks (or syntenic regions), and their evolution can be modeled by large-scale mutation events we call *genome rearrangements*. Different genome rearrangement models define different events and costs (weights) for them, and their most basic application is to determine the lowest cost required to transform one genome into another. This is the *genome distance problem*. A related problem is to find sequences of rearrangement operations with minimum cost. We call this the *genome sorting problem*. However, a single arbitrary sequence of events among many optimal ones is hardly representative of the evolutionary process, especially considering that sorting algorithms might be biased towards certain kinds of sorting sequences. On the other hand, listing all possible optimal scenarios is not practical, because their number is simply too large.

A first step towards exploring the solution space of the genome sorting problem as a whole is to count the number of optimal sorting scenarios between two given genomes. This can reveal helpful patterns in the optimal solutions and suggest strategies for real-world applications. Enumeration studies have been made for several distance models,

including a simple model involving inversion operations only, the *Double-Cut-and-Join* (DCJ) model, and the breakpoint-related *Single-Cut-or-Join* (SCJ) model. Here, we produced an analogous set of results for the rank distance model [Zanetti et al. 2019].

2. Genomes and the Rank Distance

We define genomes as a collection of *chromosomes*, each of them a linear or circular sequence of *genes*. A gene is a linear segment with two *extremities*: a *tail* and a *head*. When two genes appear consecutively in a chromosome, we indicate this fact by linking their closest extremities with an *adjacency*. Adjacencies are thus unordered pairs of extremities. An extremity may not be involved in more than one adjacency. If an extremity is not in any adjacency, it is called a *free end*. A genome is completely characterized by its adjacencies and free ends. Figure 1 shows an example of a genome with three genes.

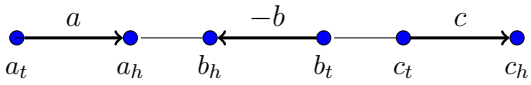


Figure 1. Genome with one linear chromosome, and adjacencies $\{a_h, b_h\}$ and $\{b_t, c_t\}$. The extremities a_t and c_h are free ends.

The rank distance, although originally defined using matrices, can be defined as the weight of an optimal sequence of operations transforming A into B. We allow three basic operations that are sufficient and necessary to transform any genome into another: a *cut* of an adjacency $\{x, y\} \rightarrow \{x\}\{y\}$, a *join* of two free ends $\{x\}\{y\} \rightarrow \{x, y\}$, and a *double swap* of two adjacencies into two new ones using the same four extremities, for example $\{x, y\}\{a, b\} \rightarrow \{x, a\}\{y, b\}$. Any other rearrangement operation can be decomposed as a sum of these three kinds of operations. In the context of this work, the most important information about the basic operations is that cuts and joins have weight 1, while double swaps have weight 2.

3. Breakpoint Graph

Genomes, as we describe here, are matchings over the set of gene extremities. We can graphically represent two genomes A and B as matchings, using one color (or line style) for A and another for B, as shown in Figure 2.

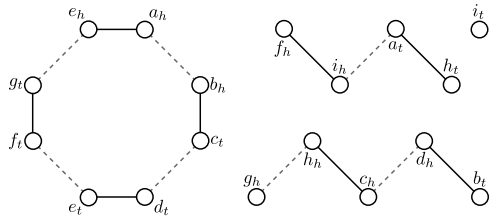


Figure 2. Example breakpoint graph $BG(A, B)$. Genome A has adjacencies $a_h b_h$, $c_t d_t$, $e_t f_t$, $e_h g_t$, $a_t i_h$, $g_h h_h$, and $c_h d_h$, drawn as dashed edges, and free ends b_t, f_h, h_t, and i_t. Genome B has adjacencies $a_h e_h$, $b_h c_t$, $d_t e_t$, $f_t g_t$, $f_h i_h$, $a_t h_t$, $c_h h_h$, and $b_t d_h$, drawn as solid edges, and free ends g_h and i_t.

Given two genomes A and B over the same set of extremities, we build the two-genome breakpoint graph $BG(A, B)$ as follows: its vertices are the extremities of the

genomes, and its edges connect pairs of extremities that are adjacent in a genome, being dashed edges for adjacencies in A and solid edges for adjacencies in B. The process of *sorting* consists of the application of basic operations to genome A until we get genome B.

Every node in the breakpoint graph has degree 0, 1, or 2. Because of this, the connected components of the graph are disjoint cycles and paths. We call cycles with four or more edges *big cycles* and paths with at least one edge *big paths*. These components need to be worked on in order to sort genome A into B. The following formula computes the rank distance using the breakpoint graph between genomes A and B:

$$d(A, B) = 2n - 2c - p$$

where n is the number of genes, and c and p are respectively the number of cycles and paths in $BG(A, B)$ [Zanetti et al. 2019, Theorem 1].

4. Counting the Number of Scenarios

As a first step to count the number of sorting scenarios, we proved a useful property, namely that no optimal rearrangement recombines the components of the breakpoint graph [Zanetti et al. 2019, Lemma 3]. In other words, we showed that no optimal operation acts on the extremities of more than one component at the same time.

With this result, we concluded that it is possible to count the optimal scenarios for each component in the breakpoint graph independently. Let s_1 and s_2 be scenarios for the components C_1 and C_2 respectively, with respective lengths ℓ_1 and ℓ_2. Then, the number of scenarios resulting in the combination of s_1 and s_2 is the number of sequences that have both as subsequences, and this is the shuffle product of s_1 and s_2, whose size is given by the binomial coefficient $\binom{\ell_1+\ell_2}{\ell_1,\ell_2} = \frac{(\ell_1+\ell_2)!}{\ell_1!\ell_2!}$. The next subsections describe the counting of optimal scenarios for cycles and paths separately.

4.1. Cycles

Given a cycle in $BG(A, B)$, the only optimal operation that can be applied to it is a double swap that splits the cycle into two smaller ones, as illustrated in Figure 3. In this case, the sorting moves are equivalent to the ones for the DCJ distance [Yancopoulos et al. 2005]. Thus, we could use the formula for DCJ to compute the number $S_c(2\ell + 2)$ of scenarios to solve a cycle of length $2\ell + 2$ [Braga and Stoye 2010, Theorem 3]:

$$S_c(2\ell + 2) = (\ell + 1)^{(\ell-1)}$$

Each of these $S_c(2\ell + 2)$ scenarios has length ℓ, and total weight 2ℓ. When A and B have the same free ends, the only components in $BG(A, B)$ are big cycles, and we can compute the total number S_{ct} of optimal sorting scenarios from A to B [Braga and Stoye 2010, Theorem 4] as follows:

$$S_{ct} = \frac{(\ell_1 + \ell_2 + \ldots + \ell_p)!}{\ell_1!\ell_2! \ldots \ell_p!} \prod_{i=1}^{p} (\ell_i + 1)^{\ell_i - 1},$$

where p is the number of big cycles in $BG(A, B)$.

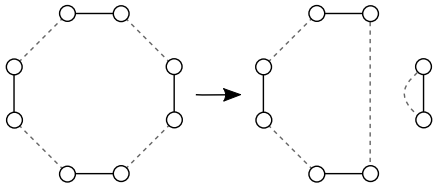


Figure 3. One example of optimal rearrangement for a cycle. A double swap splits the cycle into two smaller ones. Here, an 8-cycle is decomposed into a 6-cycle, which will require two more double swaps to complete the sorting, and a 2-cycle, already sorted.

4.2. Paths

For paths, the computation is less straightforward, as cuts and joins have weight 1, while double swaps have weight 2. As a result, scenarios have variable length, and information on the length of the sub-solutions is necessary for the shuffling. Therefore, we need a recurrence with two variables: the length of the path, and the length of the scenario.

For a path, three optimal operations are possible. First, a cut of any dashed edge, resulting in two smaller paths that are solved separately. A second option is to execute a double swap on any two dashed edges, resulting in a cycle and a path. Here again, both new components have independent scenarios that are then shuffled. The last option is to join the ends of a path, if both ends are incident to solid edges. This leads to a single cycle, that we already know how to process. These possibilities are illustrated in Figure 4.

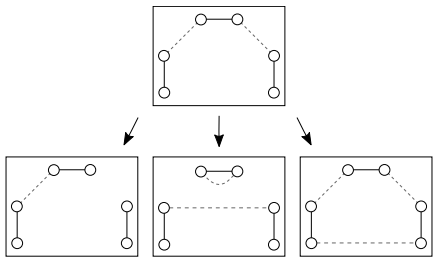


Figure 4. Examples of the three options of optimal moves from a solid path. The first one, from the top, is a cut, splitting the 5-path into two paths with 3 and 1 edges, respectively. The second is a double swap extracting a cycle from the path. The last option, only possible in this kind of path, is to join the ends, forming a cycle.

Given a path of length k, the sizes of the optimal scenarios fall within a limited range. The longest scenarios for a k-path are the ones with only cuts and joins, making up a total of k operations. Since the double swaps have twice the weight of a cut or join, scenarios with more double swaps are shorter. As for shortest scenarios for a k-path, at least $\frac{k+1}{2}$ operations are required [Zanetti et al. 2019].

With this set of optimal operations, and the range for the length of a scenario, we arrived at a recurrence relation for the number $S_p(k, \ell)$ of optimal sorting scenarios of length ℓ for a path with k edges.

$$S_p(0,0) = 1 \tag{1}$$

$$S_p(k,\ell) = \sum_{i=0}^{\lceil \frac{k}{2} \rceil - 1} \sum_{t=i}^{2i} S_p(2i,t) S_p(k-2i-1, \ell-t-1) \binom{\ell-1}{t} + \sum_{z=1}^{\lceil \frac{k}{2} \rceil - 1} \left(\left\lceil \frac{k}{2} \right\rceil - z \right) z^{(z-2)} S_p(k-2z, \ell-z) \binom{\ell-1}{\ell-z} \tag{2}$$

Having determined $S_c(k)$ and $S_p(k,\ell)$, we can count the total number of scenarios between any two genomes. The original paper summarizes this result in a single theorem [Zanetti et al. 2019, Theorem 4].

5. Experiments

We implemented our formulas and tested our method for counting the scenarios between pairs of a number of genomes from the literature. The code to run our experiments was implemented in Python, and executed on a virtual machine using a single 2.3GHz processor core and 2GB of memory. We used four data sets from different sources. The first and simplest data set consists of two pairs of circular mitochondrial DNA, comparing *Brassica campestris* against *B. oleracea* and *B. napus* [Palmer and Herbon 1988]. Both instances have 5 synteny blocks. The second data set is the human and mouse X chromosome [Pevzner and Tesler 2003]. This pair of linear, single-chromosome, inputs has 11 synteny blocks. The third data set is composed of 13 chloroplast genomes, of which 12 are from the Campanulaceae family, and 1 from tobacco as an outgroup, with 105 synteny blocks [Cosner et al. 2004]. These created 78 pairs of inputs to the sorting problem. Finally, the fourth and largest data set consists of 20 instances comparing the human genome against the genome from other animals, namely, 18 Eutherian mammals, plus opossum and chicken as outgroups [Kim et al. 2017]. These pairs have between 101 and 621 synteny blocks.

6. Results and Discussion

The two *Brassica* instances have the same rank distance of 6 (in this case three double swaps), and the same result: 9 scenarios. For the pair of X chromosomes, with a rank distance of 14, we get 237440 scenarios. With the chloroplast genome pairs, we get varying results. Some pairs, like *Trachelium* and *Campanula* are only one double swap apart, and therefore have only one optimal scenario. The pair of genomes that are farthest apart is *Merciera* and *Platycodon*, with a distance of 48. They have 1.4×10^{32} optimal scenarios.

With the Eutherian data set, due to the large number of scenarios between distantly related species, we developed a scheme that is always guaranteed to use a specified amount of memory. This scheme produced the exact result for 5 out of the 20 instances, namely, the primates (chimpanzee, marmoset, orangutan, and rhesus) and the horse. For the other instances, we obtained upper and lower bounds on the number of optimal scenarios, as described in the full paper [Zanetti et al. 2019, Section 5.3]. In Table 1 we list the number of scenarios for the 5 instances with exact results.

Table 1. Rank distance, number of scenarios between the human genome and the genomes of 5 Eutherian animals, listed in order of distance.

Genome	d	Scenarios
Chimpanzee	27	6.54×10^{11}
Orangutan	53	6.03×10^{38}
Rhesus	150	1.21×10^{138}
Marmoset	204	3.99×10^{250}
Horse	225	1.63×10^{135}

7. Conclusion

In this paper we opened the doors for the exploration of the solution space of the genome sorting problem under the rank distance. We demonstrated that there is no recombination between components of the breakpoint graph in any optimal rank sorting scenario. We then gave formulas for the number of optimal sorting scenarios for each component. Finally, the formulas were implemented to test our method for counting the scenarios between pairs of real genomes from the literature. Another aspect in the study of the solution space is the counting of intermediate genomes, which can be found in the original paper. Sampling intermediate genomes is the next step in the study of the solution space and can be very helpful in future applications

References

Braga, M. D. and Stoye, J. (2010). The solution space of sorting by DCJ. *Journal of Computational Biology*, 17(9):1145–1165.

Cosner, M. E., Raubeson, L. A., and Jansen, R. K. (2004). Chloroplast DNA rearrangements in Campanulaceae: phylogenetic utility of highly rearranged genomes. *BMC Evolutionary Biology*, 4(1):1–17.

Kim, J., Farré, M., Auvil, L., Capitanu, B., Larkin, D. M., Ma, J., and Lewin, H. A. (2017). Reconstruction and evolutionary history of eutherian chromosomes. *Proceedings of the National Academy of Sciences*, 114(27):E5379–E5388.

Palmer, J. D. and Herbon, L. A. (1988). Plant mitochondrial DNA evolved rapidly in structure, but slowly in sequence. *Journal of Molecular Evolution*, 28(1):87–97.

Pevzner, P. and Tesler, G. (2003). Genome rearrangements in mammalian evolution: lessons from human and mouse genomes. *Genome Research*, 13(1):37–45.

Yancopoulos, S., Attie, O., and Friedberg, R. (2005). Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346.

Zanetti, J. P. P., Chindelevitch, L., and Meidanis, J. (2019). Counting sorting scenarios and intermediate genomes for the rank distance. In Holmes, I., Martín-Vide, C., and Vega-Rodríguez, M. A., editors, *Algorithms for Computational Biology*, pages 137–151, Cham. Springer International Publishing.