30-Setembro-2021

Proposta Edital CNPq/MCTI/FNDCT Nº 18/2021
Faixa B (Grupos Consolidados)

Título: *Teoria e Aplicações de Paralelismo, Otimização, Combinatória e Algoritmos (TAPIOCA)*
Coordenador: *Rudini Menezes Sampaio*
`http://www.lia.ufc.br/~rudini`, *rudini@dc.ufc.br*
Instituição: *Universidade Federal do Ceará*
Programa de Mestrado e Doutorado em Ciência da Computação
Campus do Pici, Bloco 910
60440-554 Fortaleza, CE

Início: *1º semestre de 2022*
Duração: *36 meses*
Valor: *R$274.850,00 - Faixa B*

Conteúdo

1 Grupo de Pesquisa

A base da equipe executora do presente projeto de pesquisa é constituída por membros do grupo de pesquisa ParGO - Paralelismo, Grafos e Otimização (`http://pargo.ufc.br`), sediado na Universidade Federal do Ceará (UFC), cujas características principais descrevemos nesta seção, para que se possa constatar que se trata de um grupo bem consolidado, jovem e de excelência, que vem realizando pesquisas de alta relevância em sua especialidade, capaz de funcionar como fonte geradora e transformadora de conhecimento, contribuindo assim para o desenvolvimento científico e tecnológico do país. A equipe conta também com pesquisadores da Universidade da Integração Internacional da Lusofonia Afro-Brasileira (UNILAB), formados dentro do grupo, que hoje atuam de forma independente em sua instituição, porém mantendo colaboração com o ParGO.

1.1 Perfil do Grupo de Pesquisa

O ParGO é um grupo de pesquisa formado por pesquisadores que trabalham em torno de linhas de pesquisa relacionadas e complementares, notadamente em grafos, algoritmos, otimização combinatória e paralelismo, com enfoque no estudo teórico de problemas e desenvolvimento de métodos de solução para eles, e em largo espectro, que vai desde o estudo da complexidade de problemas à busca de modelagem e procedimentos computacionais para a resolução dos mesmos em quantidades massivas de dados.

O grupo integra o Diretório de Grupos de Pesquisa do CNPq (`http://dgp.cnpq.br/dgp/espelhogrupo/21115`). Ao longo dos anos, vem mantendo diversas colaborações nacionais e internacionais, participando de projetos de pesquisa multi-institucionais, que se refletem na produção do grupo (ver `http://pargo.ufc.br`). Atualmente, tem forte cooperação com a Alemanha (através de projeto CAPES/DAAD/Probral em parceria UFC/UFRGS/TU Chemnitz) e com Chile e França (através de projeto CAPES/STIC-AmSud em parceria UFC/INRIA Sophia Antipolis).

Objetivos O objetivo geral do grupo é o estudo de modelos matemáticos e métodos (algoritmos) com vistas à resolução automática (ou automatizada) de problemas, incluindo-se o estudo da complexidade desses problemas, bem como a proposição de ferramentas para a resolução distribuída ou paralela dos mesmos, em quantidade massiva de dados. Nesse sentido, destacam-se algumas atividades específicas, como:

- Estudar problemas combinatórios teóricos (como coloração de grafos, conectividade em grafos, difusão de informações em redes sociais, escalonamento de tarefas, alocação de recursos, e suas variações, entre outros), visando à obtenção de modelos discretos e contínuos, com aplicações em problemas práticos de telecomunicações e de gerenciamento da execução de processos.
- Pesquisar teoria e técnicas de desenvolvimento de algoritmos para problemas complexos de otimização combinatória, produzindo algoritmos exatos, probabilísticos, aproximativos ou heurísticas, usando técnicas de decomposição, particionamento e aproximação dos problemas, entre outras.
- Pesquisar técnicas de desenvolvimento e de implementação de algoritmos concorrentes, paralelos e distribuídos para resolução eficaz de problemas de grande porte, ou seja, problemas que lidam com grandes quantidades de dados, sobretudo os de otimização combinatória.

Áreas de Atuação O grupo atua de forma integrada nas três áreas que definem seu nome, desenvolvendo pesquisa e formando recursos humanos em temas ligados a:

- Heurísticas;
- Teoria dos Grafos;
- Teoria da Computação;
- Otimização Combinatória;
- Programação Concorrente, Paralela e Distribuída;
- Programação Matemática (notadamente linear e inteira).

Atividades Além da pesquisa em si nos temas acima citados, as atividades do grupo incluem:

- formação de recursos humanos para a pesquisa, na graduação e pós-graduação, por meio de orientações de trabalhos de iniciação científica, dissertações de mestrado e teses de doutorado.
- divulgação científica, através de ciclos de seminários, onde os membros do grupos e convidados apresentam os resultados de suas pesquisas e novos problemas.
- intercâmbio científico, através da colaboração estreita e do desenvolvimento de projetos financiados com grupos nacionais e internacionais de excelente reputação na sua área de atuação.

1.2 Histórico e realizações

O ParGO foi criado em 1999 e, desde então, vem crescendo e se afirmando como um grupo produtivo e comprometido com a qualidade dos resultados que gera, tanto em suas atividades de pesquisa quanto de formação de pessoal. O grupo tem experimentado um grande crescimento quantitativo e qualitativo nos últimos anos, destacando-se nos cenários local, nacional e internacional em sua área de atuação.

Por exemplo, as instituições do Estado do Ceará possuem atualmente 19 bolsistas de produtividade em pesquisa do CNPq na área de Ciência da Computação (CA-CC), apenas 3 no nível 1, sendo que 14 deles são professores da UFC. O ParGO reúne 8 bolsistas de produtividade (7 deles neste projeto), sendo 7 na área de Ciência da Computação (CA-CC) e 1 em Pesquisa Operacional (CA-EP: Engenharia de Produção). Ademais, 2 dos 3 bolsistas de produtividade CA-CC do CNPq com nível 1 no Ceará são do ParGO.

Outra evidência de destaque do grupo nacionalmente está no fato de que a CE-ACO da SBC (Comissão Especial em Algoritmos, Combinatória e Otimização) possui desde a sua criação em 2015 pelo menos dois membros do ParGO, sendo o Prof. Manoel Campêlo (PQ-2) o atual coordenador no triênio 2021-2023. Membros do ParGO estiveram na organização de todos os eventos da CE-ACO, como as seis edições do ETC-CSBC (Encontro de Teoria da Computação da SBC).

O coordenador deste projeto, Prof. Rudini Sampaio, é um dos 2 bolsistas de produtividade nível PQ-1D do ParGO e também é membro do comitê gestor da CE-ACO da SBC, desde a sua fundação. Juntamente com Prof. Manoel Campêlo e os outros 2 pesquisadores da equipe do projeto com mais de 10 anos de doutoramento, Prof. Heron Carvalho e Prof. Rafael Andrade (PQ-2), têm contribuído largamente com o Programa Pós-Graduação em Ciência da Computação (MDCC) da UFC, como professores e membros da coordenação, já tendo orientado juntos mais de 50 dissertações de mestrado e teses de doutorado no programa. Além destes 4, os outros 8 pesquisadores da equipe principal do projeto (6 deles do ParGO) obtiveram o título de doutor nos últimos 10 anos e já contam com destacada produção científica. Maiores destalhes serão vistos na próxima seção.

Esses números e fatos comprovam a excelência, jovialidade e vitalidade do ParGO e de seus pesquisadores envolvidos neste projeto. Ademais, comprovam que a área de inserção deste projeto, dentro da Ciência da Computação, tem relevância reconhecida pelo CNPq e seus comitês assessores. Ressalte-se que a integração e dinâmica do grupo, além de proporcionadas pela semelhança e complementaridade de seus temas de trabalho, são resultados em grande parte da convivência diária de seus participantes no laboratório de pesquisa, reforçada pelo ciclo de seminários.

A dedicação desse grupo tem sido reconhecida por órgãos de fomento à pesquisa científica, como CNPq, CAPES, FINEP, FUNCAP, que têm financiado os projetos individuais e coletivos de que participamos, bem como por renomados pesquisadores de outras instituições, nacionais e internacionais, com os quais desenvolvemos projetos de pesquisa e formação conjuntos. Entre os projetos multi-institucionais, podemos citar nossa participação em projetos EUROPAID/ALPHA, CAPES/DAAD/Probral, CAPES/STIC-AmSud, CAPES/Cofecub, CNPq/PRONEX, CNPq/PADCT, CNPq/Prosul, CNPq/INCT.

O reconhecimento do trabalho do ParGO também nos tem possibilitado organizar excelentes eventos científicos, a saber:

- CIMPA-1999 (Internacional CIMPA School on Advanced Parallel Techniques and Applications, Natal)
- CIMPA-2001 (International CIMPA School on Combinatorics and Algorithms, Fortaleza)

- LAGOS-2001 (I Latin American Graphs, Algorithms and Optimization Symposium, Fortaleza)
- I Workshop em Combinatória e Telecomunicações (Fortaleza, 2005);
- SBLP-2008 (XII Simpósio Brasileiro de Linguagens de Programação), Fortaleza, 2008;
- GRASTA-2008 (II Workshop on Graph Searching, Theory and Applications, Fortaleza).
- ELAVIO-2010 (XV Escuela Latino-americana de Investigacion Operativa)
- GCO-2012 (I Workshop Franco-brésilien de Graphes et Optimisation Combinatoire, Fortaleza) `http://www-sop.inria.fr/mascotte/Events/GCO2012`
- LAGOS-2015 (VIII Latin-American Algorithms, Graphs and Optimization Symposium, Fortaleza) `http://www.lia.ufc.br/lagos2015`
- GCO-2016 (II Workshop Franco-brésilien de Graphes et Optimisation Combinatoire, Fortaleza) `http://www.lia.ufc.br/GCO2016`
- ParGO 20+50 (Workshop de Comemoração de 20 anos de Fundação do ParGO, 2019)
- ForWorC-2019 (I Fortaleza Workshop em Combinatória) `http://www.mat.ufc.br/portal/ptbr/events/229-forworc2019`

O envolvimento e a representatividade dos pesquisadores do ParGO no cenário científico nacional e nas suas parcerias internacionais refletem-se na presença de membros do ParGO na organização de todos esses eventos. Em particular, as escolas e workshops propiciaram uma oportunidade ímpar, tanto para os membros do ParGO, quanto para outros grupos que trabalham na mesma área ou em áreas correlatas, com vistas à interação com pesquisadores estrangeiros de renome internacional, tais como Abdel Lisser (Université Paris Sud), Adrian Bondy (Université Lyon 1), Bruce Reed (McGill University), Gérard Cornuéjols (Carnegie Mellon University), Gérard Plateau (Université Paris 13), László Lovász (Eötvös Loránd University, Budapest), Maria Chudnovsky (Princenton University), Michel Goemans (MIT), Paul Seymour (Princeton University), Vasek Chvátal (Concordia University), William Tutte (University of Waterloo), dentre outros, que ministraram cursos, apresentaram seus resultados de pesquisa mais recentes ou discutiram problemas em aberto.

Todas estas atividades e realizações têm servido certamente para a consolidação do ParGO. Em particular, elas alimentam uma prática recorrente no grupo, que o mantém renovado e trabalhando na fronteira do conhecimento das áreas onde atua, que é o incentivo constante à participação em conferências e workshops, à interação com outros pesquisadores e à realização de estágios em centros de pesquisa avançados. O trabalho do grupo também contribui para o crescimento da Ciência da Computação no país, na medida em que gera conhecimentos relevantes para a área e forma de maneira cuidadosa novos pesquisadores da Computação, que trabalham em várias universidades do Brasil e do exterior. A título de exemplo, considerando apenas o estado do Ceará, doutores formados pelo ParGO atuam em três departamentos da UFC em Fortaleza, bem como dos Campi Avançados da UFC em Quixadá, Crateús, Russas e Sobral, além da UFCA (Universidade Federal do Cariri), da UVA (Universidade Estadual Vale do Acaraú), da UECE (Universidade Estadual do Ceará), da UNIFOR (Universidade de Fortaleza), da UNILAB (Universidade da Integração Internacional da Lusofonia Afro-Brasileira).

2 Pesquisadores do Projeto

Os pesquisadores deste projeto cobrem, de maneira complementar e em permanente interação, as diferentes especialidades integrantes das suas áreas de atuação. A diversidade de formação acadêmica (em universidades nacionais e internacionais de prestígio em suas respectivas áreas) e linhas de atuação da equipe imprimem à mesma um aspecto multidisciplinar onde complementaridade das especialidades é coerente com o perfil científico do grupo.

2.1 Pesquisadores Principais

A equipe principal deste projeto é composta por 12 (doze) pesquisadores de duas instituições, a saber a UFC-Fortaleza e a UNILAB, dentre os quais 7 (sete) são bolsistas de produtividade do CNPq: 1 com nível PQ-1D no CA-CC, 5 com nível PQ-2 no CA-CC e 1 com nível PQ-2 no CA-EP, onde CA-CC se refere ao Comitê de Assessoramento em Ciência da Computação e CA-EP se refere ao Comitê de Assessoramento em Engenharia de Produção do CNPq. Os cinco demais pesquisadores brasileiros principais, quase todos com menos de 10 anos de titulação, desenvolvem pesquisa em colaboração com outros membros do grupo e, como poderá ser visto posteriormente, possuem os requisitos exigidos pelo CNPq para enquadramento como bolsista de produtividade em pesquisa nível PQ-2. Além disso, conta com três pesquisadores de universidades estrangeiras, que já mantêm estreita colaboração com os demais membros. No que segue, descrevemos brevemente os perfis dos pesquisadores de cada universidade, listando primeiramente os bolsistas de produtividade do CNPq.

UFC (Universidade Federal do Ceará) – Campus Fortaleza

Rudini Menezes Sampaio (Coordenador)

- Doutor em Ciência da Computação, IME-USP (Universidade de São Paulo), Brasil, 2008.
- Bolsista de Produtividade em Pesquisa do CNPq - Nível 1D - CA CC - Ciência da Computação.
- Afiliado ao Departamento de Computação, UFC.
- Membro do Programa de Pós-Graduação em Ciência da Computação (MDCC) da UFC.
- Membro da CE-ACO da SBC (Comissão Especial em Algoritmos, Combinatória e Otimização)

Ana Shirley Ferreira da Silva

- Doutora em Mathématiques et Informatique, Université Grenoble Alpes, UGA, França, 2010.
- Pós-doutorado, Centrum Wiskunde & Informatica, Holanda, 2015.
- Bolsista de Produtividade em Pesquisa do CNPq - Nível 2 - CA CC - Ciência da Computação.
- Afiliada ao Departamento de Matemática, UFC.
- Membro do Programa de Pós-Graduação em Matemática (PGMAT) da UFC.
- Membro afiliado da ABC (Academia Brasileira de Ciências) - período 2021-2025.

Fabrício Siqueira Benevides

- Doutor em Matemática, University of Memphis, TN, Estados Unidos, 2011.
- Bolsista de Produtividade em Pesquisa do CNPq - Nível 2 - CA CC - Ciência da Computação.
- Afiliado ao Departamento de Matemática, UFC.
- Membro do Programa de Pós-Graduação em Matemática (PGMAT) da UFC.
- Membro afiliado da ABC (Academia Brasileira de Ciências) - período 2014-2018.

Júlio César Silva Araújo

- Doutor em Ciência da Computação, UFC/Université Côte D'Azur (co-tutela), França, 2012.
- Pós-Doutorado, Centre de Recherche Inria Sophia Antipolis - Méditerranée, França, 2013.
- Pós-Doutorado, Université de Montpellier/LIRMM, França, 2020.
- Bolsista de Produtividade em Pesquisa do CNPq - Nível 2 - CA CC - Ciência da Computação.
- Afiliado ao Departamento de Matemática, UFC.
- Membro do Programa de Pós-Graduação em Matemática (PGMAT) da UFC.

Manoel Bezerra Campêlo Neto

- Doutor em Engenharia de Sistemas e Computação, UFRJ, Brasil, 1999.
- Pós-Doutorado, Carnegie Mellon University, CMU, Estados Unidos, 2009.
- Bolsista de Produtividade em Pesquisa do CNPq - Nível 2 - CA CC - Ciência da Computação.
- Afiliado ao Departamento de Estatística e Matemática Aplicada, UFC.
- Membro do Programa de Pós-Graduação em Ciência da Computação (MDCC) da UFC.
- Membro do Programa de Pós-Graduação em Modelagem e Métodos Quantitativos (MMQ) da UFC.
- Coordenador da CE-ACO da SBC (Comissão Especial em Algoritmos, Combinatória e Otimização)

Rafael Castro de Andrade

- Doutor em Informatique, Université de Paris XIII (Paris-Nord), França, 2002.
- Pós-Doutorado, Université Paris-Sud, França, 2012.
- Bolsista de Produtividade em Pesquisa do CNPq - Nível 2 - CA-EP - Pesquisa Operacional.
- Afiliado ao Departamento de Estatística e Matemática Aplicada, UFC.
- Membro do Programa de Pós-Graduação em Ciência da Computação (MDCC) da UFC.
- Membro do Programa de Pós-Graduação em Modelagem e Métodos Quantitativos (MMQ) da UFC.

Victor Almeida Campos

- Doutor em Ciência da Computação, UFC (Universidade Federal do Ceará), Brasil, 2011.
- Bolsista de Produtividade em Pesquisa do CNPq - Nível 2 - CA CC - Ciência da Computação.
- Afiliado ao Departamento de Computação, UFC.
- Membro do Programa de Pós-Graduação em Ciência da Computação (MDCC) da UFC.

Ana Karolinna Maia de Oliveira

- Doutora em Informatique, Université Côte D' Azur, França, 2014.
- Pós-Doutorado, Universidade Federal do Ceará, 2014-2016.
- Pós-Doutorado, Université de Montpellier/LIRMM, França, 2020.
- Afiliado ao Departamento de Computação, UFC.
- Membro do Programa de Pós-Graduação em Ciência da Computação (MDCC) da UFC.
- 03 artigos em periódicos indexados e 03 artigos em conferências indexadas nos últimos 5 anos.
- 01 orientação de mestrado concluída nos últimos 2 anos e 01 orientação de doutorado em andamento.

Francisco Heron de Carvalho Júnior

- Doutor em Ciência da Computação, UFPE (Universidade Federal de Pernambuco), 2003.
- Afiliado ao Departamento de Computação, UFC.
- Membro do Programa de Pós-Graduação em Ciência da Computação (MDCC) da UFC.
- 05 artigos em periódicos indexados e 06 artigos em conferências indexadas nos últimos 5 anos.
- 06 orientações de doutorado concluídas nos últimos 6 anos e 01 orientação de doutorado em andamento.

Ronan Pardo Soares

- Doutor em Ciência da Computação, UFC, 2013 (co-tutela em Université Côte D' Azur, França).
- Afiliado ao Departamento de Estatística e Matemática Aplicada, UFC.
- Membro do Programa de Pós-Graduação em Modelagem e Métodos Quantitativos (MMQ) da UFC.
- 04 artigos em periódicos indexados e 01 artigo em conferência indexada nos últimos 6 anos.
- 01 orientação de mestrado concluída nos últimos 2 anos.

UNILAB (Universidade da Integração Internacional da Lusofonia Afro-Brasileira)

Allberson Bruno de Oliveira Dantas

- Doutor em Ciência da Computação, UFC, 2017.
- Afiliado ao IEDS (Inst. Engenharias e Desenvolvimento Sustentável) da UNILAB.
- 02 artigos em periódicos indexados e 03 artigos em conferências indexadas nos últimos 5 anos.

Nicolas de Almeida Martins

- Doutor em Ciência da Computação, UFC, 2018.
- Afiliado ao IEDS (Inst. Engenharias e Desenvolvimento Sustentável) da UNILAB.
- 04 artigos em periódicos indexados e 02 artigos em conferências indexadas nos últimos 5 anos.

Pesquisadores Estrangeiros

Andrea Marino

- Doutor em Ciência da Computação, Universidade de Florença, Itália, 2013.
- Afiliado à Universidade de Florença, Itália.
- 50 artigos em periódicos ou conferências indexadas nos últimos 5 anos.

Ignasi Sau Valls

- Doutor em Teoria do Grafos, INRIA Sophia Antipolis, França, 2009.
- Professor visitante na Universidade Federal do Ceará, 2019.
- Afiliado à Université de Montpellier/LIRMM, França.
- 60 artigos em periódicos ou conferências indexadas nos últimos 5 anos.

Mamadou Moustapha Kanté

- Doutor em Informatique, Université de Bordeaux, França, 2008.
- Afiliado à Université Clermont Auvergne, França.
- 25 artigos em periódicos ou conferências indexadas nos últimos 5 anos.

2.2 Alunos

Aluno	Nível
Alexandre Azevedo Cezar	Doutorado
Allen Roossim Passos Ibiapina	Doutorado
Arthur Rodrigues Araruna	Doutorado
Claudio Soares de Carvalho Neto	Doutorado
Efraim Naassom Helem Dantas Rodrigues	Doutorado
Gabriel Hellen de Sousa	Doutorado
Isnard Lopes Costa	Doutorado
Jhonata Adam Silva Matias	Doutorado
Jonas Costa Ferreira da Silva	Doutorado
Luiz Alberto do Carmo Viana	Doutorado
Mardson Da Silva Ferreira	Doutorado
Pedro Jorge de Abreu Figueredo	Doutorado
Pedro Paulo de Medeiros	Doutorado
Rubens Cainan Saboia Monteiro	Doutorado
Ana Beatriz da Silveira Martins	Mestrado
Emanuel Elias Silva Castelo	Mestrado
Felipe Albuquerque Brito da Silva	Mestrado
Francisco Antonio Ferreira de Almeida	Mestrado
Francisco Igor de Souza Oliveira	Mestrado
José Robertty de Freitas Costa	Mestrado
Marcus Vinicius Martins Melo	Mestrado
Marília Cristina do Carmo Viana	Mestrado
Rayane Gomes de Castro	Mestrado
Rodrigo Fernandes Ribeiro	Mestrado
Claro Henrique Silva Sales	Graduação
Davi de Andrade Iácono	Graduação
George Augusto da Silva Filho	Graduação
João Luca Teixeira Carvalho	Graduação
Kenny Wille Domingues	Graduação
Laiane Miranda Alves	Graduação
Leonardo Cavalcante de Abreu	Graduação
Lucas do Vale Pimentel	Graduação
Márcio Barros Oliveira de Souza	Graduação
Matheus Alves da Silva Pascoal	Graduação
Matheus Ribeiro Alencar	Graduação
Matheus Sousa Correia	Graduação
Pedro Amaral Fontes de Sales	Graduação
Rodrigo Nogueira Lima David	Graduação

3 Caracterização do Tema de Pesquisa

O tema deste projeto é a resolução computacional eficiente de problemas de otimização combinatória, envolvendo teoria, algoritmos e implementações sequenciais e paralelas, com aplicações. Nesse sentido, as atividades do projeto seguem os três eixos complementares do ParGO (Paralelismo, Grafos e Otimização). Para facilitar a descrição, dividimos o tema central em 4 sub-temas correlacionados, brevemente apresentados a seguir. Detalhes técnicos serão apresentados na Seção 9.

3.1 Estruturas Matemáticas para Resolução de Problemas Computacionais

Muitos dos problemas investigados neste projeto estão entre os mais difíceis computacionalmente. Assim, faz-se necessário o estudo estrutural do problema com o objetivo de poder classificar a sua complexidade computacional de tempo e de espaço/memória, bem como a sua complexidade parametrizada. Ademais, também é importante a análise matemática estrutural das instâncias do problema, que neste projeto são, em sua maioria, grafos. Essa análise costuma auxiliar na obtenção de algoritmos eficientes para instâncias de médio ou grande porte, bem como instâncias bem estruturadas, de certas classes de grafos, comuns nas aplicações.

3.1.1 Análise Combinatória Estrutural

Em muitos casos, as instâncias de grande porte seguem padrões estruturais em âmbito global que, após uma análise adequada, permitem estimar certos parâmetros (*parameter estimation*), bem como verificar a ocorrência de certas propriedades (*property testing*) de modo rápido. Essas abordagens muitas vezes são úteis para a compreensão do problema em questão. Assim, é importante o estudo de técnicas para a análise da estrutura combinatória das instâncias de grande porte, como a Teoria de Ramsey, a técnica dos *containers*, a *flag* álgebra e a teoria de objetos limites, como *graphons* e *permutons*.

3.1.2 Classes de Grafos

Um grafo é uma estrutura matemática muito utilizada na descrição dos problemas tratados neste projeto. Há problemas formulados sobre grafos que são classificados como computacionalmente difíceis, quando o objetivo é a obtenção de um algoritmo eficiente para qualquer grafo que lhe seja apresentado como entrada. No entanto, o estudo de algoritmos para um problema restrito a grafos especiais pode trazer conhecimentos sobre o problema. Essa abordagem consiste em supor que o grafo possui alguma propriedade estrutural especial e explorar essa propriedade no algoritmo. Tal propriedade estrutural define uma classe de grafos, como, por exemplo, as *árvores* ou os *grafos planares*. O objetivo passa a ser construir um algoritmo eficiente para uma classe de grafos e, em seguida, explorar as ideias utilizadas para resolver para classes de grafos cada vez maiores ou com menores restrições estruturais.

3.1.3 Complexidade Computacional e Complexidade Parametrizada

A classificação de um problema como computacionalmente difícil é de extrema importância para o entendimento do mesmo. Existem várias classificações de dificuldade: com relação ao tempo (como a classe NP-Difícil para problemas de otimização combinatória), com relação ao espaço/memória (como a classe PSPACE-Difícil para problemas geralmente relacionados a jogos computacionais, alguns tratados na Seção 3.3.2 deste projeto), com relação à aproximabilidade do problema (possibilidade de obtenção de algoritmos aproximativos, como as classes PTAS e APX-Difícil), bem como com relação à tratabilidade por parâmetro fixo (como as classes XP, FPT, W[1]-Difícil e W[2]-Difícil para problemas parametrizados). Esta última faz parte de uma área da Teoria da Computação em grande crescimento, conhecida como Complexidade Parametrizada, que contém diversas técnicas para obtenção de algoritmos FPT (*fixed parameter tractable*): algoritmos eficientes para instâncias em que certo parâmetro é limitado, assim como para demonstrar, sob certas hipóteses de complexidade, que tais algoritmos não devem existir. Entre essas técnicas, destaca-se a *kernelização*, cuja ligação quanto à aproximabilidade do problema tem sido alvo de pesquisa recente no ParGO (ver Seção 9.5.1). Finalmente, a própria classificação dos problemas em algumas dessas classes de complexidade necessita de técnicas específicas de redução entre problemas, que também é alvo de investigação deste projeto.

3.2 Métodos Algorítmicos para Otimização Combinatória

Uma das linhas de ação deste projeto se constitui na elaboração de métodos algorítmicos eficientes para os problemas estudados. Como em geral tratam-se de problemas complexos, para os quais não se espera

algoritmos polinomiais no tamanho da sua entrada, diferentes abordagens, usualmente combinadas, serão empregadas para a construção dos métodos de solução. Descrevemos brevemente a seguir algumas dessas abordagens, destacando relações entre elas. Vale ressaltar que, no intuito de gerar algoritmos eficientes, a melhor forma de aplicação/combinação dessas abordagens pode variar de um problema para outro e sua determinação constitui linha relevante de pesquisa.

3.2.1 Métodos de Decomposição

A decomposição de um problema complexo ou de grande porte é uma estratégia amplamente empregada como uma forma de estabelecer subproblemas total ou parcialmente independentes. Mais precisamente, decompor um problema significa desmembrar a estrutura que o descreve matematicamente em um certo conjunto de estruturas menores que tenham alguma relação com aquela de partida. As estruturas obtidas representam subproblemas do problema original. Quando uma decomposição adequada é conhecida, a resolução dos subproblemas, com a consequente combinação dos seus resultados para formar o resultado do problema original, requer menor esforço computacional do que a resolução do problema não-decomposto, possibilitando assim a obtenção de algoritmos mais eficientes. Neste projeto, particular atenção é dada aos métodos combinatórios de decomposição de grafos e aos métodos de decomposições de formulações linear-inteiras, que serão frequentemente usadas para modelar os problemas de otimização aqui tratados. A abordagem de decomposição de problemas tem importância especial para a obtenção de algoritmos paralelos para resolvê-los.

3.2.2 Relaxações

Grosseiramente falando, a relaxação de um problema consiste em desconsiderar algumas de suas restrições, com o propósito de tornar sua solução possível por meio de técnicas conhecidas (as quais não poderiam ser usadas ou seriam computacionalmente inviáveis mantendo-se as restrições originais). Um fato frequente é que o problema, uma vez relaxado, toma uma forma mais adequada à decomposição que gere subproblemas independentes. Em geral, a solução do problema obtido pela relaxação é uma aproximação da solução do problema original. Essa aproximação é útil, por exemplo, para descartar alguns dos subproblemas gerados pela decomposição, concentrando o esforço de resolução nos subproblemas que podem levar a uma solução ótima. Em particular, os problemas tratados neste projeto podem ser objeto de relaxações e decomposições associadas a restrições do problema (por exemplo, relaxação linear e relaxação Lagrangeana) e também a variáveis (por exemplo, operações de *lift-and-project* e decomposição de Dantzig-Wolfe e Benders). Mais detalhes em [199].

3.2.3 Planos de Corte

Muitos dos problemas de otimização tratados neste projeto podem ser modelados via programação linear-inteira. Nesse caso, uma relaxação usual é obtida removendo-se as restrições de integralidade das variáveis. Encontrada uma solução ótima para essa relaxação (que em geral não resolve o problema original), um plano de corte consiste em uma desigualdade linear que é violada por tal solução, mas é satisfeita por todas as soluções do problema. O acréscimo dessa nova condição fortalece a relaxação, aproximando-a mais do problema original. Um estratégia geral de solução consiste em, iterativamente, resolver uma relaxação e acrescentar planos de corte. Encontrar planos de corte que abreviem esse processo é uma questão de grande relevância para solução eficiente dos problemas tratados nesse projeto. Adicionalmente, constitui importante desafio a determinação de formas eficientes de combinar a geração desses planos de corte com a decomposição do problema.

3.2.4 Descrição de Poliedros

O estudo estrutural de poliedros associados a problemas de otimização combinatória e de programação inteira fornece conhecimento útil para a descrição de suas soluções. Esse estudo pode explorar propriedades específicas de uma aplicação abordada ou mesmo explorar simplesmente duas características básicas comuns aos

problemas envolvidos nesse projeto: a linearidade das restrições e a integralidade das variáveis que os definem. No primeiro caso, costumam-se obter resultados mais fortes para uma aplicação de particular interesse. Já o segundo é normalmente motivado pela abrangência das propriedades que podem ser obtidas. Em qualquer caso, o conhecimento oriundo desse estudo pode ser utilizado na elaboração e utilização de heurísticas para determinação de planos de corte que servem para fortalecer as relaxações.

3.3 Aplicações

Os problemas que recebem maior atenção neste projeto são problemas considerados computacionalmente difíceis segundo o número de passos que os algoritmos que os resolvam realizam, em função do tamanho da entrada fornecida. Para esses problemas, esse número de passos aumenta exponencialmente com o tamanho da entrada, o que significa que, mesmo para instâncias de tamanho pequeno, a obtenção de um algoritmo eficiente não é uma tarefa fácil (talvez até mesmo impossível). A investigação desses problemas, pela sua dificuldade, faz avançar o conhecimento teórico sobre as estruturas matemáticas a eles associadas, assim como sobre técnicas algorítmicas e de implementação. Buscando conjugar o interesse científico com a aplicabilidade prática dos resultados produzidos, elegemos algumas aplicações de particular interesse neste projeto, a seguir enumeradas, sem contudo restringirmos, a priori, o escopo do projeto a esta lista. Outras aplicações poderão somar-se a estas ao longo do desenvolvimento do projeto.

3.3.1 Coloração de Grafos

O clássico problema de Coloração de Vértices (colorir os vértices de modo que vértices adjacentes tenham cores diferentes) usando o menor número possível de cores é um dos problemas mais estudados da Teoria dos Grafos, além de um dos mais difíceis computacionalmente [136, 168, 138]. Além das motivações teóricas (como o famoso Teorema das 4 cores [11, 174]), boa parte do interesse se deve ao fato de que muitos problemas práticos podem ser modelados desta forma, em especial problemas relacionados a redes de comunicação. De fato, a gama de aplicações é extensa e citamos apenas alguns problemas que podem ser modelados diretamente como um problema de coloração própria de grafos: alocação de frequências, escalonamento de tarefas (em processadores, ou máquinas de uma fábrica, etc), alocação de registradores, planejamento de torneios, classificação de um conjunto em grupos de semelhança (clusterização), etc. Particularmente, essa última aplicação de coloração aparece no problema de patrulhamento por viaturas, onde se deseja dividir uma certa cidade (modelada pelo grafo) em zonas de patrulhamento (as classes de cor), de modo que o deslocamento entre dois pontos de uma mesma zona possa ser feito em um tempo máximo pré-estabelecido (isso define as arestas). Restrições físicas também podem ser consideradas na definição da classe de cor, como por exemplo a "forma geométrica" da zona, levando assim a problema de coloração de vértices mais restrito.

Além disso, variações do problema de coloração são alvo de muita pesquisa científica. Por exemplo, no problema da Coloração de Arestas, são as arestas que devem ser coloridas de modo que arestas de um mesmo vértice tenham cores diferentes. Esse problema modela situações como a seguinte: em um campeonato, cada time deve jogar com os demais, mas nenhum time pode jogar duas vezes em um mesmo dia. O planejamento do torneio pode então ser visto como uma coloração das arestas do grafo completo, onde cada cor representa um dia distinto. A partir daí, várias outras restrições podem surgir, como, por exemplo, por motivos televisivos, não pode haver mais de certo número de jogos em um mesmo dia. Em outras palavras, o problema investigado trata-se de coloração de arestas do grafo completo onde apenas uma quantidade limitada de arestas pode receber a mesma cor.

Estas restrições adicionais, tão comuns nos problemas práticos, são o que originam as muitas variações do problema de coloração. Neste projeto, propomos a investigação de algumas variações do problema de coloração advindas de aplicações específicas, como o problema de patrulhamento de viaturas, ou variações já formalmente definidas e estudadas na literatura. Em particular, estudaremos variantes definidas a partir de heurísticas para coloração (coloração gulosa, gulosa conexa, gulosa parcial, b-coloração, *subfall*), além de outras variantes como coloração *backbone* e orientação própria. Para exemplificar alguns problemas práticos que podem ser modelados por estas últimas variações, citamos que: a b-coloração tem sido usada em proble-

mas de clusterização; a coloração gulosa é um dos algoritmos não-exatos de coloração própria mais utilizados na prática; a coloração *backbone* foi definida pensando em problemas práticos de alocação de frequência onde alguns pares possuem um maior grau de interferência entre si.

3.3.2 Conectividade

Em muitos problemas práticos, deseja-se determinar uma interligação entre dois ou mais elementos de um universo, como por exemplo um caminho entre dois pontos de uma cidade, uma rede de comunicação entre computadores, uma rede para distribuição de água etc. Naturalmente, esses problemas podem ser modelados por grafos, e essas interligações se traduzem nos conceitos de caminho, árvore, fluxo etc. Alguns dos problemas tradicionais nesse universo, como caminho mínimo, árvore geradora mínima e fluxo máximo, possuem soluções bem conhecidas e eficientes. Não raro, porém, esses problemas aparecem com restrições adicionais ou em outros contextos que não podem ser diretamente modelados por grafos simples.

Essas variações tornam, muitas vezes, os problemas básicos desafiadores. Tanto por esse aspecto, quanto pela larga aplicabilidade em cenários atuais, a pesquisa sobre problemas de conectividade em grafos continua bastante ativa. Nessa linha, destacamos alguns temas de interesse. Nos dois primeiros, consideramos outros parâmetros na definição dos grafos, além dos vértices e arestas, a saber:

- grafos temporais, onde vértices e arestas mudam ao longo do tempo, modelando assim redes de conexão dinâmicas - nesse caso, o conceito de caminho é estabelecido também em função dessa característica, posto que as arestas precisam existir efetivamente no momento em que são utilizadas (mais detalhes na Seção 9.2.1);
- grafos rotulados em arestas, onde os rótulos identificam arestas com características similares (por exemplo, *links* de comunicação que usam uma mesma tecnologia) - aqui, parâmetros relativos a um caminho (como tamanho) podem depender dos rótulos das arestas nele presentes (mais detalhes na Seção 9.2.2).

Nos outros temas, consideramos restrições adicionais além da conectividade:

- *t-spanners*, que consiste num subgrafo contendo todos os vértices do grafo e tal que a distância entre cada par de vértices não exceda um múltiplo t da distância original - esse conceito aparece em contextos em que se deseja encontrar uma rede conexão com menos ligações que a original mas que ainda permita uma troca de mensagens eficiente/barata (mais detalhes na Seção 9.2.3);
- fluxos arco-disjuntos: entendendo um fluxo com uma rede de caminhos, há cenários em que a interseção entre eles precisa ser monitorada, levando ao conceito de fluxos arco-disjuntos (mais detalhes na Seção 9.2.4);
- árvore dominante, ou seja, uma estrutura minimamente conexa entre um subconjunto de vértices, tal que cada outro vértice se ligue diretamente a algum desse subconjunto - essa estrutura aparece na modelagem de redes de sensores, por exemplo (mais detalhes na Seção 9.2.5).

Estudaremos essas estruturas de maneira geral ou em aplicações específicas (ver Seção 9)

3.3.3 Jogos Multiagentes em Grafos

Vários jogos multiagentes em grafos têm sido alvo de pesquisas recentes, como jogos de coloração em grafos e jogos de perseguição em grafos. No clássico Jogo de Coloração [42], introduzido em 1991, dois agentes (Alice e Bob) colorem alternadamente os vértices do grafo de maneira própria, tendo Alice o objetivo de minimizar o número de cores e Bob maximizar o número de cores. A complexidade desse problema estava em aberto desde 1991 e foi provado ser PSPACE-Difícil somente em 2020 por dois membros deste projeto [82]. Algumas variantes desse jogo, como o jogo de coloração conexa ou o jogo de coloração gulosa também tem atraído atenção recentemente.

No grupo dos jogos de perseguição em grafos, destacam-se o jogo de polícia e ladrão (*Cops and Robber game* [45]), o jogo da dominação eterna [122] e o jogo do espião (*Spy game* [76]). No jogo de polícia e ladrão,

um certo número k de policiais cooperam entre si para capturar um ladrão, onde todos os agentes do jogo se posicionam sobre vértices e se movem alternadamente sobre arestas do grafo. O objetivo dos policiais é posicionar um policial no mesmo vértice do ladrão. O jogo do espião é semelhante, mas o espião é mais rápido, podendo se mover através de um certo número dado de arestas (a velocidade s do espião). Aqui o objetivo dos policiais é sempre manter um policial a certa distância pré-determinada (a distância d de vigilância) do espião. A questão em ambos os jogos é computar o menor número k de policiais necessários para vencer o jogo sobre o grafo. Dois membros deste projeto provaram que uma variação direcionada do jogo do espião é PSPACE-Difícil [76], mas a complexidade do problema original permanece em aberto. O jogo de dominação eterna [122], no qual a velocidade s do espião é infinita, também foi alvo de vários artigos recentes.

3.3.4 Difusão e Propagação em Redes Sociais

As redes sociais são uma realidade em nossos dias e constituem meios poderosos para divulgação de informação e conteúdos. Nesse contexto, aparecem diversas questões relevantes, tais como a identificação do menor número de indivíduos necessários para propagar uma informação sobre uma comunidade de interesse, ou ainda a determinação do tempo máximo para que uma informação seja espalhada para toda a rede. Uma outra aplicação similar, muito estudada atualmente em função da pandemia, é o processo de contaminação de populações por um agente infeccioso. Problemas dessa natureza podem ser modelados por meio do conceito de convexidade em grafos, particularmente convexidade de intervalos.

Uma função de intervalo sobre um grafo $G = (V, E)$ recebe um subconjunto de vértices e devolve outro subconjunto de vértices, contendo o original. Precisamente, é uma função $I : 2^V \to 2^V$ tal que $I(\emptyset) = \emptyset$ e $S \subseteq I(S) \subseteq I(S')$ para todo $S \subseteq S' \subseteq V$. Por simplicidade, dizemos que os vértices de $I(S) \setminus S$ são *ativados*, *contaminados*, *infectados* ou *gerados* por S. Os pontos fixos dessa função $\{S \subseteq V : I(S) = S\}$ são chamados conjuntos convexos (com respeito à convexidade definida por I). Note que $\emptyset$ e V são convexos. A envoltória convexa (*hull set*) de S é o menor conjunto convexo que contém S. Deve-se notar que a envoltória de um certo conjunto $S \subseteq V(G)$ pode ser obtida iterativamente, pela aplicação sucessiva da função de intervalo I. Tal processo, pode ser visto como o processo de "difusão da informação" ou de "contaminação" do grafo dado a partir deste conjunto inicial de vértices S.

Na literatura, são consideradas várias funções de intervalo. Note que cada uma corresponderá a um processo de "difusão/contaminação" distinto, com suas possíveis aplicações práticas. Por exemplo, $I(S)$ igual a S unido a todos os vértices com pelo menos 2 vizinhos em S; ou ainda $I(S)$ igual a S junto com todos os vértices em algum caminho mínimo entre pares de elementos de S. Essas duas funções definem respectivamente as convexidades P_3 [182] e geodésica [177].

O problema TSS (*Target Set Selection* [75]), também bastante estudado recentemente, pode ser modelado como Convexidade de Grafos. Cada vértice v do grafo possui um valor limite $\lim(v) > 0$ de modo que um vértice não ativado v é ativado se possui $\lim(v)$ vizinhos já ativados. Nesse caso, a função de intervalo $I(S)$ é definida como sendo S unido a todo vértice v com pelo menos $\lim(v)$ vizinhos em S. Nesse contexto, são interessantes problemas como determinar o menor número de vértices para ativar todo o grafo (TSS-SIZE [75]), ou determinar o tempo máximo para que o grafo inteiro seja ativado (TSS-TIME [139]), ou determinar o menor número de vértices para que o grafo inteiro seja ativado em um único passo (VECTOR-DOMINATION ou TSS-DOMINATION [158]), bem como determinar o maior conjunto que seja convexo (TSS-MAX-CONVEX), incapaz de ativar novos vértices, para identificação de comunidades "fechadas" em uma rede social.

3.3.5 Redes Neurais Baseadas em Grafos

Nos últimos anos, as redes neurais profundas (DNN[1]) [124] têm se mostrado eficazes para viabilizar a Inteligência Artificial em uma série de aplicações de natureza estratégica, em muitas áreas diferentes, tanto de interesse científico quanto industrial. Problemas comumente difíceis de extrair representação, tais como reconhecimento de objeto em imagens e vídeos, processamento de sinais, processamento de linguagem natural, com aprendizagem profunda podem ser resolvidos de forma eficiente. A aprendizagem profunda consiste em

[1] *Deep Neural Network*

uma extensão multicamadas dos classificadores de redes neurais tradicionais, empregando um mecanismo de treinamento baseado em *feed-forward*, *backpropagation* e otimização dos vetores de peso de camada. O poder de representação vem de vários tipos diferentes de camadas, que basicamente representam relações lineares ou não lineares entre a entrada e a saída da rede neural. A utilização das mais diferentes funções de ativação no final das camadas adiciona interessantes graus de não linearidade, porém aumentando a necessidade de processamento eficiente e maiores recursos computacionais.

Podem-se apontar diferentes tipos de redes neurais profundas, geralmente classificadas pelo tipo específico de problema que pretendem atacar. O tipo mais popular consiste nas Redes Neurais Convolucionais (CNN), que são bem-sucedidas para detecção e reconhecimento de imagens. Dado um conjunto de filtros comumente pré-disponível (*kernels*), algoritmos de rede convolucional são capazes de empregá-los para extrair imagens características de regiões locais ou de padrões invariáveis de localização [149].

Com o aumento da complexidade das aplicações em que se deseja aplicar técnicas de Inteligência Artificial, motivadas pelo sucesso das redes neurais profundas, tem-se observado uma tendência crescente da aplicação desse tipo de rede neural para resolver problemas em conjuntos de dados com estrutura irregular que podem ser representados através de grafos [92, 143, 151], oriundos sobretudo de interações em redes sociais, processamento em bioinformática e neurociência, dentre outros. Para esses problemas, Redes Neurais baseadas em Grafos (GNNs[2]) têm demonstrado resultados promissores, ao mapear as *features* do problema a vértices e arestas de um grafo e propagar e agregar os resultados para produzir *features* para a camada seguinte da rede neural [156].

Nenhum dos principais motores de processamento eficiente de grandes grafos possui suporte a GNNs [160, 154, 123, 201]. Já os arcabouços de aprendizagem profunda mais difundidos, tais como TensorFlow, PyTorch e MxNet, não oferecem processamento escalável necessário para lidar com grandes grafos. Tais deficiências têm limitado fortemente a exploração do potencial das GNNs em escala, contudo permitem que novas iniciativas de pesquisa sejam propostas em algumas frentes, tais como modelos de programação paralela eficientes para GNNs.

Isto posto, pretende-se no projeto, utilizando a expertise do grupo em grafos, avançar no estado-da-arte em técnicas de processamento paralelo escaláveis para aprendizado em GNNs. Alguns problemas em aberto neste sentido decorre do fato de que os motores de processamento de grafos mais difundidos empregam em geral um estilo de programação de corte de vértices conhecido como Gather-Apply-Scatter (GAS) [123], derivado do modelo Bulk Synchronous Parallel (BSP) [73], o que dificulta o mapeamento de arquiteturas de redes neurais baseadas em grafos nas construções dos grafos analisados. Nesse sentido, por exemplo, investigar o mapeamento eficiente de estruturas de redes neurais baseadas em grafos em outros modelos de programação paralela configura questão de interesse. Outra questão de investigação relevante diz respeito à ausência nos arcabouços de aprendizagem profunda populares de implementações de operadores de propagação de grafos para GPU.

3.4 Computação Paralela na Resolução de Grandes Instâncias de Problemas

Os problemas de otimização combinatória considerados nesta proposta estão classificados entre aqueles para os quais não são conhecidos algoritmos eficientes, o que justifica todo o aparato teórico e algorítmico empregado para resolvê-los, descritos nas Seções 3.1 e 3.2. Ao mesmo tempo, são justamente as instâncias de médio e grande porte desses problemas que interessam às aplicações, tais como as descritas na Seção 3.3, para as quais mesmo os melhores algoritmos que possamos construir frequentemente não tornam viável a sua aplicação prática em sistemas de computação convencionais. Por essa razão, dado o interesse estratégico dessas aplicações, tanto para as ciências quanto para a indústria e governos, motiva-se a pesquisa em plataformas e técnicas de *Computação de Alto Desempenho* (HPC[3]), com o propósito de explorar ao máximo o estado-da-arte do desempenho de sistemas de computação

A principal premissa de investigadores e projetistas de sistemas de computação de alto desempenho (sistemas HPC) é a exploração de recursos de múltiplos sistemas de computação agregados em um único

[2] *Graph Neural Networks*
[3] *High Perfomance Computing*

sistema cujo potencial de desempenho aproxime-se da soma de suas partes, de onde surge o conceito de *computação paralela*. Fundamentalmente, a computação paralela envolve contribuições de três áreas:

- **arquiteturas de computadores**, de onde surgem projetos de *plataformas de computação paralela*, incluindo processadores com capacidade superescalar, vetorial e multicore, aceleradores computacionais, tais como GPUs, interconexões de alto desempenho e a agregação de plataformas independentes, de forma recursiva, em uma mesma plataforma (e.g. clusters e MPPs (Massive Parallel Processors);
- **algoritmos**, de onde surgem versões paralelas de algoritmos sequenciais, capazes de explorar o desempenho de plataformas de computação paralela;
- **linguagens de programação**, de ondem surgem artefatos de desenvolvimento de *programas paralelos*, ou componentes paralelos de software de larga escala, os quais implementam os algoritmos paralelos de forma eficiente para uma ou mais plataformas de computação paralela alvo.

Um *sistema de computação paralela* é uma combinação de uma plataforma de computação paralela e um conjunto de algoritmos paralelos implementados, através de uma ou mais linguagens de programação apropriadas, de forma a explorar de forma eficiente o desempenho da plataforma alvo na resolução de problemas de interesse de uma aplicação. Tais problemas apresentam alta demanda computacional, caracterizando requisitos de sistemas HPC. Portanto, vale enfatizar, no contexto de sistemas HPC, o desenvolvimento de algoritmos paralelos, bem como a escolha da linguagem de programação para sua implementação, é intimamente influenciada pela arquitetura da plataforma de computação paralela alvo, de forma a buscar explorar ao máximo o seu potencial de desempenho.

3.4.1 Estado-da-Arte e Tendências em Sistemas de Computação Paralela

As tecnologias em torno da HPC evoluem de forma rápida, impulsionadas pelo interesse na viabilização de aplicações estratégicas. Esse interesse força a indústria a propor novas arquiteturas e componentes de plataformas de computação paralela. A tendência atual na arquitetura dessas plataformas, de acordo com a iniciativa Top500[4], são clusters e MPPs constituídas por muitos nós de processamento de memória distribuída e múltiplas hierarquias de paralelismo, onde um único nó de processamento é um multiprocessador, i.e., constituído por um grande número de processadores capazes de se comunicar pela memória compartilhada oferecida pelo nó, cada qual com algumas dezenas de núcleos de processamento. Além disso, empregam aceleradores computacionais, tais como FPGAs [129], GPUs [106] e TPUs [137], em diferentes configurações, geralmente com múltiplas unidades por nó de processamento. Enquanto clusters empregam componentes de prateleira, bem como são de propósito geral, MPPs empregam tecnologias proprietárias e/ou especializadas visando otimização de desempenho geralmente visando um certo nicho de aplicações (propósito especial). Logo, uma característica importante das plataformas de computação paralela modernas é a heterogeneidade.

Na *computação heterogênea* [203], desenvolvedores de sistemas de computação paralela são desafiados a empregar múltiplas técnicas e artefatos de desenvolvimento de programas paralelos, dentre linguagens de programação, bibliotecas de subrotinas e *frameworks* de mais alto nível [167]. A computação heterogênea é considerada essencial para viabilizar instâncias de grande porte em aplicações de interesse estratégico, tais como otimização, simulação computacional e Inteligência Artificial (IA), alcançando sistemas de computação paralela hexa-escaláveis, cujo desempenho é medido na ordem dos hexaflops[5] [3].

Uma outra tendência em sistemas HPC é o paralelismo de larga escala, envolvendo recursos de computação geograficamente distribuídos através da abstração de recursos (grades computacionais [117]) ou, mais recentemente, de serviços (nuvens computacionais [55]), motivado pelo crescente aumento do desempenho e conectividade em redes de longa distância bem como o interesse em aplicações com requisitos BigData. A "HPC em nuvens", nesse contexto, surgiu motivada por aspectos econômicos, permitindo reduzir os custos da aquisição de recursos de computação paralela de alta tecnologia por parte de instituições acadêmicas e

[4] `http://www.top500.org`
[5] 10^{15} operações de ponto flutuante por segundo.

indústrias. Na última década, surgiram vários frameworks de computação de larga escala baseados em paradigmas específicos de computação paralela, tais como MapReduce [150] e processamento de streams [83, 189], fortemente utilizados em aplicações comerciais e industriais com requisitos HPC e BigData.

A computação heterogênea e a demanda por computação paralela em larga escala somam-se a outros fatores, antecedentes, no aumento da complexidade de sistemas de computação paralela nas últimas duas décadas. De fato, a preocupação com a inadequação de artefatos e métodos de engenharia de software nesse contexto ganhou força a partir do início dos anos 2000 a partir da comunidade das ciências computacionais [186, 185], tradicionais usuários de sistemas HPC, motivando o surgimento de consórcios acadêmico-industriais interessados no projeto de novas linguagens de programação e arcabouços de desenvolvimento de software [193, 155], dentre os quais destacamos o *desenvolvimento baseado em componentes* [25, 30, 38]. Concomitante a essas iniciativas, ampliava-se o conjunto dos nichos de aplicação para tecnologias HPC, além do nicho tradicional das ciências computacionais e engenharias, alcançando aplicações de interesse comercial, corporativas e no mercado financeiro, recentemente alavancadas pelo interesse crescente em aplicações de Inteligência Artificial (IA) [133] e BigData que também demandam por HPC, especialmente computação heterogênea.

3.4.2 Contribuições Antecedentes em Sistemas HPC Orientados a Componentes

Nos últimos 15 anos, este grupo de pesquisa tem se dedicado a contribuir no avanço do estado-da-arte do desenvolvimento orientado a componentes voltado a sistemas HPC, em particular sistemas de computação paralela, em confluência com os novos desafios na área de HPC na última década, descritos na seção anterior. A partir de projetos iniciais financiados pelo Edital Universal do CNPq (486021/2011-4 e 480307/2009-1), pesquisadores do grupo ParGO desenvolveram novos modelos, arcabouços e plataformas de componentes voltados à construção de sistemas paralelos, tais como: o modelo de componentes paralelos denominado Hash [89]; a plataforma de componentes HPE (Hash Programming Environment), voltado à computação em clusters [90]; e a plataforma HPC Shelf (`http://www.hpcshelf.org`), voltada à oferta de serviços para construção, implantação e execução de sistemas de computação paralela orientados à componentes através da abstração de nuvens computacionais [91, 87]. De fato, a HPC Shelf é um produto resultante de contribuições de cinco teses de doutorado desenvolvidas no contexto deste grupo de pesquisa, entre os anos de 2011 e 2019.

Atualmente, busca-se a disseminação do uso da HPC Shelf através da construção de aplicações que possam atender às necessidades de nichos de usuários HPC. Com esse propósito, por exemplo, foi desenvolvido durante o ano de 2021, sobre os serviços da HPC Shelf, uma aplicação de propósito geral denominada Swirls (`https://www.hpcshelf.org/#h.8eglmidqf5uh`), cujo propósito é permitir a criação, implantação e execução interativa, por meio de notebooks Jupyter[6], de sistemas de computação paralela capazes de executar programas paralelos MPI escritos nas linguagens C, C++, C# ou Python, em clusters, chamados de *plataformas virtuais* no contexto da HPC Shelf, instanciados sobre provedores IaaS suportados pelo arcabouço de componentes oferecido pelo Swirls. Em sua versão atual, o Swirls oferece suporte para criação de plataformas virtuais sobre os serviços Google Cloud Platform (GCP)[7] e Amazon Elastic Compute Cloud (EC2)[8]. Além da simples execução de programas MPI, o Swirls oferece o suporte para comunicação por passagem de mensagens entre diferentes programas MPI, instanciados em diferentes plataformas virtuais, através de componentes conectores, oferecendo portanto o suporte à computação paralela de larga escala envolvendo múltiplos clusters. Dizemos portanto que o Swirls atende requisitos multicloud e multicluster, devido a sua capacidade de explorar os recursos de múltiplos clusters possivelmente instanciados sobre diferentes infraestruturas de computação paralela.

3.4.3 HPC Shelf para Resolução de Problemas de Otimização Combinatória

Como dito no início desta seção, em aplicações de interesse prático, tais como aquelas apresentadas na Seção 3.3 para este projeto, os problemas de otimização abordados pelos pesquisadores do ParGO motivam requisitos HPC nessas aplicações, levando a dois eixos em torno dos quais a HPC pode ser empregada:

[6] `http://www.jupyter.org`
[7] `http://cloud.google.com`
[8] `http://https://aws.amazon.com/ec2`

- o apoio ao desenvolvimento de versões paralelas de algoritmos nas diferentes frentes tratadas pelo grupo de pesquisa, adaptados a diferentes paradigmas de computação paralela (computação heterogênea), bem como o estudo teórico da complexidade e escalabilidade deses algoritmos;
- o emprego da HPC Shelf como plataforma para compartilhamento e disseminação das contribuições do grupo referentes à implementação de algoritmos voltados a aplicações de uso prático, bem como acesso a recursos de computação heterogênea, tanto nos equipamentos disponíveis nos laboratórios quanto em nuvens computacionais.

Muitos dos métodos e técnicas utilizadas nas implementações das soluções para esses problemas são comuns a diferentes aplicações. Portanto, o reaproveitamento de código da implementação de algoritmos, em suas versões tanto sequenciais quanto paralelas, em diferentes aplicações, sem que isso cause impacto significativo sobre o desempenho das aplicações, é um requisito importante a ser considerado no método de trabalho deste grupo de pesquisa. A experiência de pesquisadores do ParGO em técnicas de desenvolvimento baseado em componentes para sistemas HPC tem o potencial de não somente aprimorar o compartilhamento de contribuições entre os pesquisadores do grupo, mas também com a comunidade em geral. Para essa finalidade, a plataforma HPC Shelf pode ser empregada, tornando possível a construção de arcabouços de componentes para construção das aplicações de interesse do grupo, bem como sua disponibilização para a comunidade externa. Além disso, através do Swirls, pesquisadores do ParGO podem usufruir de forma mais simples, quando necessário, de recursos de HPC oferecidos por provedores IaaS, em nuvens, incluindo processadores e aceleradores de última geração que não podem ser adquiridos com recursos do projeto devido ao seu custo, bem como grande capacidade de memória e armazenamento.

Vale ressaltar que, além das contribuições notórias que a HPC Shelf e o Swirls podem oferecer para os trabalhos dos pesquisadores deste grupo de pesquisa, as aplicações oriundas das contribuições desse projeto também servem para validação e evolução da própria plataforma, bem como uma forma de divulgar a própria HPC Shelf, bem como o Swirls, para a comunidade externa, o que, como mencionado anteriormente, constitui um objetivo importante na etapa atual do trabalho dos pesquisadores do grupo envolvidos com a HPC Shelf.

4 Objetivos

4.1 Objetivos Científicos

Os objetivos gerais são:

1. estabelecer modelos, formulações e propriedades estruturais para problemas de otimização combinatória, especialmente aqueles definidos sobre coloração de grafos, conectividade de grafos e difusão em grafos;
2. desenvolver algoritmos exatos, aproximativos e heurísticos eficientes usando técnicas de decomposição, particionamento, kernelização;
3. obter implementações computacionais sequenciais e paralelas capazes de resolver instâncias de grande porte desses problemas.

Os objetivos específicos incluem:

1. Estudar problemas relacionados à coloração de vértices, particularmente coloração backbone, orientações próprias e variantes de coloração baseadas em heurísticas, analisando suas complexidadades em classes de grafos, elaborando formulações de programação inteira e descrevendo os politopos associados;
2. Estudar problemas relacionados à difusão em grafos, particularmente *Target Set Selection*, convexidade P_3, convexidade geodésica, convexidade monofônica, convexidade circular, entre outras;
3. Desenvolver algoritmos exatos e heurísticas para problemas sobre conectividade em redes de comunicação, entre eles problemas de grafos temporais, grafos rotulados em arestas, t-spanners, fluxos arco-disjuntos e redes de sensores clusterizados;

4. Elaborar e implementar algoritmos paralelos para problemas definidos por restrições sobre coloração e conectividade em grafos, usando técnicas de modelagem matemática;

5. Utilizar o "framework" de componentes para implementação paralela e distribuída eficiente dos algoritmos, reutilizando códigos comuns em diferentes aplicações.

4.2 Objetivos Institucionais

Os principais objetivos institucionais são: (a) Consolidar o grupo de pesquisa ParGO nacional e internacionalmente, aumentando a sua inserção no meio acadêmico e a sua relevância para o desenvolvimento da Ciência da Computação no Brasil e no mundo, (b) Contribuir para a promoção, popularização e divulgação científica, a partir das áreas de atuação do grupo. Como metas, incluímos:

1. Disseminação das contribuições conceituais por meio de publicações em periódicos e conferências nacionais e internacionais

2. Formação de recursos humanos nas áreas envolvidas, em nível de pós-graduação: 8 doutores e 10 mestres;

3. Fortalecimento das cooperações nacionais e internacionais através de visitas de trabalho e orientação de teses de doutorado em co-tutela;

4. Acolhimento de jovens doutores no seio do grupo com vistas a futuras expansões do corpo docente do grupo.

5. Criação de página na Internet para transparência do projeto, mostrando objetivos, pesquisadores e disponibilizando a produção científica (toda ela estará com livre acesso através do portal arXiv), bem como disponibilização dos códigos computacionais.

6. Promoção, popularização e divulgação científica por meio de atividades semestrais voltadas a alunos do ensino médio, com apresentações lúdicas e interativas, procurando mostrar como pesquisas do projeto ajudam na resolução de problemas concretos da vida real.

5 Metodologia

Como sugere a amplitude dos objetivos descritos acima, as linhas de ação deste projeto irão demandar uma abordagem abrangente, integrando atividades interrelacionadas. A metodologia a ser seguida reflete esse desafio e se baseia nas atividades usuais do ParGO, o que inclui:

1. aquisição de conhecimentos teóricos sobre as estruturas matemáticas associadas aos problemas, a partir de levantamento e estudo bibliográficos conjugados a sessões frequentes de discussões com outros professores e alunos da equipe, além de sessões de trabalho individual;

2. elaboração e análise de algoritmos para os problemas estudados, tomando como base os conhecimentos teóricos adquiridos e sua utilização em conjunto com diversos métodos e técnicas de elaboração de algoritmos;

3. desenvolvimento e utilização de ferramentas para implementação eficiente desses algoritmos, com ênfase, sempre que possível, na exploração de paralelismo;

4. delineamento e execução de experimentos computacionais, tanto para avaliar a eficiência dos métodos propostos, como também objetivando extrair desses experimentos direções para a pesquisa teórica;

5. realização de eventos e encontros para maior integração dos pesquisadores e alunos das duas instituições participantes;

6. atividades semestrais com alunos do ensino médio para popularização da ciência;

7. uso intensivo do portal Pargo.ufc.br para divulgação da produção didático-científica do grupo e de outro portal a ser criado para transparência do projeto com linguagem acessível ao público geral;

8. realização de seminários semanais, permitindo a integração entre os temas e uma formação diversificada aos alunos;

9. participação em congressos e cooperações com outras equipes nacionais e internacionais.

6 Cronograma de Atividades

Apresentamos a seguir as atividades planejadas para cada um dos três anos do projeto. Em cada ano, as atividades estão agrupadas como de pesquisa, de divulgação científica e colaborações, e de formação.

6.1 Primeiro Ano

As atividades previstas para o primeiro ano destinam-se basicamente à continuação de trabalhos que já vêm sendo desenvolvidos nos temas do projeto bem como ao início de outros estudos a serem desenvolvidas nos anos seguintes. Especificamente, estão previstos:

- Definição dos subproblemas mais promissores para o ataque no primeiro ano, sendo de particular interesse investigar:
  - Propriedades de coloração parcial de Grundy em produtos de grafos: produto lexicográfico;
  - Complexidade da subfall-coloração em classes de grafos: grafos linha;
  - Caminhos em grafos temporais: definições e complexidade;
  - Polinomialidade em classes de grafos do cálculo dos parâmetros hull number, Caratheodory number e Radon number nas convexidades P_3 e geodésica;
  - Relação entre o Problema *Target Set Selection* e Problemas de Convexidade em Grafos;
  - Complexidade computacional do Jogo de Coloração Gulosa Conexa e do Jogo do Espião
  - Problemas sobre a coloração backbone;
  - Limites superiores para o número de orientação de grafos;
  - Uso de coloração de vértices na resolução do problema de alocação de viaturas;
  - Métodos de solução eficientes para o problema da árvore t-spanner;
- Realização do Ciclo de Seminários do ParGO, com a ocorrência de um seminário semanal;
- Participação de pelo menos 03 membros do grupo em congressos nacionais ou internacionais com apresentação de trabalhos aceitos nos eventos;
- Realização de até 02 missões de visita científica: 01 visita de um membro do grupo a um colaborador nacional ou a recepção de um colaborador nacional, e 01 visita de um membro do grupo a um colaborador internacional ou a recepção de um colaborador internacional nas instalações do grupo.
- 02 atividades de popularização da ciência para alunos do ensino médio com divulgação por meio de fotos/vídeos na página do projeto na Internet

6.2 Segundo Ano

As atividades previstas para o segundo ano são:

- Definição dos subproblemas mais promissores para o ataque no segundo ano, levando-se em conta os resultados obtidos até então. A priori, estaremos interessados principalmente nos seguintes problemas ao longo do segundo ano:
  - Propriedades de coloração parcial de Grundy em produtos de grafos: produto cartesiano;
  - Complexidade da subfall-coloração em classes de grafos: classes evidenciadas pelos estudos no primeiro ano;
  - Parâmetros em grafos temporais e complexidade FPT: levantamento bibliográfico;
  - Limite superior para $BBC(G, T)$ onde G é planar e T é uma árvore;
  - Número de orientação de grafos planares;
  - Formulações de programação linear para o problema de t-spanner em garfos e de mínimo fator de dilatação;
  - Polinomialidade em classes de grafos do cálculo de parâmetros recentes das convexidades P3, geodésica e monofônica, como rank, Helly number e tempo máximo;
  - Algoritmos polinomiais em classes de grafos para o Jogo de Coloração Gulosa Conexa e do Jogo do Espião
  - Extensões de técnicas para obtenção de algoritmos FPT e provas de complexidade parametrizada, bem como extensões de meta-teoremas lógicos que abranjam mais linguagens, procurando consequências para os problemas desse projeto
- Realização do Ciclo de Seminários do ParGO, com a ocorrência de um seminário semanal;
- Participação de pelo menos 03 membros do grupo em congressos nacionais ou internacionais com trabalhos aceitos nos eventos;
- Realização de até 02 missões de visita: 01 visita de um membro do grupo a um colaborador nacional ou a recepção de um colaborador nacional, e 01 visita de um membro do grupo a um colaborador internacional ou a recepção de um colaborador internacional nas instalações do grupo.
- 02 atividades de popularização da ciência para alunos do ensino médio com divulgação por meio de fotos/vídeos na página do projeto na Internet

6.3 Terceiro Ano

As atividades previstas para o terceiro ano são:

- Identificação dos principais subproblemas apresentados no projeto ainda não atacados pelo grupo e definição daqueles mais promissores para ataque no terceiro ano, levando-se em consideração os resultados até então obtidos. A priori, estaremos interessados principalmente nas seguintes questões ao longo do terceiro ano:
  - Escrita dos resultados sobre coloração parcial de Grundy e subfall-coloração;
  - Introdução de novos parâmetros em grafos temporais e aplicação na obtenção de algoritmos FPT para problemas difíceis;
  - Florestas geradoras em grafos rotulados em arestas, miminizando o número de cores ou o número de árvores;

Quantidade de nós	8 nós de computação / 1 nó de administração
Modelo de processador	Intel Xeon E5-2650 V3
Quantidade de processadores por nó	2
Quantidade de núcleos por processador	10
Total de núcleos de processamento	160
Memória por nó	64GB
Total de memória	512GB
Armazenamento por nó	8GB por nó de computação / 24GB no nó de administração
Interconexão	Switch Gigabit Ethernet de 24 portas
Acelearador GPU	1x NVIDIA K40 (no nó de administração)
Acelerador MIC	1x Intel Xeon Phi 7120A (no nó de administração)

Tabela 1: Cluster do LIA

  - Polinomialidade em classes de grafos do cálculo dos parâmetros Radon number, Helly number e rank (posto) nas convexidades P_3, P_3^*, monofônica e triangle-path;
  - Complexidade e Algoritmos polinomiais em classes de grafos para variantes do Jogo de Coloração e do Jogo do Espião
  - Existência de emparelhamento M onde $BBC(G, M)$ não é muito superior ao número cromático de G;
  - Número de orientação de grafos cordais;

- Realização do Ciclo de Seminários do ParGO, com a ocorrência de um seminário semanal;
- Participação de pelo menos 03 membros do grupo em congressos nacionais ou internacionais com trabalhos aceitos nesses eventos;
- Realização de até 02 missões de visita: 01 visita de um membro do grupo a um colaborador nacional ou a recepção de um colaborador nacional, e 01 visita de um membro do grupo a um colaborador internacional ou a recepção de um colaborador internacional nas instalações do grupo.
- 02 atividades de popularização da ciência para alunos do ensino médio com divulgação por meio de fotos/vídeos na página do projeto na Internet

7 Infraestrutura e Apoio Técnico

Na UFC-Campus de Fortaleza, o projeto será desenvolvido junto ao grupo de pesquisa ParGO (Paralelismo, Grafos e Otimização). Para os trabalhos computacionais, o ParGO conta com um laboratório equipado atualmente com 20 microcomputadores, ligados em rede e com acesso à internet. Dispomos também de um cluster de pequeno porte, com as características especificadas na Tabela 1, para os experimentos de maior carga computacional tratados através de processamento paralelo. Esses equipamentos ocupam duas salas de cerca de $20m^2$ cada, pertencentes ao Laboratório de Pesquisa em Computação (LIA/UFC), mas serão movidos no início de 2022 para um espaço maior, recentemente construído, integrando o CIPEMAC (Centro Integrado de Pesquisa em Matemática e Computação). Para a resolução de problemas de programação matemática, temos disponíveis cópias do resolvedor CPLEX para uso institucional. No atual espaço, pertencente ao LIA, há um corpo de técnicos para gerenciamento da rede e das plataformas computacionais utilizadas em experimentos.

Na UNILAB, os pesquisadores tem acesso remoto às instalações do laboratório em Fortaleza e estão atualmente escrevendo um projeto junto à universidade para a construção de um laboratório que sirva de

apoio para todos os grupos de pesquisa. O presente projeto irá reforçar o apoio da universidade com a compra de 03 Desktops e 02 Nobreaks para equipar o referido laboratório.

Esses laboratórios vêm sendo regularmente atualizados e expandidos com verbas de projetos. Parte do orçamento agora solicitado destina-se à continuidade dessas ações, notadamente revitalizar o laboratório do grupo em Fortaleza e melhor equipar o laboratório na UNILAB. Em particular, devido à mudança de espaço mencionada acima, será necessária a adequação das bancadas e outros itens. Além disso, alguns dos computadores atuais apresentam problemas de funcionamento, em virtude do desligamento prolongado por causa da pandemia da COVID-19, precisando de reparos, substituição de peças ou mesmo da máquina em si.

No que diz respeito a material para pesquisa bibliográfica, todos os participantes do projeto têm acesso ao portal de periódicos CAPES e ao sistema de bibliotecas da UFC, que está ligado ao sistema COMUT, além de contarem com uma seleção de livros específicos nos temas no projeto, que compõem um acervo mantido pelo grupo.

Deve-se mencionar ainda a disponibilidade da instituição em infra-estrutura física para abrigar seminários e pesquisadores visitantes, assim como fornecer serviços de secretaria, correios, telefones e faxes.

8 Orçamento Detalhado

8.1 Justificativas

O detalhamento do orçamento está nas tabelas que seguem. Ressaltamos que o mesmo deve ser compartilhado pelos membros da equipe, que inclui 12 pesquisadores doutores, 07 alunos de doutorado e 04 de mestrado, além de darem suporte também a atividades de pesquisa, no tema do projeto, de nossos alunos de iniciação científica.

A seguir apresentamos justificativa para cada item do orçamento.

8.1.1 Capital

Desktops Os recursos solicitados têm a finalidade de permitir manter a infra-estrutura laboratorial em condições adequadas e com permanente atualização das máquinas, além de servirem para equipar a expansão/instalação dos laboratórios dos grupos envolvidos no projeto, com a ocupação do novo espaço no CIPEMAC pelo grupo da UFC e com a construção do laboratório de pesquisa que comportará o grupo da UNILAB (ver Seção 7). São ao todo 10 novos computadores para a UFC e 03 para a UNILAB, com uma configuração "top", a serem adquiridos tão logo sejam disponibilizados os recursos para este fim, e que serão divididos entre os membros da equipe.

Nobreaks Solicitamos nobreaks em número compatível com a quantidade de máquinas pedidas. Os testes computacionais necessários pelas nossas aplicações (grandes instâncias de problemas) consomem dias de processamento e sofrem grandes atrasos com eventuais quedas de energia. Daí a importância que as máquinas onde se realizam tais testes sejam equipadas com Nobreaks.

Notebooks Essenciais para dar maior mobilidade e agilidade aos membros da equipe, em seus deslocamentos intra e inter-campi ou ainda por ocasião de visitas a outras instituições, para a apresentação de seminários e resultados de pesquisa ou mesmo para realização e demonstração de experiências computacionais.

Material bibliográfico O grupo ParGO-UFC mantém um acervo nas áreas em que atua e já possui um acervo robusto de apoio para os alunos e professores. Já a UNILAB ainda não possui material bibliográfico em quantidade e qualidade adequados à execução de suas metas dentro deste projeto. Espera-se suprir em parte essa carência. A grande maioria dos títulos a serem adquiridos são importados, sendo o valor unitário médio de U$ 120,00.

8.1.2 Custeio

Passagens e Diárias Internacionais - Eventos e visitas A participação dos pesquisadores em congressos internacionais qualificados é atividade fundamental para o Grupo pela troca de conhecimentos e experiências com outros grupos, além da promoção de sua visibilidade e inserção dos alunos de pós-graduação na comunidade. Somente participações para apresentação de trabalhos aceitos serão financiadas. Ao longo dos três anos de duração do projeto, pretendemos também reforçar nossas várias cooperações internacionais com visitas aos grupos de pesquisa associados ao ParGO.

Somos 12 docentes e há 10 viagens internacionais previstas, com uma previsão de 4 diárias para cada viagem, ao longo dos 03 anos de execução do projeto. Destaca-se que, mesmo incluindo o período de pandemia, entre 2018 e 2021 os professores da equipe publicaram em cerca de 13 congressos internacionais, como MFCS-2021, COCOON-2021, IWOCA-2021, LATIN-2020, IPEC-2018, ISCO-2018, etc, por vezes com mais de um artigo aceito em um dado congresso. Além disso, foram produzidos, no mesmo período, cerca de 15 artigos com pesquisadores afiliados a instituições estrangerias em periódicos internacionais de renome, como Journal of Combinatorial Theory Series B, Algorithmica, Journal of Graph Theory e Theoretic Computer Science. Desta forma, acreditamos que a previsão de 10 viagens internacionais é até modesta diante do nível de inserção internacional demonstrado.

Passagens e Diárias Nacionais - Eventos O grupo ParGO já possui uma excelente inserção nacional, tendo colaborações com pesquisadores da UFRJ, USP, UFMG, UFRGS, dentre outras, além de participar de comitês científicos e da organização de alguns eventos nacionais, como o ETC (Encontro de Teoria da Computação - CSBC), Simpósio Brasileiro de Pesquisa Operacional (SBPO) e TeoComp-NE (Escola de Teoria da Computação - Nordeste), assim como eventos internacionais que ocorrem esporadicamente no Brasil, como LAWCG (Latin-American Workshop on Cliques in Graphs) e LAGOS (Latin&American Algorithms, Graphs and Optimization Symposium). Os eventos mencionados se tem consolidado como as principais arenas de encontro de professores e alunos da área deste projeto, e os membros da equipe tem estado presentes com publicações nestes eventos em todas suas últimas edições. Desta forma, prevemos a utilização de 05 viagens nacionais, com uma previsão de 4 diárias para cada viagem, para participação em eventos de alunos e professores do grupo.

Serviços de terceiros Aglutinamos, nas Tabelas 2 e 3 que contem o orçamento detalhado, os itens a seguir sob uma única rubrica, chamada "Serviços de terceiros". A razão para esta aglutinação é a dificuldade em prever exatamente quanto será gasto em cada um dos itens, pois estes possuem um carácter dinâmico ou que precisam de orçamento por parte de empresa especializada.

Serviços de terceiros - Material de consumo Aquisição de tonners, papel, pincel e outros materiais de consumo para escritório, bem como peças de reposição tais como fontes, memórias, placas etc.

Serviços de terceiros - Pessoa física Serviços de Manutenção do parque de informática já instalado, instalação de novos equipamentos, pequenos reparos, manutenção de página web, serviços eventuais na administração do laboratório e similares.

Serviços de terceiros - Pessoa Jurídica Serviços de instalação e manutenção do parque de informática já instalado, incluindo "upgrade" e conserto de máquinas.

Serviços de terceiros - Adequação laboratorial Como já mencionado na Seção 7, em 2022 o grupo ParGO-UFC irá ocupar um novo espaço no CIPEMAC e, para tanto, precisaremos adequar o espaço físico com a instalação de bancadas e armários, adequação da rede elétrica e instalação de ar-condicionados.

Serviços de terceiros - Contratação de Máquinas Virtuais para HPC Para laboratórios de pesquisa sem grandes recursos financeiros disponíveis, a aquisição de instâncias de máquinas virtuais de provedores de serviços de infraestrutura em nuvem (IaaS[9]) para execução de cargas de trabalho HPC constitui hoje uma alternativa economicamente mais viável em relação à compra de equipamentos físicos que se prestem a mesma tarefa, tais como clusters e servidores de alta capacidade. Usando serviços desse tipo, tais como Amazon Elastic Compute Cloud (EC2) e Google Cloud Platform (GCP), pesquisadores dispõem do que já de mais avançado em termos de recursos de processamento, incluindo processadores e aceleradores, bem como espaço ode memória e armazenamento. A HPC Shelf, plataforma que propõe-se a ser utilizada neste projeto conforme descrito na Seção 3.4, bem como sua aplicação Swirls, são destinadas justamente a facilitar o acesso a esse tipo de plataforma de computação paralela, muito embora possam também ser usadas para acesso a infraestrutura física do próprio laboratório. Em sua implementação atual, oferece suporte para os serviços EC2 e GCP, podendo ser estendido para outros provedores, bem como clusters de outras instituições de acordo com a necessidade do grupo. Essa possibilidade é especialmente relevante dentro da não existência, nesse momento, de recursos para renovar os recursos de HPC atualmente disponíveis nos laboratórios que servem ao grupo, já necessitando de renovação.

8.1.3 Bolsa

Bolsa de apoio técnico Como já falado na seção anterior e na Seção 7, o grupo ParGO-UFC irá ocupar, em 2022, um espaço no CIPEMAC. Para isso, iremos precisar contratar mão de obra para instalação do novo laboratório (instalação e configuração das máquinas e das impressoras e instalação e configuração da rede).

[9] *Infrastructure-as-a-Service*

8.2 Valores

Orçamento por rubrica.

	Item	Quant.	Valor Unid.	Total
Capital				
	Desktop (UFC)	10	R$ 5.000,00	R$ 50.000,00
	Nobreak (UFC)	5	R$ 800,00	R$ 4.000,00
	Desktop (UNILAB)	3	R$ 5.000,00	R$ 15.000,00
	Nobreak (UNILAB)	2	R$ 800,00	R$ 1.600,00
	Notebooks	4	R$ 6.000,00	R$ 24.000,00
	Livros	10	R$ 630,00	R$ 6.300,00
Total capital				R$ 100.900,00
Custeio				
	Passagem internacional	10	R$ 5.000,00	R$ 50.000,00
	Diária internacional	40	R$ 1.942,50	R$ 77.700,00
	Passagem nacional	5	R$ 1.200,00	R$ 6.000,00
	Diária nacional	20	R$ 320,00	R$ 6.400,00
	Serviços de terceiros		R$ 30.000,00	R$ 30.000,00
Total custeio				R$ 170.100,00
Bolsa				
	Apoio Técnico	7	R$ 550,00	R$ 3.850,00
Total bolsa				R$ 3.850,00
Total do projeto				R$ 274.850,00

Tabela 2: Valores a serem gastos em cada rubrica, designados por itens a serem comprados. O valor unitário do livro foi estimado com base no valor corrente do livro "Introduction to Graph Theory" - Douglas West, que atualmente custa U$ 120,00 e que é um dos livro base para cursos na área. O valor da diária internacional se trata apenas de uma conversão do valor atual da diária (U$ 370,00) para o real. O valor de conversão utilizado foi de R$ 5,25 por dólar.

Orçamento por ano.

	Ano I	Ano II	Ano III
Capital			
Desktop (UFC)	R$ 50.000,00		
Nobreak (UFC)	R$ 4.000,00		
Desktop (UNILAB)	R$ 15.000,00		
Nobreak (UNILAB)	R$ 1.600,00		
Notebook		R$ 12.000,00	R$ 12.000,00
Livros	R$ 6.300,00		
Custeio			
Passagem Internacional	R$ 15.000,00	R$ 20.000,00	R$ 15.000,00
Diárias Internacionais	R$ 23.310,00	R$ 31.080,00	R$ 23.310,00
Passagem Nacional	R$ 2.400,00	R$ 1.200,00	R$ 2.400,00
Diárias Nacionais	R$ 2.560,00	R$ 1.280,00	R$ 2.560,00
Serviços de terceiros	R$ 20.000,00	R$ 5.000,00	R$ 5.000,00
Bolsa			
Apoio Técnico	R$ 1.650,00	R$ 2.200,00	
Total por ano	R$ 141.820,00	R$ 72.760,00	R$ 60.270,00

Tabela 3: Essa tabela contem os valores a serem gastos a cada ano, tomando como base os valores unitários apresentados na Tabela 2.

9 Detalhamento dos Problemas de Pesquisa

Nesta seção, apresentamos de modo formal os problemas e termos matemáticos relativos aos temas de pesquisa. Começamos com algumas definições básicas em Teoria dos Grafos, que modela todos os problemas listados a seguir. Para maiores detalhes, veja [46].

Um *grafo* G é uma tripla formada por um conjunto de *vértices* V, um conjunto de *arestas* E e uma função φ, chamada de *função de incidência*, que associa a cada aresta um par não-ordenado de vértices não necessariamente distintos. Tais vértices são chamados de *extremidades* da aresta e são ditos *vizinhos*. Se uma aresta e do grafo tem como ambas extremidades um mesmo vértice u, então dizemos que e é um *laço*. Se duas arestas e_1, e_2 do grafo têm as mesmas extremidades, então dizemos que e_1 e e_2 são *arestas múltiplas*. Um grafo é *simples* se não possui laços nem arestas múltiplas. Dado um grafo simples G, sejam $V(G)$ e $E(G)$ os seus conjuntos de vértices e arestas, respectivamente. O número de vértices de G é chamado de *ordem* de G e é denotado por $n(G) = |V(G)|$, ou simplesmente n quando G está claro no contexto. O número de arestas de G é o *tamanho* de G e denotado por $m(G) = |E(G)|$, ou somente m. O *grau* de um vértice $v \in V(G)$, denotado por $d_G(v)$, é o número de arestas das quais v é uma extremidade. A *vizinhança* de $v \in V(G)$ em G, denotada por $N_G(v)$, é o conjunto de vértices de vizinhos de v em G e sua *anti-vizinhança* é $\bar{N}_G(u) = V \setminus (N(u) \cup \{u\})$. Se G e H são grafos cujas funções de incidência são φ_G e φ_H, respectivamente, dizemos que H é *subgrafo* de G, denotado por $H \subseteq G$, se $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$ e $\varphi_H \subseteq \varphi_G$. Se G é um grafo simples, então o *complemento* (ou *grafo complementar*) de G é o grafo $\bar{G}$ simples tal que $V(\bar{G}) = V(G)$ e $e \in E(\bar{G})$ se, e somente se, $e \notin E(G)$. Um subconjunto de vértices $S \subseteq V(G)$ é um *conjunto independente*, ou *estável*, se para todos $u, v \in S$ não existe aresta $e = uv$. Dizemos que $S \subseteq V(G)$ é uma *clique* se para todos $u, v \in V(G)$ existe aresta $e = uv$. Denotamos por $\omega(G)$ o tamanho da maior clique e por $\alpha(G)$ é o tamanho do maior conjunto independente de um grafo G.

Um *grafo direcionado* (ou *digrafo*) D é formado por uma tripla consistindo de um conjunto de vértices $V(D)$, um conjunto de *arcos* $A(D)$ e uma função que associa a cada arco um par *ordenado* de vértices não-necessariamente distintos. Para um digrafo D, se (u, v) é o par de vértices associado ao arco $a \in A(D)$, então dizemos que u e v são as *extremidades* de a. Dizemos ainda que a é um arco de u para v, que u é a *cauda* de a, que v é a *cabeça* de a. Seja v um vértice em um digrafo D. O *grau de saída* $d^+_D(v)$ (resp. *grau de entrada* $d^-_D(v)$) é o número de arcos em que v é cauda (resp. v é cabeça). A *vizinhança de saída* (resp. *vizinhança de entrada*) de v é o conjunto $N^+_D(v) = \{u \mid (v, u)$ é associado a um arco de $D\}$ (resp. $N^-_D(v) = \{u \mid (u, v)$ é associado a um arco de $D\}$). Dado um grafo simples $G = (V, E)$, uma *orientação* de G é um digrafo D obtido a partir de G pela substituição de cada aresta de G por exatamente um dos arcos com as mesmas extremidades. Também nos referiremos a D como um *grafo orientado*.

9.1 Coloração de Vértices

Dado um grafo simples $G = (V, E)$, uma *k-coloração própria* de G é uma função $c : V(G) \to \{1, \ldots, k\}$ tal que $c(u) \neq c(v)$ sempre que $uv \in E(G)$. É bem conhecido que os números inteiros $1, \ldots, k$ representam então as *cores* que os vértices de G receberão pela coloração c. O número cromático de G, denotado por $\chi(G)$, é o menor inteiro k para o qual G admite uma k-coloração própria de G. O problema de Coloração de Grafos (em Vértices) consiste em, dado um grafo $G = (V, E)$, determinar $\chi(G)$. Determinar o número cromático de um dado grafo é um problema NP-difícil [138]. Um dos resultados mais conhecidos na literatura é o celebrado Teorema das 4 cores [11]. Deve-se notar que uma k-coloração própria de um grafo G define uma partição do conjunto de vértices de G em k conjuntos independentes formados pelos vértices de mesma cor. Tais conjuntos independentes são também chamados de *classes de cor*.

Devido às suas diversas aplicações práticas e teóricas, diversas variantes foram definidas na literatura. Detalhamos a seguir as principais variantes que serão abordadas neste projeto.

9.1.1 Colorações *Backbone*

Uma das aplicações práticas mais conhecidas de Coloração de Grafos é o problema de alocar frequências a dispositivos em uma rede de forma que dispositivos próximos devam receber frequências diferentes. O objetivo

é o de minimizar a quantidade de frequências utilizadas. Diversas variações do problema de Coloração de Grafos nasceram da adição de restrições a esse problema de atribuição de frequências. A mais comum dessas restrições é a de que as frequências alocadas para dispositivos próximos devem levar em consideração a distância entre eles.

Nesta variante, trataremos exatamente desse tipo de restrição adicional em um subgrafo também dado como entrada. Para os dispositivos que são adjacentes em tal subgrafo, as suas frequências devem estar suficientemente afastadas uma da outra.

Mais formalmente, dado um grafo $G = (V, E)$, note que uma coloração própria c de G é tal que $|c(u) - c(v)| \geq 1$, para toda aresta $uv \in E(G)$. Seja $H = (V, E(H))$ um subgrafo gerador de G, que chamaremos de *backbone* de G. Dizemos que c é uma k-coloração q-*backbone* de (G, H) se c é uma k-coloração própria de G e $|c(u) - c(v)| \geq q$, para toda aresta $uv \in E(H)$. O *número cromático* q-backbone de (G, H), denotado por $\mathrm{BBC}_q(G, H)$, é o menor inteiro k tal que (G, H) admite uma k-coloração q-*backbone* [52].

Observe que, quando $q = 1$, temos $\mathrm{BBC}_q(G, H) = \chi(G)$. Logo, determinar o número cromático q-*backbone* de (G, H) também é um problema NP-difícil, como discutimos no início desta seção. Além disso, se c é uma k-coloração de G, então $c' : V \to \{1, \ldots, q \cdot k - q + 1\}$ definida como $c'(v) = q \cdot c(v) - q + 1$ é uma coloração q-*backbone* de (G, H), para qualquer subgrafo gerador H de G. Veja também que se $H = G$ e $k = \chi(G)$, então c' é uma coloração q-*backbone* ótima de (G, H). Portanto,

$$q \cdot \chi(H) - q + 1 \leq \mathrm{BBC}_q(G, H) \leq q \cdot \chi(G) - q + 1. \tag{1}$$

uma vez que $\mathrm{BBC}_q(H, H) \leq \mathrm{BBC}_q(G, H) \leq \mathrm{BBC}_q(G, G)$.

A noção de coloração *backbone* pode ser ainda generalizada ao se tomar uma métrica circular nas cores. Uma k-coloração q-backbone *circular* c de (G, H) é tal que $q \leq |c(u) - c(v)| \leq k - q$. De modo similar, o *número* q-backbone *circular* de (G, H) é o menor inteiro k tal que (G, H) admite uma k-coloração q-backbone circular.

Veja que uma coloração q-*backbone* circular é uma coloração q-*backbone*. Mais ainda, pode-se obter de uma k-coloração q-*backbone* de (G, H) uma $(k + q - 1)$-coloração q-*backbone* circular de (G, H). Logo,

$$\mathrm{BBC}_q(G, H) \leq \mathrm{CBC}_q(G, H) \leq \mathrm{BBC}_q(G, H) + q - 1. \tag{2}$$

para todo par de grafos (G, H) sendo H um subgrafo gerador de G.

Combinando as Desigualdades (1) e (2), temos

$$q \cdot \chi(H) \leq \mathrm{CBC}_q(G, H) \leq q \cdot \chi(G). \tag{3}$$

Já existem alguns trabalhos na literatura que estudam o número q-*backbone* (circular) em algumas classes de grafos, tanto do ponto de vista computacional, quanto na busca por limitantes superiores [52, 59, 127, 53, 20, 13].

Nesse tema, temos alguns trabalhos na literatura de membros do ParGO. Em [20], Araujo et al. provaram limitantes para o número cromático *backbone* circular de grafos planares com restrições nos ciclos que podem ter como subgrafo e tendo uma árvore como *backbone*. Tais resultados foram motivados pela Conjectura de Steinberg, que foi desprovada [77]. Em [13], Araujo et al. apresentam limitantes para número *backbone* circular de grafos planares quando o *backbone* é uma árvore ou um emparelhamento. Tais limitantes respondem parcialmente a perguntas deixadas em aberto na literatura. Também foi publicado em 2016 um trabalho por Araujo et al. [135] no Journal of Graph Theory generalizando outros oito trabalhos na literatura sobre a existência de uma árvore como *backbone* de um grafo planar G tal que o seu número cromático *backbone* seria limitado a quatro. Foi demonstrado que tal árvore sempre existe. Em um trabalho mais recente [12], apresentam-se diversos limitantes sobre o número cromático *backbone* circular de grafos planares cujo *backbone* é uma galáxia, além da correção de um resultado preliminar encontrado na literatura.

O objetivo primordial nesta variante, é o estudo da mais relevante conjectura na área:

Conjectura 1 ([52])**.** *Se G é planar e $H \subseteq G$ é uma floresta, então* $\mathrm{BBC}_2(G, H) \leq 6$.

Há ainda inúmeras perguntas em aberto na literatura e diversos caminhos para pesquisa. Pretendemos estudar as versões *backbone* de teoremas e conjecturas em Colorações de Grafos que propõem limitantes superiores para o número cromático de um dado grafo, assim como explorar a complexidade da parametrização dual do mesmo.

9.1.2 Orientações Próprias

Esta variante de Coloração é intimamente relacionada com as noções de grafos orientados. Para mais detalhes sobre grafos direcionados e grafos orientados, veja [28].

Uma coloração de um digrafo $D = (V, A)$ nada mais é do que uma coloração do seu grafo subjacente, ou seja, do grafo $G = (V, E)$ obtido a partir de D pela conversão de seus arcos em D (pares ordenados de $A \times A$) em arestas (conjuntos com dois elementos de A). Há interessantes resultados na literatura sobre colorações e orientações. O mais famoso deles é certamente o Teorema de Gallai-Hasse-Roy-Vitaver que implica que qualquer orientação de um grafo k-cromático G, i.e. tal que $\chi(G) = k$, contém um caminho direcionado com k vértices. Os parâmetros que estudaremos são relacionados à noção de orientação própria (ponderada), que é um conceito que novamente interliga as noções de Colorações e Grafos Orientados.

Seja D um grafo orientado. Para cada vértice $v \in V(D)$, o grau de entrada de v em D, denotado por $d^-_D(v)$, já foi definido no começo da seção como a quantidade de arcos com sumidouro v. No caso de D ser ponderado em seus arcos, ou seja, no caso de haver uma função $w : A(D) \to \mathbb{Z}^*_+$, dizemos que o *grau de entrada ponderado* de $v \in V(D)$ com respeito ao par (D, w) é a soma dos pesos $w(a)$ dos arcos $a \in A(D)$ que têm v como sumidouro, denotado por $d^-_{(D,w)}(v)$.

Para $k \in \mathbb{Z}^*_+$, uma *k-orientação própria* de um grafo simples G é uma orientação D de G de modo que vértices adjacentes em G tem graus de entrada distintos em D e que o maior grau de entrada de um vértice em D é limitado superiormente por k. De modo análogo, uma *k-orientação própria ponderada* de um par (G, w), onde G é um grafo simples e $w : E(G) \to \mathbb{Z}^*_+$, é uma orientação D de G de modo que vértices adjacentes em G tem graus de entrada ponderados distintos em (D, w') (um arco $a \in A(D)$ correspondendo à orientação de uma aresta $e \in E(G)$ recebe o mesmo peso dessa aresta, ou seja $w'(a) = w(e)$) e que o maior grau de entrada de um vértice em D é no máximo k.

Deve-se notar que uma k-orientação própria define uma $(k+1)$-coloração própria já que pode-se considerar grau de entrada atribuído a cada vértice como uma cor (a diferença de uma unidade se deve ao fato de o menor grau de entrada poder ser igual à zero, enquanto que a definição de coloração própria tem como menor cor o número um).

Diversos parâmetros foram definidos na literatura a respeito dessas noções. Começamos detalhando o primeiro, e mais estudado, deles.

Número de orientação própria. O *número de orientação própria* de um grafo G, denotado por $\overrightarrow{\chi}(G)$, é o menor inteiro k tal que G possui uma k-orientação própria [4]. Como toda k-orientação própria define uma $(k+1)$-coloração própria, note que $\omega(G) \leq \chi(G) \leq \overrightarrow{\chi}(G) + 1$, onde $\omega(G)$ é o tamanho da maior clique de G. Além disso, é bem conhecido que $\chi(G) \leq \Delta(G) + 1$ e pode-se facilmente mostrar por indução que $\overrightarrow{\chi}(G) \leq \Delta(G)$, sendo $\Delta(G)$ o maior grau de um vértice de G.

Ahadi e Dehgan [4] introduziram este parâmetro e mostraram um algoritmo polinomial para calcular o número de orientação própria de um grafo 3-regular e que determinar se esse parâmetro vale 3 ou 4 em um grafo 4-regular já é um problema NP-completo. Além disso, eles mostram que é NP-completo decidir se $\overrightarrow{\chi}(G) \leq 2$ para um grafo planar.

Araujo et al. [17] demonstraram que se T é uma árvore, então $\overrightarrow{\chi}(T) \leq 4$ e que se G é um grafo bipartido geral, então:

$$\overrightarrow{\chi}(G) \leq \left(\Delta(G) + \sqrt{\Delta(G)}\right)/2 + 1.$$

Do ponto de vista de Complexidade Computacional, eles melhoraram o resultado de Ahadi e Deghan mostrando que ainda é NP-completo decidir se $\overrightarrow{\chi}(G) \leq 2$ para um grafo planar G mesmo que G tenha grau máximo 3 e que é NP-completo decidir se $\overrightarrow{\chi}(G) \leq 3$ para um grafo bipartido planar com grau máximo 5 [17].

Também mostraram que é NP-completo decidir se $\overrightarrow{\chi}(G) = \Delta - 1$ ou $\overrightarrow{\chi}(G) = \Delta$ para qualquer valor de $\Delta = \Delta(G) \geq 3$.

Araujo et al. [19] demonstraram que $\overrightarrow{\chi}(G) \leq 7$ para qualquer grafo cactus G, e que este limitante pode ser atingido. Knox et al. [146] apresentaram uma demonstração mais simples de que $\overrightarrow{\chi}(T) \leq 4$, para toda árvore T, e apresentaram limitantes superiores para subclasses de grafos planares bipartidos.

Mais recentemente, houve considerável interesse da comunidade sobre o tema e diversos artigos relacionados foram publicados e novos parâmetros foram definidos.

Considerando ainda exclusivamente o número de orientação própria (não ponderado), em [7], os autores apresentam limitantes superiores para o número de orientação própria para grafos periplanares livres de triângulos e sem pontes. Em [171], mostra-se que todo grafo planar bipartido com grau mínimo 3 admite uma 3-orientação própria. Ainda este ano, Araujo et al. [16] mostram uma redução para mostrar que determinar o número de orientação própria para grafos cordais é NP-difícil, mas que admite um algoritmo FPT quando parametrizado pelo valor da solução. Além disso, argumentamos que tal problema não admite *kernel* polinomial, sob a hipótese que $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$. Ainda do ponto de vista da Complexidade Parametrizada, mostramos algoritmos de *kernelização* para as classes de grafos split e de cobipartidos, apesar de não sabermos ainda se o problema permanece NP-difícil para essas classes. Também mostramos limitantes superiores apertados para a classe de grafos split e de grafos bloco uniformes. Mostramos limitantes superiores para a classes de grafos planares que não possuem vértice de grau máximo adjacentes, e para a classe de grafos periplanares maximais 2-conexos. Também mostramos limitantes superiores para as classes de grafos threshold e cografos.

Com respeito ao número de orientação própria, diversas perguntas permanecem em aberto e as abordaremos nos próximos anos: esse parâmetro é limitado por alguma constante quando restrito a classe de grafos (peri)planares? Qual a complexidade de determinação deste quando restrito a subclasses como a dos grafos split, cobipartidos, cografos? Há alguma função que dependa do número cromático e do tamanho da maior clique que limite superiormente este parâmetro?

Número de orientação própria acíclico. Uma outra pergunta que nos intriga e leva à definição de um novo parâmetro é a seguinte: e se restringirmos nosso espaço de busca a orientações do grafo G dado como entrada que sejam acíclicas? Obviamente, um parâmetro associado, que já sabemos ser bem definido, serviria como limitante superior para o número de orientação própria. Já verificamos que diversos resultados supra-mencionados se aplicam a esse caso particular. A questão é se tal restrição pode ajudar na obtenção de novos limitantes superiores para o número de orientação própria. Isso será objeto de estudo desse projeto.

A seguir, detalhamos outros parâmetros relacionados que foram definidos recentemente na literatura. Como há poucos trabalhos a respeito de cada um deles, nos limitaremos a listar as contribuições existentes já que há muito o que ser abordado em cada um deles e não faz sentido mencionar algumas poucas questões em aberto.

Número de orientação entrada-saída-própria. Ainda no contexto não ponderado, este ano um novo parâmetro foi definido: *in-out proper orientation number* [93]. Nesse caso, dado um grafo simples G e uma orientação D de G, o *grau-entrada-saída* de um vértice $v \in V(D)$ é a diferença, em módulo, do grau de entrada menos o grau de saída de v em D. Uma orientação D de G é dita *k-orientação entrada-saída-própria* se vértices adjacentes em G possuem grau-entrada-saída distintos em D e se o grau-entrada-saída de cada vértice não excede k. O número de orientação entrada-saída-própria é o menor inteiro k tal que G admite uma k-orientação entrada-saída-própria. No que acreditamos ser o único trabalho no tema, Dehghan [93] mostra que tal parâmetro é no máximo 3 para árvores e exibe um algoritmo para decidir se o mesmo é no máximo 2 para grafos subcúbicos. Além disso, é demonstrado que determinar se o mesmo é no máximo 1 é NP-completo, mesmo para grafos bipartidos de grau máximo 3, assim como algumas contribuições para grafos regulares são apresentadas.

Número de orientação própria ponderado. Em 2019, Araujo et al. [23] definiram uma versão ponderada do problema. Dado um grafo simples G e uma função de pesos $w : E(G) \to \mathbb{Z}_+^*$, o número de orientação

própria ponderado de (G, w) é o menor inteiro positivo k tal que G admite uma k-orientação própria ponderada, denotado por $\overrightarrow{\chi}(G, w)$. Os autores demonstram que a determinação desse parâmetro em árvores é (fracamente) NP-difícil e que admite um algoritmo pseudo-polinomial. No caso mais geral de grafos com largura em árvore limitada é demonstrado que decidir se $\overrightarrow{\chi}(G, w) \leq k$ se G tem largura em árvore igual a t pode ser feito em tempo $\mathcal{O}(2^{t^2}k^{3t}tn)$ e tal resultado é complementado por uma demonstração que este problema quando parametrizado exclusivamente pela largura em árvore é W[1]-difícil.

Número de orientação semi-própria. Em outro artigo publicado este ano [94], uma outra versão ponderada foi definida. Dado apenas um grafo simples G, o número de orientação semi-própria de G é o menor inteiro k tal que (G, w) admite uma k-orientação própria ponderada, dentre todas as possíveis funções $w : E(G) \to \mathbb{Z}_+^*$. Os autores mostram que apesar de estarmos livres para tomarmos qualquer função de pesos, sempre há uma que leva a uma solução ótima cuja imagem é restrita a $\{1, 2\}$. Eles também demonstram que este parâmetro admite limitantes superiores em função do maior grau médio de G, da *degeneracy* de G, além de mostrar limitantes superiores constantes apertados para árvores, cactus, periplanares e planares. Eles também mostram que a determinação deste parâmetro é um problema NP-difícil. Resultados parciais foram obtidos em paralelo em [125].

Rotulação universal. Uma terceira versão ponderada relacionada foi definida na literatura ainda em 2017 [5]. Nesse caso, o que se recebe como entrada é também apenas um grafo simples G, mas o objetivo é determinar uma função de pesos, chamada de *rotulação universal*, $w : E(G) \to \mathbb{Z}_+^*$ tal que *toda* orientação ponderada (D, w) seja própria. Os autores conjectura que todo grafo que admite tal função de pesos, sempre possui uma ponderação em que os pesos sejam polinomiais em $n(G)$. Eles demonstram tal afirmação para o caso particular de árvores. Mostram que o parâmetro associado é NP-difícil de ser determinado, além de uma relação entre este parâmetro e o número cromático em arestas do grafo dado. Eles também apresentam resultados probabilísticos que indicam que a conjectura de fato deve ser verdadeira.

9.1.3 Variantes baseadas em heurísticas

Na literatura e na prática, há ao menos três heurísticas para obtenção de colorações de grafos bastante estudadas e utilizadas que são as heurísticas "a", "b" e "gulosa". O estudo da performance de cada leva à definição de um parâmetro de coloração distinto.

Heurística a**:** Se existem duas classes de cores X e Y tais que $X \cup Y$ é um conjunto independente, podemos colorir os vértices de Y com a cor da classe X, eliminando, dessa forma, uma classe de cor.

Heurística b**:** Se existe uma classe de cor X tal que cada vértice na classe possui pelo menos uma cor (diferente da sua) que não aparece em sua vizinhança, podemos recolorir todos os vértices em X, eliminando portanto uma classe de cor.

Heurística *gulosa***:** Dados um grafo $G = (V, E)$ e uma ordem $\langle v_1, \ldots, v_n \rangle$ no conjunto $V(G)$, no iteração i, para todo $1 \leq i \leq n$, atribua ao vértice v_i a menor cor $j \in \{1, 2, \ldots, n\}$ tal que nenhum vizinho de v_i no subconjunto $\{v_1, \ldots, v_{i-1}\}$ tenha recebido a cor j em iterações anteriores à i.

As heurísticas acima motivaram as seguintes definições:

Definição 1 (a-Coloração)**.** *Uma coloração $c : V \to \{1, \ldots, k\}$ de um grafo $G = (V, E)$ é uma "a-coloração" se, para quaisquer i, j, $i \neq j$, $1 \leq i, j \leq k$, existem vértices u, v tais que $c(u) = i$, $c(v) = j$ e $(u, v) \in E$.*
Definição 2 (b-Coloração)**.** *Uma coloração $c : V \to \{1, \ldots, k\}$ de um grafo $G = (V, E)$ é uma "b-coloração" se, para todo i, $1 \leq i \leq k$, existe $u \in V$ tal que $c(u) = i$ e, para todo j, $1 \leq j \leq k$, existe$(u, v) \in E$ com $c(v) = j$ (dizemos que u é um b-vértice da cor i).*

Definição 3 (Coloração Gulosa)**.** *Uma coloração $c : V \to \{1, \ldots, k\}$ de um grafo $G = (V, E)$ é uma "coloração gulosa" se, para todo $1 \leq i \leq k$, todos os vértices $v \in V(G)$ de cor $c(v) = i$ tiverem ao menos um vizinho colorido com cada cor $j \in \{1, \ldots, k-1\}$.*

O pior desempenho dessas estratégias pode ser medido pelos número a-cromático, b-cromático e número guloso de um grafo, respectivamente, cujas definições seguem.

Definição 4 (Número a-cromático)**.** *Denotado por $\chi_a(G)$, é o maior valor k para o qual G possui uma a-coloração com k cores.*

O problema de determinar se $\chi_a(G) \leq k$ para um dado grafo G e um valor k é NP-completo, mesmo quando G é uma árvore.

Definição 5 (Número b-cromático)**.** *Denotado por $\chi_b(G)$, é o maior valor k para o qual G possui uma b-coloração com k cores.*

Definição 6 (Número guloso)**.** *Denotado por $\Gamma(G)$, é a maior quantidade de cores utilizadas em uma coloração gulosa, tomadas todas as possíveis ordenações de $V(G)$.*

Na literatura, também encontramos referências a esse valor como *número cromático first-fit* (neste caso, comumente denotado por $\chi_{FF}(G)$) ou *número de Grundy*.

A literatura a respeito desses parâmetros é muito extensa e por isso não a incluímos aqui. O ParGO possui um longo e extremamente relevante histórico de contribuições sobre o estudos desses parâmetros, sobretudo com relação ao número guloso e ao número b-cromático (vide `https://pargo.ufc.br/pt/publicacoes/`).

Neste projeto, além de continuarmos os estudos nas variantes supra-mencionadas, focaremos também em outras variantes mais recentemente definidas na literatura:

Coloração Gulosa Parcial. Uma coloração é dita "gulosa parcial" se cada classe de cor possui ao menos um vértice colorido de maneira gulosa. Mais formalmente, $c : V \to \{1, \ldots, k\}$ é gulosa parcial se para cada $i \in \{2, \ldots, k\}$, existe u de cor i tal que u possui algum vizinho de cor j, para toda cor $j < i$.

Coloração Gulosa Conexa. Uma coloração gulosa também pode ser definida como sendo uma coloração obtida aplicando o algoritmo guloso sobre uma determinada ordem dos vértices do grafo. A coloração é chamada de "gulosa conexa" quando a ordem é conexa, isto é, quando todo vértice da ordem, com exceção do primeiro, possui um vizinho que ocorre antes dele na ordem.

***Fall* Coloração.** Uma coloração é "fall" se todos os vértices são b-vértices.

***Subfall* Coloração.** Uma subfall coloração de G é qualquer fall coloração de um subgrafo induzido de G.

9.2 Conectividade

Problemas computacionais de estabelecer conexões entre dois ou mais vértices de um grafo são bastante comuns, como o problema do menor caminho, da árvore geradora mínima, da árvore de Steiner, do fluxo máximo, entre outros. Muitos desses problemas clássicos que possuem algoritmos exatos polinomiais se tornam difíceis computacionalmente quando surgem restrições adicionais ao problema. Neste projeto, pretende-se investigar alguns problemas dessa natureza, listados a seguir.

9.2.1 Grafos Temporais

Um grafo temporal $\mathcal{G}$ é uma tripla (G, γ, λ) onde G é um grafo e γ, λ são funções sobre $V(G)$ e $E(G)$, respectivamente, que dizem quando os vértices e arestas de G estão ativos. Mais formalmente, para cada $v \in V(G)$ temos que $\gamma(v) \subseteq \mathbb{N}$, e para cada aresta $e \in E(G)$ temos que $\lambda(e) \subseteq \mathbb{N} \times \mathbb{N}$. Ademais, se $(t, t') \in \lambda(uv)$, então $t \leq t'$, $t \in \gamma(u)$ e $t' \in \gamma(v)$. Consideramos somente grafos temporais finitos, i.e., $T = \max \bigcup_{v \in V(G)} \gamma(v)$ é definido e é chamado de *tempo de vida de $\mathcal{G}$*. Chamamos também G de *grafo base de $\mathcal{G}$*.

Grafos temporais tem sido objeto de muito interesse da comunidade na última década (veja por exemplo os levantamentos bibliográficos [166, 131, 148]). Isto se deve principalmente ao fato que muitos problemas da vida real possuem uma componente dinâmica envolvida. Por exemplo, a troca de mensagem entre pessoas em uma rede social, o escalonamento de linhas de ônibus em uma rede de transporte público, a contaminação de pessoas por uma doença infecciosa, etc.

Um aspecto interessante dos grafos temporais é que, a depender da interpretação dada a uma propriedade, tal propriedade pode valer ou não. Por exemplo, o famoso Teorema de Menger não é válido quando o corte de vértices é considerado no grafo base de G [32, 140], mas passa a valer quando o corte consiste de pares (vértice,tempo) [165].

Tendo em vista que a base teórica para estas estruturas ainda está se consolidando, e que a maior parte dos trabalhos existentes analisam problemas apenas pelo ponto de vista de aplicações, vemos que há uma grande importância em se investigar problemas mais básicos nestas estruturas, em particular problemas que envolvem conectividade. De fato, tais problemas estão dentre aqueles que tem chamado mais a atenção da comunidade até então, mesmo em se considerando trabalhos mais práticos. Propomos então a investigação dos resultados existentes relacionados a conectividade de grafos temporais com o intuito de identificar possíveis lacunas do conhecimento, para então tentar preenchê-las. Destaca-se que a Profa. Ana Shirley Silva já possui três trabalhos em congresso e dois artigos aceitos para publicação sobre problemas em grafos temporais [60, 61, 134, 163, 164]. Em quase todos estes trabalhos foram investigados problemas relacionados a conectividade, com exceção de [163], que trata de um problema de coloração.

Pretendemos trabalhar nos seguintes aspectos, sempre buscando preencher lacunas e consolidar uma base teórica de grafos temporais:

Definições e notação. Mesmo que grafos temporais tenham sido estudados há bastante tempo, com os primeiros artigos datando ainda da década de 90 [198], uma maior atenção tem sido dada somente na última década, principalmente no que diz respeito a trabalhos puramente teóricos. Devido a isto, a comunidade ainda não convergiu para uma notação comum. Muitos autores tem feito um esforço para unificar a linguagem, mas parece haver ainda muita divergência de opiniões sobre a melhor forma de denotar estas estruturas. Nós pretendemos tentar agir nesta consolidação, reforçando melhores linguagens e um maior formalismo.

Caminhos entre pares de vértices. Ao considerar um caminho entre um par de vértices, diferentes conceitos de minimalidade ocorrem no contexto temporal. Por exemplo, pode-se estar interessado naquele que chega mais cedo, ou que sai mais tarde, ou que usa a menor quantidade de arestas, ou ainda que demanda o menor tempo total para ser percorrido [54, 200, 107]. Além disso, quando vértices podem se tornar inativos, é preciso também garantir que a espera em um vértice só possa ser feita caso tal vértice esteja ativo [63]. Estes problemas estão entre os poucos problemas que permanecem polinomiais no contexto temporal.

Árvores e componentes conexas. Definições de árvores temporais [148] e árvore temporal geradora (mínima) [132] foram propostas e investigadas, e em geral consiste em garantir que exista um caminho válido entre a raiz para os demais vértices. Em particular, em [132] os autores apresentam um algoritmo linear que computa uma árvore geradora temporal que alcança todos os vértices em tempo mínimo, e provam que o problema se torna NP-difícil caso sejam considerados pesos.

Com relação a componentes conexas, em [34] os autores apresentam duas possíveis definições de componentes fortemente conexas no contexto temporal e provam que tais definições possuem diferentes complexidades, sendo uma computável em tempo polinomial e outra NP-difícil. O problema também foi estudado em [170], onde os autores dão outros resultados de NP-dificuldade.

9.2.2 Grafos rotulados em arestas

Um grafo simples e não direcionado é dito colorido/rotulado em arestas (GCA) quando é da forma $G = (V, E, L, l)$ onde V é o conjunto de vértices, E o conjunto de arestas, L um conjunto de rótulos e $l : E \to L$ é uma função de coloração das arestas de G, que associa um rótulo para cada aresta de G.

Dados um GCA e um inteiro positivo k, o problema da Floresta Geradora k-rotulada (FGkR) consiste em encontrar uma floresta geradora com a menor quantidade de árvores (componentes) e número de cores (rótulos) distintas menor ou igual a k [64]. Já o problema da Floresta Geradora Rotulada com k componentes (FGRkC) consiste em determinar uma floresta geradora com até k componentes e menor número de rótulos. Ambos são generalizações do problema da Árvore Geradora Minimamente Rotulada (MLSTP - Minimum Labeling Spanning Tree Problem), cujo objetivo é encontrar uma árvore geradora com a menor quantidade de cores de um GCA. Os três problemas são NP-Difíceis [66, 64]. Neste projeto, concentramos nossa atenção nos dois problemas mais gerais, FGkR e FGRkC, ainda pouco explorados na literatura.

Os primeiros trabalhos sobre FGkR apresentam meta-heurísticas para resolver o problema, adaptando algoritmos similares desenvolvidos para MLSTP [64, 78, 79]. Dentre as meta-heurísticas utilizadas encontram-se as bem conhecidas Arrefecimento Simulado, Algoritmo Genético, GRASP e Busca em Vizinhança Variável (VNS). Além delas, foram também avaliadas nesses trabalhos as variantes do VNS chamadas Intel-VNS e Co-VNS (propostas em [80] para o MLSTP) e Método Piloto (ver descrição dessa meta-heurística em [197]). Com respeito a métodos exatos, ao nosso conhecimento, há apenas dois trabalhos relacionados a FGkR: [64], onde é proposto um algoritmo enumerativo, usando *backtracking*, e [112], onde são apresentadas algumas formulações de programação inteira baseadas em diferentes caracterizações do problema e reportados alguns experimentos computacionais com os modelos. Neste último trabalho, orientado por pesquisador da equipe, mostrou-se também que é possível realizar diferentes operações sobre o grafo de entrada para gerar uma instância equivalente.

Neste projeto, pretendemos dar continuidade ao trabalho em [112], fortalecendo as formulações e aplicando métodos de decomposição para resolvê-las. É também planejado explorar nesses métodos de solução as operações de conversão de instâncias mostradas em [112], no intuito de gerar procedimentos mais eficientes.

9.2.3 t-Spanners

Dados um grafo conexo G, ponderado em arestas, e um real $t \geq 1$, um t-spanner (tS) de G é um subgrafo gerador H tal que, para todo par $\{i, j\}$, com $i, j \in V(G)$, a distância entre i e j em H é no máximo t vezes a distância no grafo original. O parâmetro t é denominado de fator de dilatação. De maneira formal, um subgrafo gerador H é um t-spanner de G se:

$$\text{dist}_H(i, j) \leq t * \text{dist}_G(i, j), \qquad \forall i, j \in V(G) \tag{4}$$

onde $\text{dist}(i, j)$ é o menor peso de um i, j-caminho no grafo considerado. Se além de subgrafo gerador, for acíclico, portanto uma árvore geradora, dizemos que H é uma árvore t-spanner. t-Spanners descrevem contextos onde se deseja "aproximar" o grafo original por subgrafos esparsos, mantendo controle sobre o aumento da distância vértice-a-vértice [180]. Dessa forma, encontram aplicações em áreas diversas, como redes de comunicação, sistemas distribuídos, robótica, projeto de redes, geometria computacional e bioinformática [57, 58, 173, 179, 181].

Nesse contexto, dada uma tripla (G, w, t), onde $G = (V, E)$ é um grafo simples, $w : E \to \mathbb{R}^+$ é uma função de ponderação das arestas e $t \geq 1$ é um número real (fator de dilatação), alguns problemas de otimização aparecem. No t-spanner de custo mínimo (tSCM), queremos determinar um tS em G de peso mínimo, ou seja, tal que a soma dos pesos de suas arestas seja mínimo. Na árvore t-spanner de custo mínimo (AtSCM), desejamos determinar, se existir, um tS de G que seja acíclico e que tenha o menor peso possível. Quando os pesos são unitários, AtSCM se torna um problema de decisão, a ser referenciado simplesmente como AtS, visto que toda árvore geradora, em particular uma árvore t-spanner, terá peso $|V| - 1$. Finalmente, outro problema de otimização, denominado Árvore t-spanner com menor fator de dilatação (ASMFD), pode ser definido quando apenas o par (G, w) é dado como entrada e desejamos determinar o menor real t tal que G admite uma árvore t-spanner. Considere $m = |E(G)|$ e $n = |V(G)|$.

Sabe-se que ASMFD é NP-difícil em grafos panares [110] e inaproximável com fator $2-\epsilon$, $\epsilon > 0$, em grafos arbitrários [118]. Por outro lado, o caso não ponderado de ASMFD é aproximável com fator $O(\log |V|)$ [103]. Quanto a tSCM, é NP-Difícil para $t \geq 2$ [56, 180], porém $O(n^{\frac{2+\epsilon}{t-1}})$ aproximável para $t > 1$, $\epsilon > 0$. No caso não

ponderado, é $O(\log m/n)$ e $O(n^{\frac{2}{t-1}})$ aproximável, para $t = 2$ e t≥ 3, respectivamente, enquanto tem fatores de inaproximabilidade $c \log n$ inaproximável ($c < 1$) e $2^{\frac{\log^{1-\epsilon} n}{t}}$ ($\epsilon > 0$), para $t = 2$ e $t \geq 3$, respectivamente.

Os resultados de complexidade sobre o caso de árvores são mais abundantes. Primeiro, vale ressaltar que nesse caso pode-se restringir t aos inteiros [58]. Em geral, AtS é NP-completo para $t \geq 4$. Quando $t = 1$ o problema se torna trivial e, para $t = 2$ também é polinomial [56, 58]. Para o caso $t = 3$, a complexidade ainda está em aberto para grafos arbitrários. A Tabela 4, retirada de [48], apresenta a complexidade de AtS para algumas classes específicas de grafos, na qual os casos em aberto estão identificados por ?. No caso de pesos arbitrários, é NP-Completo verificar se AtSCM é viável, para cada $t > 1$, porém quando $t = 1$ uma solução ótima pode ser obtida em tempo em $O(m \log \beta(m, n))$, onde $\beta(m, n) = \min\{i | \log^{(i)} n \leq \frac{m}{n}\}$ [58].

Tabela 4: Complexidade de AtS

Tipos de Grafos	t=3	t
cografo, split, complemento de bipartido	P [56]	
intervalo, permutação, bipartido regular	P [157]	
planar	P [110]	
exoplanar		P ($t \geq 1$) [169]
bipartido convexo	P [195]	
bipartido	? [157]	NP-c ($t \geq 5$) [51]
grafo de cocomparabilidade	? [157]	
grau máximo $\leq b * \log n, b > 0$	P [176]	
fortemente cordal	P [50]	P ($t \geq 4$) [49]
1-split	P [50]	
cordal com diâmetro≤ 2	P [50]	
cordal com diâmetro$\leq t + 1$ (para t par)		NP-c ($t \geq 4$) [50]
cordal com diâmetro$\leq t + 2$ (para $t$ímpar)		NP-c ($t \geq 4$) [50]
grafo com diâmetro≤ 5	P [175]	
cordal	? [157]	NP-c ($t \geq 4$) [50]
grafos que não possuem K_6 como menor		NP-c ($t \geq 4$) [99]
P (Polinomial); NP-c (NP-completo)		

Pretendemos abordar esses problemas, especialmente o caso de árvores, via programação matemática. Nessa linha, existem pouquíssimos trabalhos [6]. Destacamos uma tese de doutorado [48] sobre tSCM e AtSCM, uma dissertação de mestrado [191] sobre ASMFD, e uma segunda dissertação de mestrado sobre AtS e AtSC, esta última orientada por um pesquisador deste projeto. Esses trabalhos apresentam algumas formulações de programação linear-inteira para os casos abordados. Alguns experimentos computacionais são apresentados em [48, 191], mostrando que as formulações precisam ser aprimoradas para tratar instâncias de médio porte. Neste projeto, nosso objetivo é propor métodos de solução baseados nessas formulações, usando desigualdades válidas e estratégias de decomposição.

9.2.4 Fluxos Arco-Disjuntos

Seja $D = (V, A)$ um digrafo. Uma *rede* $\mathcal{N} = (D, u)$ é formada por $D = (V, A)$ e uma *função de capacidade* $u : A(D) \to \mathbb{Z}_+$. Se todos os arcos em D tem capacidade λ, escrevemos simplesmente $u \equiv \lambda$. Um *fluxo* em $\mathcal{N}$ é uma função $f : A(D) \to \mathbb{Z}_+$ tal que $f(vw) \leq u(vw)$, $\forall vw \in A(D)$. O *vetor de balanço* de um fluxo f em uma rede $\mathcal{N} = (D, u)$ é a função $b_f : V(D) \to \mathbb{Z}$ que associa cada vértice $v \in V(D)$ ao valor $\sum_{vw \in A} f(vw) - \sum_{wv \in A} f(wv)$.

Fluxos são amplamente estudados uma vez que permitem, com certa elegância e simplicidade, modelar problemas em diferentes áreas de estudo como transporte, logística e telecomunicações. No campo teórico, eles são usados para resolver vários problemas em grafos e digrafos. A possibilidade de considerar a existência

de fluxos simultâneos em uma rede dá ainda mais poder de modelagem para esta ferramenta.

Dois fluxos f e g em uma rede $\mathcal{N} = (D, u)$ são arco-disjuntos se $f(vw) \cdot g(vw) = 0$, $\forall vw \in A(D)$. O problema de encontrar fluxos arco-disjuntos foi introduzido e estudado em [27]. Os autores consideraram diferentes restrições mostrando que algumas delas generalizam problema importantes, conhecidamente difíceis, como o problema de encontrar caminhos arco-disjuntos em um grafo direcionado. Entre os resultados de dificuldade, eles mostraram que o seguinte problema é NP-completo: decidir se existem dois fluxos arco-disjuntos em uma rede em que todos os arcos tem capacidade no máximo dois.

Dizemos que um digrafo D é um *s-branching* se existe um caminho direcionado de s para todos os outros vértices de D e grafo subjacente de D é uma árvore. Um clássico resultado de J. Edmonds [102] caracteriza os digrafos contendo k s-branchings arco-disjuntos.

Teorema 2 ([102])**.** *Um digrafo $D = (V, A)$ possui k s-branchings arco-disjuntos se e somente se $d^-_D(X) \geq k$ para todo $X \subseteq V(D) \setminus \{s\}$ com $X \neq \emptyset$.*

Um *fluxo s-branching* f em $\mathcal{N} = (D, u)$ é tal que $b_f(s) = n-1$ e $b_f(v) = -1$ para todo $v \in V(D)\setminus\{s\}$. Em outras palavras, f alcança todos os vértices de D e cada vértice diferente de s *retém* uma unidade de fluxo. Podemos reduzir o problema de encontrar um fluxo s-branching em uma dada rede para o de encontrar um (s, t)-fluxo. Também é possível encontrar k fluxos s-branching arco-disjuntos em tempo polinomial quando $u \equiv n-1$: em [27] foi mostrado que, neste caso, $\mathcal{N}$ admite um fluxo s-branching se e somente se D contém um s-branching. Então, aplicando o Teorema 2, os autores forneceram uma caracterização de redes admitindo k fluxos s-branching arco-disjuntos. Eles também apresentaram um algoritmo de tempo polinomial que encontra tais fluxos se eles existem.

A tratabilidade deste problema depende, em geral, dos valores da função de capacidade. Em [29], foi mostrado que o problema de decidir se uma rede possui k fluxos s-branching arco-disjuntos é NP-completo se todo arco tem capacidade ℓ, para todo $\ell \geq 2$ fixo. Por outro lado, no mesmo artigo provou-se que o problema pode ser resolvido em tempo polinomial quando $\lambda = n - c$, para $c \geq 1$ fixo. Foi mostrado recentemente que esse problema é FPT com parâmetro c [33].

Neste trabalho, gostaríamos de caracterizar redes admitindo k fluxos s-branching arcos disjuntos, através de uma condição como a do Teorema de Edmonds. Uma propriedade inicial está sendo investigada [62], mas já sabemos que uma condição mais forte do que a definida nesse trabalho é necessária.

Outra direção para esta pesquisa seria a investigação de algumas questões interessantes de complexidade relativas a fluxos s-branching arco-disjuntos que permanecem em aberto, como o caso em que todas os arcos de $\mathcal{N} = (D, u)$ possuem capacidade $(\log n)$ e a determinação de uma dicotomia entre os casos tratáveis e intratáveis quando o digrafo D recebido como entrada é um DAG.

9.2.5 Projeto de redes de sensores clusterizadas

Redes de sensores são usadas para monitorar eventos em uma dada área geográfica. Vários sensores são instalados de forma dispersa cobrindo uma área a ser monitorada. Alguns sensores ficam em permanente estado de atividade (acordados), enquanto outros podem ficar em estado de inatividade (dormindo) até sensorearem algo. Ocasião em que devem enviar os dados coletados para uma unidade central responsável por processar toda a informação da rede que deve estar conectada. Uma forma de aumentar o tempo de vida da rede (reduzir o consumo de energia para a comunicação de dados), é eliminar o envio de dados repetidos para o ponto central da rede. Equipar sensores com essa capacidade tem um custo adicional. Sensores com essa função são chamados de mestres (cabeça do cluster). Os demais sensores no raio de alcance de um mestre ou que para eles enviam seus dados formam o cluster. Uma rede clusterizada é formada por vários clusteres, sendo que clusteres vizinhos devem compartilhar um sensor em comum (ponte) para fazer a comunicação de dados entre seus mestres. Apenas mestres coletam e eliminam dados repetidos quando dois ou mais sensores do cluster detectam o mesmo evento. Por serem mais caros, devemos dispor mestres de forma que não estejam no raio de cobertura um do outro. Uma questão interessante é saber a qual distância, em saltos, dispor os sensores de um cluster de seu mestre. Podemos usar vários saltos (em número de links, arestas) entre sensor e mestre [196]. Porém, quanto menor for essa distância, mais rápido pode-se eliminar informação repetida. Assim, nosso interesse em um primeiro momento é de estudar o caso onde todos os sensores de um cluster

estão conectados diretamente a seu mestre, ou seja, essa distância é de um salto. Uma rede de sensores clusterizada deve ser necessariamente uma árvore dominante. Uma topologia em árvore dominante pode ser vista como um subgrafo acíclico $T = (V(T), E(T))$ de um grafo $G = (V, E)$, com $V(T) \subset V$ e $E(T) \subset E$, com V e E sendo o conjunto de nós (sensores) e de conexões entre esses sensores em G, respectivamente. Uma conexão entre dois sensores existe se ambos estão no raio de alcance um do outro. O objetivo é determinar uma estrutura minimamente conexa entre sensores permanentemente ativos (dominantes) de forma que cada sensor em estado de dormência (dominado) esteja dentro do raio de alcance de um sensor ativo. Além disso, deseja-se estabelecer uma estrutura clusterizada para a rede, onde cada cluster é formado por um sensor mestre, em permanente atividade, e sensores adjacentes a ele. Sensores mestres são alocados em pontos da rede sem que dois deles pertençam ao raio de alcance um do outro, formando um conjunto independente. Sensores pontes são nós dominantes e devem estar sempre 'acordados', mas não coletam dados de sensores dominados. O principal consumo de energia é para manter conexos os sensores mestres e pontes e depende da distância entre eles. Dessa forma, o objetivo do problema é obter uma estrutura em 'árvore dominante' que tenha o menor custo de conexão (soma das distâncias entre os sensores dominantes) mantendo uma estrutura de conjunto independente para os mestres [159].

Para definir o problema formalmente, inicialmente considere alguns conceitos. No grafo $G = (V, E)$ cada aresta $\{u, v\} \in E$ tem custo $c_{uv} \geq 0$. O custo relacionado à comunicação bidirecional entre dois sensores u e v, conectados por uma aresta $\{u, v\} \in E$, é o mesmo, ou seja, $c_{uv} = c_{vu}$. Um conjunto $S \subseteq V$ é dominando se, para cada $v \in V$, $v \in S$ ou v é adjacente a algum $u \in S$; caso contrário, S é um conjunto não dominante. Um nó pertencente (resp. não pertencente) a um conjunto dominante é um nó dominante (resp. dominado). Um conjunto $S \subseteq V$ é independente se, para cada par $u, v \in S$, então $\{u, v\} \notin E$. Seja $N(u) = \{v \in V \mid \{v, u\} \in E\}$ a vizinhança do nó $u \in V$. A vizinhança fechada de $v \in V$ é definida como $N[u] = N(u) \cup \{u\}$. Uma árvore T de G, um subgrafo acíclico conexo, é dominante se seus nós formam um conjunto dominante de G. Se escolhermos um nó s de uma árvore T para ser sua raiz, então a contagem de saltos $\pi(u)$ representa o número de saltos (em número de conexões) no caminho de um nó u de T a s, com $pi(s) = 0$. Definimos um grafo direcionado $\hat{G} = (V, A)$ obtido de $G = (V, E)$ cujo conjunto de nós é o mesmo e o de arcos A é tal que se uma aresta $\{u, v\} \in E$, então os arcos (u, v) e (v, u) pertencem a A, com os custos $c_{uv} = c_{vu}$ sendo iguais ao custo da aresta $\{u, v\}$. Uma árvore s-enraizada T de $\hat{G}$ é uma árvore sem arco de entrada em s, enquanto todos os outros nós v de T tem exatamente um arco de entrada. Uma arborescência s-enraizada é uma árvore na qual há exatamente um caminho direcionado de s para todos os outros nós.

Desenvolvemos um modelo matemático baseado em saltos para esse problema como segue. Seja uma arborescência T de $\hat{G}$ representada pelas variáveis $x \in \{0, 1\}^{|A|}$, onde $x_{uv} = 1$ se o arco (u, v) pertence a T, e $x_{uv} = 0$, caso contrário. Considere variáveis $y_v \in \{0, 1\}$, para todo $v \in V$, onde $y_v = 1$ se v é um nó mestre de T; e $y_v = 0$, caso contrário. Da mesma forma, vamos definir variáveis $z_v \in \{0, 1\}$, para todo $v \in V$, onde $z_v = 1$ se v é um nó ponte de T; e $z_v = 0$, caso contrário. Para todo $v \in V$, vamos definir variáveis $\pi_v \in \mathbb{R}_+$ representando a contagem de saltos (em número de arcos) do nó raiz para v em T. Para nós que não pertencem a T, sua distância até o nó raiz é irrelevante. O modelo matemático é:

$$(M) \quad \min \sum_{(u,v)\in A} c_{uv}x_{uv} \tag{5}$$

$$\sum_{(u,v)\in A} x_{uv} = \sum_{v\in V}(z_v + y_v) - 1, \tag{6}$$

$$\pi_v - \pi_u - (|V|-1)x_{uv} - (|V|-3)x_{vu} \geq 2 - |V|, \quad \forall\,(u,v) \in A, \tag{7}$$

$$\sum_{u\in N(v)} x_{uv} \leq y_v + z_v, \quad \forall\, v \in V, \tag{8}$$

$$\sum_{u\in N[v]} y_u \geq 1, \quad \forall\, v \in V, \tag{9}$$

$$y_u + y_v \leq 1, \quad \forall\,\{u,v\} \in E, \tag{10}$$

$$y_v + z_v \leq 1, \quad \forall\, v \in V, \tag{11}$$

$$z_u + z_v + x_{uv} + x_{vu} \leq 2, \quad \forall\,\{u,v\} \in E, \tag{12}$$

$$2x_{uv} \leq z_u + z_v + y_u + y_v, \quad \forall\,(u,v) \in A, \tag{13}$$

$$z, y \in \{0,1\}^{|V|}, x \in \{0,1\}^{|A|}, \pi \in [0, |V|-2]^{|V|}. \tag{14}$$

No modelo (M), a restrição (6) estabelece que o o número de arcos é uma unidade a menos que a soma dos nós mestres e pontes em uma solução T. Restrições (7) estabelecem que se o arco (u,v) pertencer a T, então π_v é pelo menos uma unidade maior que π_u. Pelas restrições (8) e (11), há no máximo um arco entrando em cada nó pertencente a T. Restrições (7) e (8) eliminam ciclos. Restrições (9) impõem que um nó ou é um de seus vizinhos é um nó mestre dominante. Restrições (10) garantem que as duas extremidades de uma aresta não podem ser ambas nós mestres. Restrições (11) estabelecem que $v \in V$ é um mestre ou um nó ponte; caso contrário, não pertence a T. Restrições (12) impõem que se um dos arcos (u,v) ou (v,u) pertencer a T, então ambas as suas extremidades não podem ser nós ponte. Pelas restrições (13), se um arco (u,v) pertencer a T, então uma de suas extremidades será ponte e a outra um nó mestre. Como as restrições (11) impõe que no máximo uma das variáveis y_w e z_w será 1, para cada $w \in V$, então as restrições (8) e (13) também impõe que se u (resp. v) for um nó dominado, então nenhum arco sai de u (resp. entra em v) na solução. O domínio das variáveis é dado em (14).

Proporemos modelos matemáticos, desigualdades válidas e algoritmos de planos de corte (branch-and-cut) para esse problema. Por essa topologia de rede ser nova, iremos compará-la a outras existentes na literatura [2, 190] em termos de custo da (e nós dominantes da) solução. Por conta da existência de fluxo de dados, trabalharemos igualmente algoritmos de fluxo máximo para tratar o problema.

9.3 Jogos Multiagentes em Grafos

9.3.1 Jogos de Coloração em Grafos

No Jogo clássico de Coloração de Grafos, dado um grafo G e um conjunto C de inteiros (representando o conjunto de cores), dois jogadores (Alice e Bob) jogam alternadamente (começando com Alice) escolhendo vértices ainda não coloridos para que sejam coloridos com uma cor do conjunto C não atribuída a nenhum de seus vizinhos que já estejam coloridos. Alice ganha se todos os vértices do grafo forem coloridos com sucesso, caso contrário, Bob é o vencedor. Do teorema clássico de Zermelo-von Neumann da Teoria dos Jogos, um dos dois jogadores tem uma estratégia vencedora, já que são jogos finitos de informação perfeita e sem empate [204]. Assim, quando ambos os jogadores usam estratégias ótimas, o número jogo-cromático $\chi_g(G)$ é definido como o menor número de cores no conjunto C para o qual Alice tem uma estratégia vencedora no Jogo de Coloração no grafo G.

Esse jogo foi apresentado pela primeira vez em 1981 na Scientific American [119] (coluna *Mathematical Games*), mas ficou esquecido durante 10 anos e foi "reinventado" por Bodlaender [42] em 1991. Desde então,

esse jogo tem atraído considerável atenção na pesquisa científica. Por exemplo, em 1993, Faigle et al. [105] mostraram que $\chi_g(G) \leq 3\omega(G) - 2$ para grafos de intervalo com número de clique $\omega(G)$. Em 1994, Kierstead e Trotter [141] provaram que $\chi_g(G) \leq 7$ em grafos periplanares e, em 2008, Zhu [206] provou que $\chi_g(G) \leq 17$ para grafos planares quaisquer. Em 2008, Bohman, Frieze e Sudakov [43] investigaram $\chi_g(G_{n,p})$ para o grafo aleatório $G_{n,p}$ com respeito ao seu comportamento assintótico.

No artigo de 1991 [42], a complexidade computacional foi deixada como "*um problema interessante em aberto*". Um ponto de dificuldade para definir a complexidade é a definição do problema de decisão. Como apontado por Zhu [205], o Jogo de Coloração de Grafos "*exibe algumas propriedades estranhas*" e a seguinte questão aparentemente ingênua ainda está em aberto: **Problema JC1 (Zhu [205]):** Alice tem uma estratégia vencedora para o jogo de coloração com $k+1$ cores se ela tiver uma estratégia vencedora com k cores? Assim, dado um grafo G e um inteiro positivo k, é possível definir dois problemas de decisão para o jogo de coloração de grafos. **Problema JC2:** $\chi_g(G) \leq k$? **Problema JC3:** Alice tem uma estratégia vencedora com k cores? Os Problemas JC2 e JC3 são equivalentes se e somente se o Problema JC1 tiver uma resposta afirmativa. Mesmo assim, 2 membros deste projeto provaram em 2020 que os Problemas JC2 e JC3 são PSPACE-Difíceis [82], respondendo a questão de Bodlaender [42] de 1991.

Em 2013, Havet e Zhu [128] propuseram o Jogo Guloso de Coloração de Grafos e o número jogo-Grundy $\Gamma_g(G)$ e provaram resultados para florestas e árvores parciais. Nesse jogo, Alice e Bob escolhem apenas os vértices, não podendo escolher a cor, que é sempre a menor cor possível (lembre que as cores são inteiros) e por isso se chama gulosa. Em [82], 2 membros desse projeto também provaram que o jogo guloso é PSPACE-Difícil, mas que é polinomial em grafos split e em várias superclasses de cografos.

Em 2019, Andres e Lock [10] propuseram cinco variantes do jogo de coloração: g_B (Bob inicia o jogo), $g_{A,A}$ (Alice começa e pode passar jogadas), $g_{A,B}$ (Alice começa e Bob pode passar jogadas), $g_{B,A}$ (Bob começa e Alice pode passar jogadas) e $g_{B,B}$ (Bob começa e pode passar jogadas). Eles deixaram o seguinte problema: "*a questão da PSPACE-dificuldade permanece em aberto para todas as variantes mencionadas acima*". Também em 2019, Charpentier, Hocquard, Sopena e Zhu [68] propuseram uma versão conexa do jogo de coloração (começando com Alice): o subgrafo induzido pelo conjunto de vértices coloridos deve ser conexo a cada momento do jogo. Eles provam que Alice ganha com 2 cores em grafos bipartidos e com 5 cores em grafos periplanares. Em 2020, 2 membros deste projeto conseguiram obter a complexidade computacional de todas essas variantes do jogo de coloração [162].

Apesar desses resultados promissores, muitos problemas ainda estão em aberto. Por exemplo, a variante do Jogo de Coloração Gulosa-Conexa, da qual nada se sabe. Ademais, para cada um desses jogos, pode-se investigar a variante em que um jogador inicia o jogo e outro jogador (ou o mesmo) pode passar jogadas. Finalmente, a questão proposta em [128] de se o jogo de coloração gulosa sempre obtém no máximo o mesmo número de cores do jogo clássico de coloração permanece em aberto. O mesmo pode ser dito sobre a relação entre jogo de coloração gulosa-conexa e o jogo de coloração conexa. Pretende-se investigar tais questões neste projeto, buscando contraexemplos por meio computacional com auxílio de paralelismo, bem como com provas matemáticas para a veracidade dessas questões.

9.3.2 Jogos de Perseguição em Grafos

Dados inteiros $s \geq 2$, $d \geq 0$ e $k \geq 1$ (respectivamente a velocidade do espião, a distância de vigilância e o número de guardas), o jogo do espião-(s, d) é um jogo em um grafo finito G com k guardas, que cooperam entre si, e um espião ocupando vértices de G. Os guardas e até o espião podem ocupar o mesmo vértice. É um jogo de informação completa: qualquer jogador tem total conhecimento das posições dos outros jogadores. Inicialmente, o espião é colocado em algum vértice de G e, em seguida, os k guardas são colocados em alguns vértices de G. O jogo prossegue passo a passo: primeiro o espião pode se mover ao longo de no máximo s arestas e depois cada guarda pode se mover ao longo de uma aresta. O espião ganha se, após um número finito de jogadas (após o movimento dos guardas), atingir um vértice a uma distância maior que d de cada guarda. Caso contrário, os guardas ganham o jogo: sempre há pelo menos um guarda à distância no máximo d do espião. Do clássico teorema de Zermelo-von Neumann [204], o espião ou os guardas têm uma estratégia vencedora, pois é um jogo finito de informação perfeita sem empate. Aqui podemos considerar o jogo de

espionagem como um jogo finito visto que o número de configurações possíveis do espião e dos guardas é finito em um grafo finito G. Por exemplo, podemos considerar que os guardas ganham o jogo se o espião repetir uma configuração de jogo (após sua jogada).

O parâmetro $gn_{s,d}(G)$ (*guard number*) é o número mínimo de guardas de forma que os guardas tenham uma estratégia vencedora no jogo do espião-(s,d). Pela definição, apenas um guarda é sempre suficiente se a velocidade do espião $s = 1$. Por esta razão, consideramos $s \geq 2$.

O jogo do espião foi introduzido por Cohen et al. em 2016 [76] e está intimamente relacionado ao conhecido jogo *Polícia e Ladrão* [172, 45]. Neste jogo, primeiro k policiais ocupam alguns vértices do grafo e depois um ladrão ocupa um vértice. Passo a passo, cada agente pode se mover (os policiais primeiro e depois o ladrão) ao longo de uma aresta. Os policiais ganham se um policial ocupar o mesmo vértice do ladrão após um número finito de jogadas. O parâmetro $cn(G)$ (*cop-number*) é o número mínimo de policiais necessários para capturar o ladrão em G [8].

Existem muitas generalizações do jogo de Polícia e Ladrão [44, 116, 9, 65, 113]. Por exemplo, permitindo um ladrão mais rápido com velocidade $s \geq 2$. Nesta variante, o número exato de policiais com velocidade 1 necessário para capturar um ladrão com velocidade $s = 2$ é desconhecido mesmo em grades bidimensionais [115, 26]. Em 2010, Bonato et al. [44] introduziram outra variante em que o jogo termina se um policial ocupa um vértice à distância no máximo um dado inteiro d do ladrão. Isso equivale ao jogo do espião-(s,d) para velocidade $s = 1$ em uma variante em que o espião é colocado após os guardas. Para a velocidade $s \geq 2$, a equivalência não é verdadeira e os jogos são significativamente diferentes.

Outro jogo bem conhecido relacionado é o jogo da *Dominação Eterna* [121, 122, 144, 145]. Um conjunto de k *defensores* ocupa alguns vértices de um grafo G. A cada passo, um único *atacante* escolhe um vértice $v \in V(G)$ qualquer e os defensores podem se mover ao longo de uma aresta de tal forma que pelo menos um defensor esteja à distância de no máximo um dado inteiro d de v. Existem algumas variantes do jogo de dominação eterna que permitem que mais defensores se movam em cada turno e ocupem o mesmo vértice [122, 144, 145], que são equivalentes ao jogo do espião quando a velocidade do espião é infinita (ou pelo menos o diâmetro do grafo).

O Jogo de Polícia e Ladrão é o mais antigo proposto na literatura, mas apenas recentemente a sua complexidade computacional foi obtida. Em 2012, provou-se que o problema é PSPACE-Difícil [161] e em 2015 provou-se que é EXPTIME-Difícil [142]. Em 2018, dois membros deste projeto provaram que o Jogo do Espião é NP-Difícil e que uma variante direcionada é PSPACE-Difícil [76]. Em 2021, dois membros deste projeto também obtiveram resultados do Jogo do Espião em grades e em produtos de grafos e provaram que é NP-Difícil também em grafos bipartidos [81], mas a questão da PSPACE-Dificuldade permanece em aberto. Muitos problemas ainda estão em aberto para estes jogos de perseguição em grafos, tanto com relação à complexidade computacional, bem como com relação à polinomialidade em classes de grafos.

9.4 Difusão e Propagação em Redes Sociais

Hoje em dia, as redes sociais são muito presentes no cotidiano e são ferramentas poderosas para divulgação de informação e conteúdos. Nesse ambiente, surgem várias questões importantes, como determinar a menor quantidade de indivíduos necessários para difundir certa informação dentro de uma comunidade de interesse, ou ainda determinar o tempo máximo para que uma certa informação seja propagada para toda a rede.

Por exemplo, no problema TSS (*Target Set Selection* [75]), bastante estudado recentemente, cada vértice v do grafo possui um valor limite $\lim(v) > 0$ de modo que vértices ativados nunca se tornam desativados e um vértice não ativado v é ativado se possui $\lim(v)$ vizinhos já ativados. Nesse contexto, são interessantes problemas como determinar o menor número de vértices necessários para ativar todo o grafo (TSS-SIZE [75]), ou determinar o tempo máximo para que o grafo inteiro seja ativado (TSS-TIME [139]), ou determinar o menor número de vértices para que o grafo inteiro seja ativado em um único passo (VECTOR-DOMINATION ou TSS-DOMINATION [158]), bem como determinar o maior conjunto que seja convexo (TSS-MAX-CONVEX), incapaz de ativar novos vértices, para identificação de comunidades "fechadas" em uma rede social.

Problemas dessa natureza podem ser modelados por meio do conceito de convexidade em grafos, particularmente convexidade de intervalos.

9.4.1 Convexidade de Grafos

Em um espaço Euclidiano E, um conjunto $S \subseteq E$ é *convexo* se, para cada par de pontos $x, y \in S$, o conjunto de pontos no segmento de reta que os liga também pertence à S. Consequentemente, a interseção de dois conjuntos convexos é um conjunto convexo. O fecho convexo de um conjunto $S \subseteq E$ é o menor conjunto convexo que o contém. Para mais detalhes sobre convexidade no espaço Euclidiano, veja [188].

De maneira mais geral, uma *convexidade* ou *alinhamento* sobre um conjunto finito X é uma família $\mathcal{C}$ de subconjuntos de X que é fechada sobre interseção e contém o conjunto vazio e o próprio conjunto X. Tais subconjuntos são ditos *convexos*. O par $(X, \mathcal{C})$ é um *espaço alinhado* [108]. Dado um espaço alinhado $(X, \mathcal{C})$, o *fecho convexo* de $S \subseteq X$ é o menor conjunto convexo de $\mathcal{C}$ que o contém.

A noção de convexidade é associada com grafos quando o par $(X, \mathcal{C})$ é definido em função de um grafo G dado como entrada. Na maioria das convexidades em grafos, o conjunto X corresponde ao conjunto de vértices de G e os subconjuntos em $\mathcal{C}$ correspondem a subconjuntos de vértices que correspondem à alguma propriedade sobre caminhos em G. Tais convexidades em geral são casos particulares de uma convexidade de intervalos.

Uma *convexidade de intervalos* é um espaço alinhado $(X, \mathcal{C})$ onde a família $\mathcal{C}$ é definida a partir de uma função $I : X^2 \to 2^X$ que associa, a cada par de elementos de X, um subconjunto de X. A partir dessa função, conhecida como *função de intervalos*, pode-se definir $I : 2^X \to 2^X$ como $I(S) = \bigcup_{u,v \in S} I(u, v)$, para cada $S \subseteq X$. É requerido que esta função seja monótona. Os subconjuntos da família $\mathcal{C}$ são então os pontos fixos de I, ou seja, os conjuntos convexos desta convexidade. Detalhamos a seguir as convexidades de intervalos que serão abordadas neste projeto.

Convexidade Geodética. Seja $G = (V, E)$ um grafo simples, conexo e não-direcionado. Para cada par $u, v \in V(G)$, o *intervalo fechado*, $I(u, v)$, é o conjunto de vértices que pertencem a algum **caminho mais curto** entre u e v. Para cada $S \subseteq V(G)$, defina $I(S) = \bigcup_{u,v \in S} I(u, v)$. Um subconjunto $S \subseteq V(G)$ é dito *geodeticamente convexo* se $I[S] = S$. Dessa forma, se $\mathcal{C}(\mathcal{G})$ é o conjunto de todos os subconjuntos geodeticamente convexos de $V(G)$, então o par $(V(G), \mathcal{C}(\mathcal{G}))$ é a convexidade geodética de G [71, 72]. Há inclusive um livro publicado tratando exclusivamente desse tipo de convexidade [178].

Convexidade P_3. Seja $G = (V, E)$ um grafo simples, conexo e não-direcionado. Para cada par $u, v \in V(G)$, o intervalo fechado, $I(u, v)$, é o conjunto de vértices que são vizinhos de u e v simultaneamente, ou seja, é o conjunto de vértices que pertencem a algum **caminho com 3 vértices** cujas extremidades são u e v (independente da existência ou não da aresta $uv \in E(G)$). De modo similar, definem-se os termos conjunto convexo, e a convexidade P_3 para G [108, 67].

Convexidade P_3^*. Neste caso, a única diferença com a anterior é que são considerados apenas **caminhos induzidos com 3 vértices** na definição do intervalo $I(u, v)$, ou seja, apenas quando $uv \notin E(G)$. Novamente, as noções de conjunto convexo e de convexidade P_3^* de G são definidas analogamente [24].

Outras convexidades. Há ainda diversas outras convexidades definidas na literatura como, por exemplo, a monofônica [108, 97] e a de ciclos [18, 22]. Elas não serão objeto principal dos nossos estudos, mas caso percebamos que alguns resultados obtidos também se apliquem a essas convexidades, tais resultados certamente serão escritos de modo mais geral, de forma a contemplar também tais convexidades.

Para cada convexidade supramencionada, há diversos parâmetros que podem ser estudados, como, por exemplo, o número de envoltória, o número de intervalo, o posto, o número de convexidade, o número de Carathéodory, o número de Radon, o número de Helly, etc.

Focaremos nos três primeiros, definidos a seguir.

Número de intervalo. Para um grafo simples G, o *número de intervalo* de G em uma dada convexidade é a menor cardinalidade de um subconjunto $S \subseteq V(G)$ tal que $I(S) = V(G)$.

Número de envoltória. Para um grafo simples G, o *número de envoltória* de G em uma dada convexidade é a menor cardinalidade de um subconjunto $S \subseteq V(G)$ tal que a envoltória convexa de S é igual a $V(G)$.

Posto. Um subconjunto de vértices $S \subseteq V(G)$ de um grafo simples G é dito *convexamente independente* se, para todo $v \in S$, temos que v não pertence à envoltória convexa de $S \setminus \{v\}$. O *posto* de G é a cardinalidade do maior conjunto convexamente independente de G.

A literatura sobre os três parâmetros nessas três convexidades mencionadas é muito extensa e não faremos aqui o detalhamento da mesma.

Apesar de restarem inúmeras questões em aberto, inclusive do ponto de vista de Complexidade Parametrizada desses parâmetros nessas convexidades, o nosso enfoque principal será estudar os parâmetros e convexidades analogamente definidas para grafos orientados.

Grosso modo, o que muda é que os caminhos considerados agora são caminhos direcionados e, na função de intervalos, consideramos tanto caminhos direcionados de u para v como de v para u.

Apesar dos primeiros artigos sobre convexidade tratarem do caso direcionado, a maioria da literatura feita em seguida lida com grafos não direcionados. Encontramos poucas referências sobre o caso direcionado [109, 70, 183, 69, 194, 104, 21].

9.5 Complexidade Parametrizada

Uma instância de um *problema parametrizado* tem a forma (x, k), onde x é a entrada (normalmente um grafo) e k é um inteiro não negativo chamado de *parâmetro*. Um algoritmo *tratável com parâmetro fixo*, abreviado como FPTdo inglês *fixed-parameter tractable*, é um algoritmo que decide se (x, k) é uma instância positiva em tempo $f(k) \cdot |x|^{\mathcal{O}(1)}$, onde f é uma função computável que depende apenas de k [85]. *Kernelização* [114] é uma das áreas mais estudadas dentro de complexidade parametrizada, onde o objetivo é decidir se uma instância (x, k) de um problema parametrizado pode ser transformado em tempo polinomial numa instância equivalente (x', k') cujo tamanho é limitado por uma função de k; a instância reduzida é chamada de um *kernel* e achar um kernel de tamanho pequeno, normalmente polinomial ou até mesmo linear em k, é uma das áreas mais ativas de complexidade parametrizada.

9.5.1 Relação entre kernelização e aproximabilidade

Uma *cobertura por vértices* de um grafo G é um subconjunto de vértices contendo pelo menos uma extremidade de cada aresta de G. No problema de Cobertura por Vértices parametrizado, recebemos como entrada um grafo G e um inteiro k e o objetivo é decidir se G tem uma cobertura por vértices de tamanho no máximo k. No problema de otimização associado, chamado de Cobertura por Vértices Mínima, o objetivo consiste em encontrar uma cobertura por vértices de um grafo G de tamanho mínimo. Este problema é recorrente na área de complexidade parametrizada [84, 98], servindo como problema de teste para muitas das técnicas fundamentais.

Um *conjunto de retroalimentação* de um grafo G é um subconjunto de vértices contendo pelo menos um vértice em cada ciclo de G. No problema de Conjunto de Retroalimentação parametrizado, recebemos como entrada um grafo G e um inteiro k e o objetivo é decidir se G tem um conjunto de retroalimentação de tamanho no máximo k. No problema de otimização associado, chamado de Conjunto de Retroalimentação Mínimo, o objetivo consiste em encontrar um conjunto de retroalimentação de um grafo G de tamanho mínimo.

A versão "max-min" de um problema de minimização consiste em maximizar o tamanho de uma solução minimal do problema correspondente. Esta variação do problema tem sido aplicada a vários problemas como, por exemplo, Cobertura por Vértices [14, 47, 111], Conjunto Dominante [31, 101] (cuja versão "max-min" é chamada de Dominação Superior), Conjunto de Retroalimentação [100], ou Hitting Set [15, 86].

Fernau [111] observou que o problema de Cobertura por Vértices Max-Min (CVMM) parametrizado pelo tamanho da solução admite um kernel com no máximo k^2 vértices. Dublois et al. [100] mostraram recentemente um kernel com no máximo k^3 vértices para o problema de Conjunto de Retroalimentação Max-Min (CRMM) parametrizado pelo tamanho da solução.

J. Araújo e V. Campos, pesquisadores membros deste projeto, em conjunto com M. Bougeret e I. Sau, definiram o conceito de *lop-kernel* [14] para problemas de otimização sobre subconjuntos de vértices. Informalmente, um lop-kernel corresponde a um algoritmo polinomial que decide o problema ou retorna uma instância equivalente em que o valor de uma solução ótima é "preservado", no sentido de que o decréscimo do valor ótimo é limitado pela diferença entre o parâmetro do problema original e do problema reduzido. Vale ressaltar que este tipo de kernel captura a vasta maioria de kerneis conhecidos na literatura.

Sobre lop-kerneis, o seguinte resultado é obtido em [14].

Teorema 3. *Seja* Π *um problema de maximização sobre subconjuntos de vértices e sejam* $r, \varepsilon \in (0, 1]$. *Se* Π *não admite um algoritmo aproximativo com fator de aproximação* $\mathcal{O}(n^{r-\varepsilon})$ *sobre grafos com* n *vértices, então* Π *parametrizado pelo tamanho da solução não admite um lop-kernel com* $\mathcal{O}(k^{\frac{1}{1-r}-\varepsilon'})$ *vértices para* $\varepsilon' = \frac{\varepsilon}{(1-r+\varepsilon)(1-r)}$ *quando* $r \in (0, 1)$ *ou com* $\mathcal{O}(k^{\frac{1}{\varepsilon}})$ *vértices quando* $r = 1$.

Observe que este resultado permite a transferência de limites inferiores de aproximação para lop-kerneis sem a necessidade de qualquer construção. Boria et al. [47] provaram que o problema CVMM não admite um algoritmo aproximativo com fator de aproximação $\mathcal{O}(n^{\frac{1}{2}-\varepsilon})$ para qualquer $\varepsilon > 0$, a menos que $\mathsf{P} = \mathsf{NP}$. Aplicando o Teorema 3 com $r = \frac{1}{2}$ obtemos o seguinte corolário, que diz que o kernel quadrático de Fernau [111] é essencialmente ótimo.

Corolário 4. Cobertura por Vértices Max-Min *parametrizado pelo tamanho da solução não admite um lop-kernel com* $\mathcal{O}(k^{2-\varepsilon})$ *vértices para qualquer* $\varepsilon > 0$, *a menos que* $\mathsf{P} = \mathsf{NP}$.

Dublois et al. [100] provaram que o problema CRMM não admite um algoritmo aproximativo com fator de aproximação $\mathcal{O}(n^{\frac{2}{3}-\varepsilon})$ para qualquer $\varepsilon > 0$, a menos que $\mathsf{P} = \mathsf{NP}$. Aplicando o Teorema 3 com $r = \frac{2}{3}$ obtemos o seguinte corolário, que diz que o kernel cúbico de Dublois et al. [100] é essencialmente ótimo.

Corolário 5. Conjunto de Retroalimentação Max-Min *parametrizado pelo tamanho da solução não admite um lop-kernel com* $\mathcal{O}(k^{3-\varepsilon})$ *vértices para qualquer* $\varepsilon > 0$, *a menos que* $\mathsf{P} = \mathsf{NP}$.

As técnicas existentes para obter limites inferiores de kernelização incluem composição cruzada [39, 40], composição fraca [95, 96, 130], transformações de parâmetro polinomial [35, 41], técnicas que obtem limites inferiores para os coeficientes de kerneis lineares [74], ou que relacionam kernelização e aproximação [1, 36, 126, 147, 153]. Comparado às técnicas existentes, o limite inferior obtido pelo Teorema 3 tem a vantagem de ser simples, de aplicação direta e baseado apenas na mesma condição do resultado de inaproximabilidade, que normalmente é $\mathsf{P} \neq \mathsf{NP}$. Por outro lado, ele tem algumas desvantagens. Primeiro, ele só pode ser aplicado para problemas de otimização sobre subconjuntos de vértices que são difíceis de aproximar (a versão para problemas de minimização estão em [14]), ou seja, por um fator de aproximação de $\mathcal{O}(n^{r-\varepsilon})$ para alguma constante $r > 0$. Finalmente, este limite inferior se aplica apenas à existência de lop-kerneis. Kerneis baseados em regras não padrão menores ainda podem existir.

Vale ressaltar que a vasta maioria dos kerneis exitentes na literatura são de fato lop-kerneis. Isto ocorre, por exemplo, para todos os kerneis obtidos em livros mais recentes focados apenas em kernelização [114]. Este fato indica a importância no estudo mais aprofundado de lop-kerneis.

O primeiro problema que desejamos abordar consiste em adaptar lop-kerneis para problemas que não sejam baseados em subconjuntos de vértices. Também seria interessante obter um resultado equivalente ao Teorema 3 para transferir resultados mais fracos de inaproximabilidade como, por exemplo, de inaproximabilidade por um fator constante.

Problema 6. *É possível adaptar lop-kerneis para problemas parametrizados que não sejam de maximização ou minimização de subconjuntos de vértices?*

Problema 7. *É possível transferir resultados de inaproximabilidade com fator de aproximação constante para limites inferiores de lop-kerneis?*

Boria et al. [47] perguntaram se existem kerneis de tamanho $o(k^2)$ para o problema de CVMM parametrizado pelo tamanho da solução. Embora o Corolário 4 ofereça uma resposta parcial para esta pergunta,

seria interessante responder esta pergunta por completo.

Problema 8. *Existe kernel de tamanho $o(k^2)$ para o problema de* CVMM *parametrizado pelo tamanho da solução?*

Como a grande maioria dos kerneis obtidos na literatura são lop-kerneis, é importante estudar os exemplos disponíveis de kerneis que não são.

Problema 9. *Quais exemplos de kerneis que não são lop-kerneis existem na literatura?*

O único resultado conhecido pelos proponentes que não é um lop-kernel consiste numa redução algébrica de Giannopoulou et al. [120] para o problema de remover no máximo k vértices para obter uma árvore, baseado em identificar um subconjunto de desigualdades lineares que capturam todas as soluções de tamanho no máximo k. Este problema não possui algoritmo aproximativo com fator de aproximação $\mathcal{O}(n^{1-\varepsilon})$ para qualquer $\varepsilon > 0$, a menos que P = NP [202]. Giannopoulou et al. [120] apresentam um kernel com $\mathcal{O}(k^4)$ vértices mas a versão de minimização do Teorema 3 presente em [14] indica que não pode haver lop-kernel polinomial a menos que P = NP. Assim, não é possível que o kernel apresentado seja um lop-kernel.

É importante observar que o Teorema 3 oferece uma barreira de dificuldade para a obtenção de kerneis. Este resultado também indica o que deve ser procurado para quebrar esta barreira, ou seja, a redução não pode manter soluções grandes.

Problema 10. *A técnica utilizada por Giannopoulou et al. pode ser adaptada para a obtenção de outros kerneis que quebrem o limite inferior para lop-kerneis?*

Nos problemas CVMM e CRMM temos um lop-kernel e um limite inferior obtido pelo Teorema 3 que mostram que, dentre lop-kerneis, estes são melhores possíveis. Tais lop-kerneis ótimos podem ser estendidos para quais outros problemas?

Problema 11. *Existe uma família infinita de problemas com exemplos de lop-kerneis ótimos?*

9.6 HPC Shelf, Uma Plataforma Para Aplicações e Sistemas HPC

Na Seção 3.4, motivamos o emprego do processamento paralelo para resolução de grandes instâncias de problemas de otimização combinatória, que surgem em aplicações práticas como aquelas descritas na Seção 3.3. Além da contribuição dos pesquisadores da área de HPC em desafios relativos à construção de versões paralelas de algoritmos voltados a plataformas de computação paralela de interesse, levando em consideração requisitos de computação heterogênea e em larga escala, introduzimos a HPC Shelf como plataforma para compartilhamento de contribuições em termos de sistemas de software e aplicações desenvolvidos no âmbito do projeto, tanto entre os pesquisadores envolvidos no projeto quanto para a comunidade externa.

Nesta seção, apresentamos mais detalhes sobre a arquitetura da HPC Shelf, bem como seus fundamentos.

A HPC Shelf é uma plataforma de computação paralela destinada a oferecer serviços de HPC para *aplicações* que atendem a uma comunidade de usuários específica chamados *especialistas de domínio*. Tais aplicações usam os serviços da HPC Shelf para criar soluções computacionais para problemas descritos por especialistas de domínio na forma de *sistemas de computação paralela* construídos pela composição de componentes. A HPC Shelf não prescreve o tipo de interface que suas aplicações devem oferecer para interagir com usuários especialistas. Aplicações podem usar diferentes tipos de interfaces de alto nível, tais como portais da web, ambientes de resolução de problemas, APIs de programação, interfaces de linha de comando e assim por diante, possivelmente abstraindo dos usuários especialistas a natureza concreta dos sistemas de computação paralela e concentrando-se nos meios de descrição de problemas e apresentação de suas soluções. A HPC Shelf é especialmente direcionada a problemas que exigem processamento paralelo distribuído em larga escala, ou seja, envolvendo uma ou mais plataformas de computação paralela, dentre clusters e MPPs, em centros de HPC/supercomputação de universidades, laboratórios de pesquisa, indústrias e provedores de infraestrutura em nuvens computacionais (serviços IaaS).

9.6.1 The Hash Component Model

Os componentes de sistemas de computação paralela da HPC Shelf seguem o modelo de componentes paralelos Hash [90]. O modelo Hash foi introduzido para atender aos requisitos de um modelo de componente

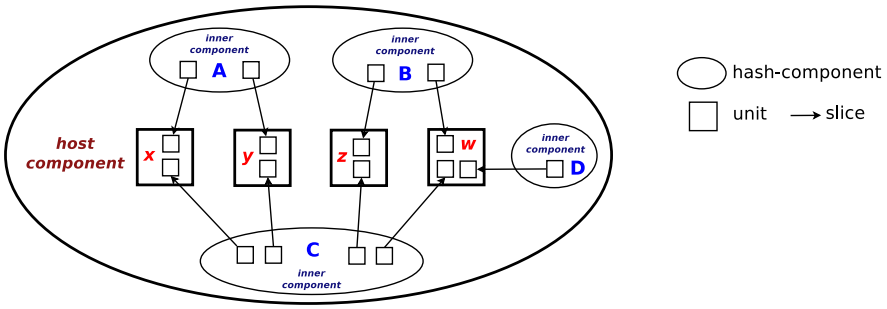

Figura 1: A estrutura de um componente Hash

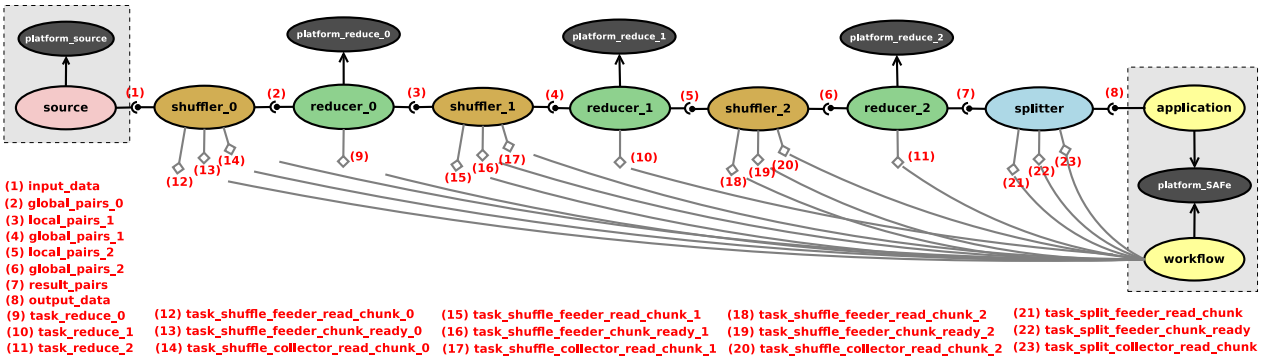

Figura 2: A MapReduce parallel computing system

para frameworks computacionais baseados em componentes direcionadas à construção de sistemas HPC [88]. Em particular, aborda o problema de encapsular interesses de paralelismo em componentes, ditos componentes paralelos, conduzindo a uma noção geral de componente paralelo [90] que leva em consideração a íntima relação entre as decisões na implementação desses componentes e as características das plataformas de computação paralela alvo, a fim de explorar de maneira eficiente seu desempenho potencial.

A Figura 1 ilustra os elementos principais na estrutura de um componente Hash (elipse). Um conjunto de *unidades* forma um componente Hash (por exemplo, retângulos x, y, z e w), cada um supostamente alocado em um nó de processamento de uma plataforma de computação paralela de memória distribuída (e.g., cluster ou MPP). Um componente paralelo pode ser construído através de um tipo de composição recursiva chamada *composição por sobreposição*, hospedando um conjunto de *componentes aninhados* (por exemplo, as elipses A, B, C e D). De acordo com a composição por sobreposição, cada unidade de um componente aninhado é mapeada para uma unidade de componente hospedeiro por meio de uma *fatia* (setas). Cada unidade implementa seu papel na implementação da preocupação abordada por seu componente hospedeiro usando os serviços fornecidos pelos seus componentes aninhados através das fatias. Uma plataforma de componentes que esteja em conformidade com o modelo Hash distingue componentes de acordo com um conjunto de *espécies de componentes*, representando abstrações para os diferentes blocos de construção de sistemas suportados pela plataforma, abordando preocupações tanto funcionais quanto não-funcionais. Na próxima seção, apresentamos as espécies de componentes suportados pela HPC Shelf.

9.6.2 Sistemas de Computação Paralela

Os componentes de um sistema de computação paralela da HPC Shelf são: um único componente *workflow*, um único componente *application* e um conjunto de *componentes de solução*. Os componentes de solução pertencem a uma dentre as seguintes espécies de componentes. ***Plataformas Virtuais***, representando pla-

taformas de computação paralela de memória distribuída, tais como clusters e MPPs. ***Fontes de Dados***, representando repositórios de dados, possivelmente distribuídos, dos quais os dados que interessam às aplicações podem ser obtidos. ***Computações***, representando implementações de algoritmos paralelos que exploram os recursos de uma classe de plataformas virtuais. ***Conectores***, que acoplam computações e fontes de dados alocadas em plataformas virtuais distintas. ***Ligações de Serviço***, que conectam pares formados por uma porta usuária e uma porta provedora pertencentes a componentes distintos, para que um componente usuário possa consumir um serviço oferecido por um componente provedor. ***Ligação de Ações***, que vinculam um conjunto de portas de ações pertencentes a computações, conectores e ao componente workflow. Finalmente, ***Qualificadores*** e ***Quantificadores***, que são usados em contratos contextuais para representar suposições e restrições funcionais e não-funcionais na implementação de componentes de solução.

O componente *application* é uma abstração para o *frontend* da aplicação. Seu papel é intermediar a comunicação entre os componentes de solução e o *frontend* da aplicação por meio de ligações de serviço, fornecendo entradas, recebendo saídas, gerenciando dados intermediários, monitorando componentes da solução, etc. Por sua vez, o componente *workflow* orquestra computações e conectores ao longo da execução por meio de ligações de ações. Para isso, as portas vinculadas por meio de uma ligação de ações carregam um conjunto comum de *nomes de ações*. Os componentes ativam ações em suas portas de ações fazendo referência a nomes de ações. Uma ativação para uma ação de nome n em uma porta p permanece bloqueada até que uma ativação pendente de n exista em cada porta vinculada a p por meio de uma ligação de ações.

Com o objetivo de demonstrar seus principais conceitos, apresentamos um exemplo de sistema de computação paralela da HPC Shelf na Figura 2, construído a partir de um arcabouço de componentes para computação MapReduce [187] cujos principais componentes são descritos na Tabela 5. Ele implementa uma computação em três estágios MapReduce que enumera os triângulos em um grafo. O grafo é obtido de um repositório de dados representado por um componente de origem (**source**), por meio de uma ligação de serviço chamada **input_data**. Por sua vez, a saída (um único inteiro) é enviada para o *frontend* da aplicação (componente *application*) por meio de outra ligação de serviço chamada **output_data**. A figura enumera as associações de serviço (1-9) para comunicação de pares chave/valor ao longo dos estágios de computação, bem como associações de ação (9-23) para orquestração por parte do componente *workflow*.

Em um sistema de computação paralela, uma instância de um componente de computação é associada a uma plataforma virtual onde ele será executado, formando o que chamamos de *componente sistema*. A noção de componente sistema é central para o Alite, uma vez que um componente de computação é desenvolvido supostamente sob fortes suposições acerca das características da(s) plataforma virtual(is) sobre a qual(is) ele poderá ser implantado. No exemplo, $\mathbf{reducer_0}$, $\mathbf{reducer_1}$ e $\mathbf{reducer_2}$ são componentes de computação do tipo REDUCER, cada um alocado em uma plataforma virtual distinta, chamada $\mathbf{platform\_reduce_0}$, $\mathbf{platform\_reduce_1}$ e $\mathbf{platform\_reduce_2}$, respectivamente. Assim, cada par $\mathbf{reducer_i}$/$\mathbf{platform\_reduce_i}$, para $i \in \{0, 1, 2\}$ representa um componente do sistema. O componente **source** também é alocado em uma plataforma virtual chamada **platform_source**, onde os dados são diretamente acessíveis pelo componente **source**.

Um conector pode desempenhar duas funções em um sistema de computação paralela. Eles podem *orquestrar* computações e outros conectores por meio de ligações de ações, assim como o componente *workflow*, ou oferecer o suporte para coreografias entre computações, fontes de dados e outros conectores por meio de ligações de serviços (por exemplo, o papel de um módulo acoplador em um software de simulação de um modelo climático global [37]). Para isso, eles compreendem um conjunto de *facetas*, cada uma alocada em uma plataforma virtual onde um componente que ele acopla encontra-se alocado, permitindo a comunicação por meio de ligações diretas de serviços. No exemplo, existem quatro conectores de dois tipos diferentes: $\mathbf{shuffler_0}$, $\mathbf{shuffler_1}$, $\mathbf{shuffler_2}$ e **splitter**. O tipo dos três primeiros conectores é SHUFFLER, enquanto o tipo do quarto é SPLITTER. Ambos os tipos de conectores suportam coreografias entre componentes de estágios de computação adjacentes implementando protocolos distintos para passar os pares chave/valor de saída produzidos em um estágio para o próximo, explicado na Tabela 5. Para isso, eles têm dois conjuntos de facetas, conforme descrito na Figura 3: um conjunto de *facetas coletoras*, através das quais recebe *chunks* de pares chave/valor do estágio anterior através de portas de serviço usuárias chamadas **collect_pairs**, e um conjunto de *facetas alimentadoras*, através do qual ele envia *chunks* de pares de chave/valor para o próximo

espécie	tipo	pares de entrada	pares de saída	significado
computações	MAPPER	[(K$_1$,V$_1$)]	[(K$_2$,V$_2$)]	*Usa uma função de mapeamento para produzir uma lista de pares de saída, do tipo* (K$_2$,V$_2$)*, para cada par de entrada do tipo* (K$_1$,V$_1$)*.*
	REDUCER	[(K$_1$,[V$_1$])]	[(K$_2$, V$_2$)]	*Usa uma função de redução para reduzir uma lista de valores do tipo* [V$_1$] *de um pair de entrada em um valor único do tipo* V$_2$*.*
conectores	SPLITTER	[(K,V)]	[(K,V)]	*Distribui pares de entrada recebidos das facetas coletoras entre as facetas alimentadoras, usando uma função de particionamento.*
	SHUFFLER	[(K,V)]	[(K,[V])]	*Análogo ao* SPLITTER*. Porém, os valores associados a mesma chave são agrupados em um único par de saída.*

Tabela 5: Componentes do Arcabouço MapReduce da HPC Shelf

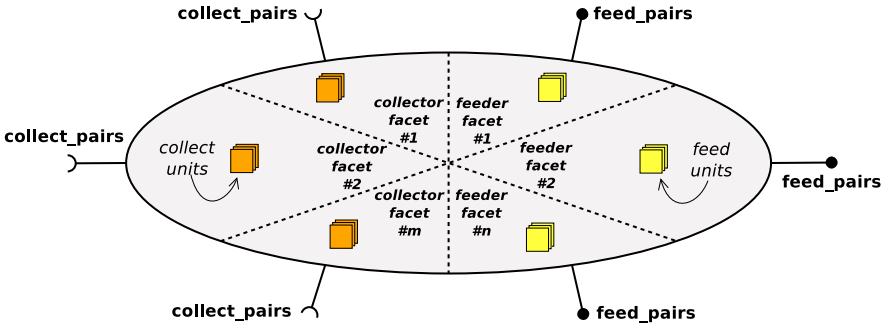

Figura 3: Multiple Facets of Splitter and Shuffler

porta de ações	nomes	significado
task_reduce	`read_chunck`	um novo *chunk* pode ser lido de **collect_pairs**
	`finish_chunck`	não há mais *chunks* a serem lidos de **collect_pairs**
	`perform`	executa a redução
	`chunk_ready`	um *chunk* está pronto na porta **feed_pairs**
task_collector_read_chunk	`read_chunk`	um novo *chunk* pode ser lido de **collect_pairs**
	`finish_chunck`	não há mais *chunks* a serem lidos de **collect_pairs**
task_feeder_read_chunk	`read_chunk`	um novo *chunk* pode ser lido de **collect_pairs**
	`finish_chunck`	não há mais *chunks* a serem lidos de **collect_pairs**
task_feeder_chunk_ready	`chunk_ready`	um *chunk* está pronto na porta **feed_pairs**

* `read_chunk` e `finish_chunk` são ações alternativas, que servem para sinalizar a terminação de iterações e escolhas entre ramos condicionais em protocolos de orquestração em SAFeSWL.

Tabela 6: Nomes de Ações de instâncias de Reducer, Shuffler e Splitter

estágio por meio de portas de serviço provedoras chamadas **feed_pairs**. Há uma única faceta coletora e outra alimentadora nos conectores do exemplo, uma vez que um único componente redutor compõe cada estágio. Como os mapeadores e redutores também possuem porta **collect_pairs** e **feed_pairs**, como mostrado na Tabela 5, eles podem ser conectados em arquiteturas complexas através de *shufflers* e *splitters* com o papel de intermediários entre os estágios de computação, como no pipeline de três estágios do sistema de contagem de triângulos, representado na Figura 2. De fato, cada estágio de computação poderia envolver um regimento de mapeadores ou redutores em vez de um único. Cálculos iterativos também são suportados. Na verdade, o arcabouço MapReduce da HPC Shelf foi projetado para lidar com limitações de expressividade bem conhecidas de estruturas MapReduce existentes [152].

O leitor pode notar o posicionamento das facetas dos *shufflers* e *splitters* em plataformas virtuais distintas. Por exemplo, a faceta coletora única de $\mathbf{shuffler_1}$ é alocada em $\mathbf{platform\_reduce_0}$ enquanto sua faceta alimentadora é colocada em $\mathbf{platform\_reduce_2}$, para que possa se comunicar através de ligações diretas com $\mathbf{reducer_0}$ e $\mathbf{reducer_1}$. De fato, toda a lógica de comunicação/sincronização entre os estágios de computação executados por $\mathbf{reducer_0}$ e $\mathbf{reducer_1}$ encontra-se encapsulada dentro de $\mathbf{shuffler_1}$. Essa separação de interesses na implementação de cálculos paralelos é um recurso de modularidade importante que é consequência do emprego do modelo Hash.

No exemplo, os componentes redutores têm uma única porta de ação chamada **task_reduce**, enquanto os *shufflers* e o *splitter* têm três portas de ação: uma delas na faceta coletora, nomeada **task_collector_read_chunk**, e duas outras na faceta alimentadora, nomeadas **task_alimentador_ler_chunk** e **task_feeder_chunk_ready**. Eles são renomeados na configuração para evitar conflitos de nomes. A Tabela 6 lista os nomes das ações dessas portas de ação e seus respectivos significados.

Cada componente de solução possui uma porta de ação predefinida, chamada de *porta de ciclo de vida*, conectada ao componente workflow, incluindo os seguintes nomes de ação: `resolve`, para acionar o procedimento de seleção de uma implementação de componente com base no contrato contextual vinculado ao componente; `deploy`, para implantar o componente previamente selecionado em uma plataforma virtual; `instantiate`, para criar uma instância do componente; `run`, para iniciar a execução de uma instância de componente; e `release`, para liberar os recursos usados por uma instância de componente, quando não mais necessária. Nos protocolos de orquestração executados pelo componente *workflow*, as portas de ciclo de vida possibilitam a criação, resolução, implantação, instanciação e liberação sob demanda (preguiçosa) de componentes de solução em paralelo com as computações, permitindo um maior controle sobre o uso de recursos baseados em nuvem em computações de longa duração, principalmente quando recursos de alto custo não são necessários durante toda a execução. Além disso, operações de longa duração típicas de ambientes de nuvem, como a criação de plataformas virtuais, bem como específicas da própria HPC Shelf, como a resolução de contratos complexos, podem ser sobrepostas com computações.

No sistema de enumeração de triângulos, o componente *workflow* orquestra os nomes das ações por meio de um protocolo complexo que apresentaremos em partes para simplificar. Inicialmente, ele executa a sequência de ativação das ações do ciclo de vida para todos os componentes necessários desde o início do cálculo (nesse caso, as únicas exceções são **splitter** e **output**) usando o seguinte protocolo especificado em SAFeSWL (SAFe Scientific Workflow Language), uma linguagem auto-explicativa baseada em XML para descrever arquiteturas de sistemas de computação paralela e orquestrações de fluxo de trabalho (*workflow*) [91]. No código, `<component_id>` representa o identificador do componente[10]:

```
0  <sequence>
1      <invoke port=<component_id> action="resolve"/>
2      <invoke port=<component_id> action="deploy"/>
3      <invoke port=<component_id> action="instantiate"/>
4  </sequence>
```

Primeiro, todas as plataformas virtuais são instanciadas em paralelo, usando o combinador `<parallel>...</parallel>`. Em seguida, são instanciada as computações e os conectores, que exigem a instanciação anterior das plataformas virtuais onde encontram-se alocados. Por fim, as ligações, que devem ser conectadas a portas de computações e conectores.

Depois de todos os componentes terem sido instanciados, a ação `run` é ativada para os *shufflers* e redutores, e a orquestração das ações computacionais é iniciada. Ele consiste em quatro iterações, uma para cada estágio de computação (*shuffle*/*reduce*) e uma para o estágio de saída do pipeline (*split*). As iterações são executadas em paralelo porque todos os estágios são ativos durante a computação. Por exemplo, o código de iteração correspondente ao i-ésimo estágio de computação do pipeline, onde $i = \{0, 1, 2\}$, é descrito abaixo:

```
0  <sequence>
1    <iterate port="task_shuffle_collector_read_chunk_i" until="FINISH_CHUNK" loop="READ_CHUNK">
2      <sequence>
3          <invoke port="task_shuffle_feeder_read_chunk_i"  action="READ_CHUNK" />
4          <invoke port="task_shuffle_feeder_chunk_ready_i" action="CHUNK_READY" />
5          <invoke port="task_reduce_i" action="READ_CHUNK" />
6          <invoke port="task_reduce_i" action="PERFORM" />
7      </sequence>
8    </iterate>
9    <invoke port="task_shuffle_feeder_read_chunk_i" action="FINISH_CHUNK" />
10   <invoke port="task_reduce_i" action="FINISH_CHUNK" />
11   <invoke port="task_reduce_i" action="CHUNK_READY" />
```

[10]Por padrão, o nome da porta de ação do ciclo de vida de um componente é o nome do próprio componente.

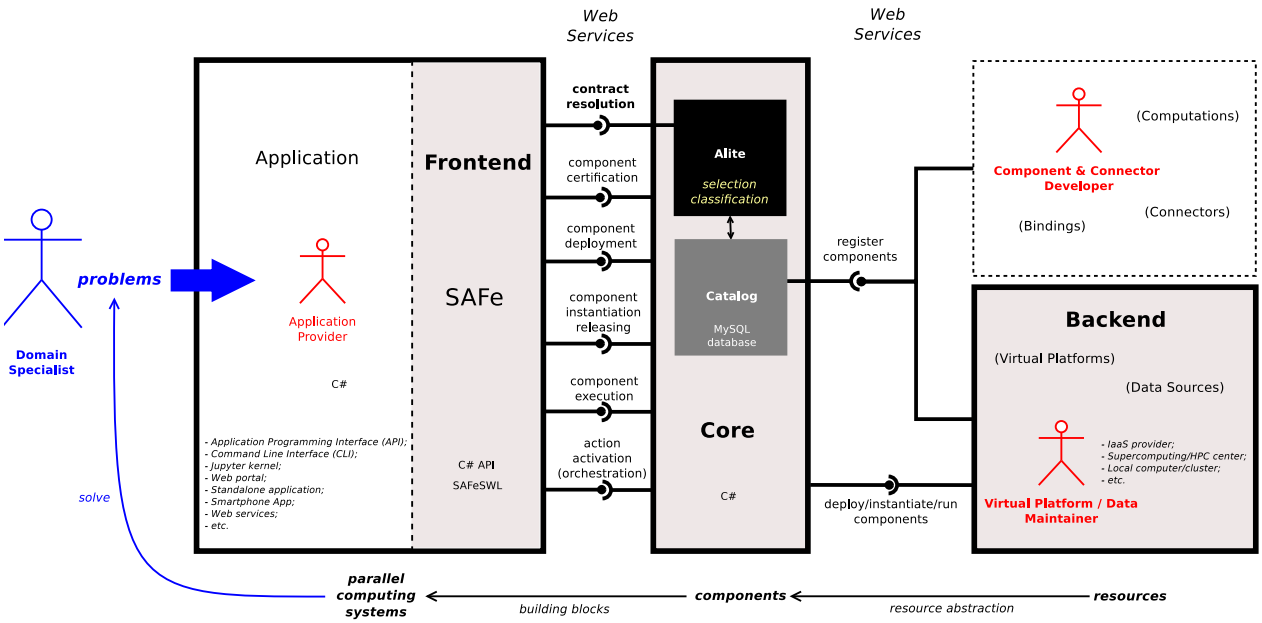

Figura 4: Intervenientes e Elementos Arquiteturais da HPC Shelf

```
12 </sequence>
```

As ações de ciclo de vida dos componentes **splitter** e **output** são acionadas no início da quarta iteração (estágio de saída), pois são necessárias apenas no final da computação. Depois que todas as iterações tiverem sido encerradas e toda a saída tiver sido enviada para o componente *application*, todos os componentes são liberados.

9.6.3 Intervenientes

Os seguintes intervenientes atuam em torno da HPC Shelf. Especialistas de domínio são os usuários finais, enquanto os outros criam, implantam e gerenciam os recursos necessários para que as aplicações possam atender aos seus especialistas alvo. Os especialistas usam aplicações para especificar e resolver problemas usando uma interface de alto nível. Assim, não precisam estar cientes da natureza baseada em componentes dos recursos a eles oferecidos, nem mesmo do fato de que a solução computacional para os problemas que desejam resolver será executada por um sistema de computação paralela. Por outro lado, ***provedores de aplicação*** conhecem as melhores soluções computacionais para os problemas especificados por especialistas de domínio. Assim, sabem quais componentes oferecidos pela HPC Shelf fornecem os recursos de software e hardware necessários para montar o sistema de computação paralela apropriado para resolver cada problema. No entanto, não são especialistas em computação paralela, ao contrário de ***desenvolvedores de componentes***, os quais possuem habilidades de programação paralela e conhecimento técnico para explorar o desempenho de plataformas virtuais de forma eficiente. Por sua vez, ***mantenedores de plataformas*** oferecem a infraestrutura de computação paralela da HPC Shelf, implantando plataformas virtuais sobre uma determinada infraestrutura de computação paralela. Por fim, ***gerentes de dados*** oferecem grandes repositórios de dados que interessam aos aplicações, na forma de componentes de fonte de dados.

Vale ressaltar que essa caracterização de intervenientes não é rígida. Ela existe apenas para promover separação de interesses. Por exemplo, na aplicação Swirls, os especialistas são usuários de programas MPI, o que inclui desenvolvedores desses programas, os quais lidam diretamente com a construção dos sistemas de computação paralela necessários para cumprir os seus propósitos. Nesse caso, possuem características tanto de especialistas de domínio quanto de desenvolvedores de componentes e provedores de aplicações.

9.6.4 Arquitetura

A arquitetura da HPC Shelf é baseada em três elementos arquiteturais, retratados na Figura 4: *Frontend*, *Core* e *Backend*. A figura também ilustra sua relação com os intervenientes descritos na Seção 9.6.3.

O *Frontend* para aplicações é o SAFe (*Shelf Application Framework*) [91], um arcaboço que os provedores de aplicações usam para construir e executar sistemas de computação paralela exigidos por suas aplicações. O SAFe suporta a linguagem SAFeSWL como uma linguagem de descrição arquitetural e de orquestração para tal finalidade. Além disso, ele suporta uma API[11] para C#.

Por sua vez, o *Core* implementa o catálogo de componentes, onde desenvolvedores e mantenedores registram componentes, bem como o Alite. As aplicações acessam os serviços *Core*, por meio de SAFe, para resolver contratos contextuais e controlar o ciclo de vida dos componentes selecionados. Depois de instanciados, as aplicações orquestram diretamente os componentes.

Finalmente, um *Backend* é um serviço que um mantenedor de plataformas oferece ao *Core* para instanciar plataformas virtuais sobre a infraestrutura de computação paralela sob sua autoridade. Uma vez instanciadas, as plataformas virtuais podem se comunicar diretamente com *Core* para instanciar componentes. Serviços *Backend* para fornecer acesso a grandes repositórios de dados também podem ser possíveis, fornecidos pelos gerentes de dados. Na verdade, os componentes fonte de dados são implementados como plataformas virtuais a partir das quais os dados podem ser acessados diretamente por meio de portas de serviço.

9.6.5 Alite, o Sistema de Resolução de Contratos Contextuais

Os sistemas de computação paralela referem-se a cada instância de componente de solução que eles exigem por meio de um *contrato contextual*. O sistema de contratos contextuais é denominado Alite.

O Alite é um tipo de sistema de gerenciamento de recursos projetado para os requisitos de HPC Shelf, onde os recursos são os elementos constituintes dos sistemas de computação paralela, representados por componentes. Os contratos contextuais podem ser formalizados sob a teoria de sistemas de tipos polimórficos de alta ordem com tipos universais e existenciais [184].

Além dos *contratos contextuais*, o Alite introduz os conceitos precedentes de *componentes abstratos* e *assinaturas contextuais* . Um componente abstrato representa um conjunto de componentes, também conhecidos como *componentes concretos*, que implementam um mesmo interesse de software sob diferentes suposições acerca dos requisitos da aplicação e das características do ambiente de execução. Tais suposições são representadas por uma assinatura contextual, definida por um conjunto de *parâmetros de contexto*.

Um *contrato contextual* é um mapeamento parcial[12] de argumentos de contexto aos parâmetros de contexto de um componente abstrato. Um parâmetro de contexto é definido por um *nome*, uma *restrição* e uma *direção*. A restrição de um parâmetro de contexto é um contrato contextual que define uma restrição no conjunto de argumentos de contexto que podem ser atribuídos a ele em um contrato contextual. Por sua vez, a direção pode ser *covariante* (<:) ou *contravariante* (:>). É contravariante se o parâmetro denota uma suposição que o componente faz sobre o ambiente contextual. O ambiente contextual é definido pela aplicação que ele atende e pela plataforma virtual em que ele é executado. Por sua vez, é covariante quando denota uma suposição que o ambiente contextual faz a respeito do componente.

Além de uma assinatura contextual, a especificação de um componente abstrato também inclui um conjunto de *unidades* e um conjunto de *componentes aninhados*, conforme exigido pelo modelo Hash. Um contrato contextual é vinculado a cada componente aninhado, possivelmente se referindo aos parâmetros de contexto do componente hospedeiro como argumentos.

A Figura 5 apresenta uma sintaxe abstrata para assinaturas contextuais e contratos contextuais, onde usamos nomes autoexplicativos para variáveis gramaticais. Porém, abstrai-se de unidades e componentes aninhados, uma vez que não são relevantes para o processo de resolução de contratos de interesse do Alite, tratado nesta seção. Pode-se notar que cada assinatura contextual possui um parâmetro chamado ***name***, que deve ser fornecido em contratos contextuais. Seu limite é um qualificador que representa o nome do

[11] Application Programming Interface.

[12] Em um mapeamento parcial, não é obrigatório atribuir argumentos de contexto a todos os parâmetros de contexto do componente abstrato.

$\langle signature \rangle$ ::= **[** $\langle constraint\_list \rangle$ **]** | ε
$\langle constraint\_list \rangle$ ::= $\langle naming\_parameter \rangle$ **,** $\langle constraint\_list \rangle$ | $\langle constraint \rangle$
$\langle constraint \rangle$ ::= $\langle parameter \rangle$ | $\langle parameter\_binding \rangle$
$\langle parameter\_binding \rangle$ ::= $\langle parameter\_id \rangle$ == $\langle parameter\_id \rangle$
$\langle parameter\_id \rangle$ ::= PARAMETER_ID (- PARAMETER_ID)*
$\langle parameter \rangle$ ::= PARAMETER_ID $\langle subtyping\_direction \rangle$ $\langle contract \rangle$
$\langle naming\_parameter \rangle$ ::= ***name*** :> QUALIFIER_ID
$\langle subtyping\_direction \rangle$ ::= :> | <:

$\langle contract \rangle$ ::= **[** $\langle argument\_list \rangle$ **]**
$\langle argument\_list \rangle$ ::= $\langle naming\_argument \rangle$ (**,** $\langle argument \rangle$)*
$\langle argument \rangle$ ::= $\langle parameter\_id \rangle$ = $\langle contract \rangle$
$\langle naming\_argument \rangle$ ::= ***name*** = QUALIFIER_ID

Figura 5: Sintaxe Abstrata para Assinaturas e Contratos Contextuais

componente no catálogo de componentes, chamado *qualificador de nome*. Como um açúcar sintático para simplificar a especificação de assinaturas contextuais, [***name*** :> QUALIFIER_ID, $\cdots$] pode ser escrito como QUALIFIER_ID[$\cdots$]. Além disso, [***name*** :> QUALIFICADOR_ID] pode ser escrito simplesmente como QUALIFICADOR_ID. Uma forma abreviada análoga é válida para contratos contextuais, trocando ':>' por '='.

O sistema de tipos de componentes por trás de Alite define uma relação de subtipagem indutivamente a partir de uma relação de subtipagem nominal entre qualificadores de nome, bem como uma regra de subtipagem estrutural para contratos contextuais. Por exemplo, seja Γ uma relação de subtipagem entre qualificadores de nome, ou seja, incluindo pares $n_1 <: n_2$ (i.e., n_1 é um subtipo de n_2). Além disso, seja $\mathcal{C}$ um conjunto bem formado de assinaturas contextuais de componentes abstratos registrados no catálogo de componentes. Dizemos que um contrato é válido em relação a $\mathcal{C}$ se

$$\frac{\begin{array}{c}\exists \langle signature \rangle \in \mathcal{C} \ s.t. \ \langle contract \rangle(\textbf{\textit{name}}) = \langle signature \rangle(\textbf{\textit{name}}) \ \wedge \\ \forall p = a \in \langle contract \rangle : \exists p \ \square \ a' \in \langle signature \rangle \ s.t. \ \Gamma, \mathcal{C} \vdash a \ \square \ a', \text{ where } \square \in \{<:, :>\}\end{array}}{\mathcal{C} \vdash \langle contract \rangle}$$

, ou seja, os argumentos contratuais devem respeitar as restrições impostas pelos limites de parâmetros por meio de uma relação de subtipagem entre contratos contextuais válidos, definido, indutivamente, pelas regras

$$\frac{\mathcal{C} \vdash \langle contract \rangle_1 \quad \mathcal{C} \vdash \langle contract \rangle_2 \qquad \forall \langle p = a_1, p = a_2 \rangle \in \langle contract \rangle_1 \times \langle contract \rangle_2 : \begin{cases} \Gamma, \mathcal{C} \vdash a_1 <: a_2 & \text{if } p \textit{ is covariant} \\ \Gamma, \mathcal{C} \vdash a_1 :> a_2 & \text{if } p \textit{ is contravariant} \end{cases}}{\Gamma, \mathcal{C} \vdash \langle contract \rangle_1 <: \langle contract \rangle_2}$$

e

$$\frac{\text{QUALIFIER\_ID}_1 <: \text{QUALIFIER\_ID}_2 \in \Gamma}{\Gamma, \mathcal{C} \vdash \text{QUALIFIER\_ID}_1 <: \text{QUALIFIER\_ID}_2}$$

, onde o primeiro define a relação de subtipagem entre contratos contextuais e o último vincula a relação de subtipagem entre qualificadores de nome (Γ). Por fim, assinaturas contextuais devem satisfazer a restrição

$$\begin{array}{c}\{\langle signature \rangle_1, \langle signature \rangle_2\} \subseteq \mathcal{C} \wedge \langle signature \rangle_1(\textbf{\textit{name}}) <: \langle signature \rangle_2(\textbf{\textit{name}}) \in \Gamma \\ \implies \forall \langle p \ \square \ b_1, p \ \square \ b_2 \rangle \in \langle signature \rangle_1 \times \langle signature \rangle_2 : \begin{cases} \Gamma, \mathcal{C} \vdash b_1 <: b_2 & \text{if } p \textit{ is covariant} \\ \Gamma, \mathcal{C} \vdash b_1 :> b_2 & \text{if } p \textit{ is contravariant} \end{cases}\end{array}$$

name	bound	meaning
input_key_type	<: Data	K_1
input_value_type	<: Data	V_1
function	<: Function	F
output_key_type	<: Data	K_2
output_key_value	<: Data	V_2
⋮	⋮	⋮

Mapper/Reducer

name	bound	meaning
key_type	<: Data	K
value_type	<: Data	V
partition_function	<: PartitionFunction	F
⋮	⋮	⋮

Splitter/Shuffler

Tabela 7: Contextual Signature of MapReduce components

name	argument
input_key_type	Vertex
input_value_type	DataGraph
function	GustyFunction
output_key_type	Vertex
output_key_value	DataGraph

(a) Contextual Contract of Gusty

name	argument
input_key_type	Vertex
input_value_type	DataTriangle
function	TC0
output_key_type	Vertex
output_key_value	DataTriangle

(b) Inquire Contextual Contract of **reducer$_0$**

Tabela 8: Exemplos de Contratos Contextuais

, a qual estabelece que se o nome de duas assinaturas (por exemplo, n_1 e n_2) estão relacionados em Γ (por exemplo, $n_1 <: n_2$), os limites de seus parâmetros correspondentes devem respeitar a relação de subtipagem entre contratos contextuais.

Exemplo A Tabela 7 apresenta alguns parâmetros de contexto contravariantes das assinaturas contextuais dos componentes MapReduce. Através deles, as implementações de Mapper, Reducer, Splitter e Shuffler podem ser ajustadas de acordo com os tipos de dados de chaves e valores que recebem de **collect_pairs** ports (Tabela 5), bem como o tipo de funções que definem seu comportamento específico (ou seja, mapear, reduzir e particionar funções). Alite seria responsável por decidir qual é a implementação mais apropriada entre as disponíveis. A implementação atual do arcabouço MapReduce da HPC Shelf fornece implementações genéricas únicas para cada componente abstrato, ou seja, cujos parâmetros de contexto dos contratos estão associados às suas próprias restrições como argumentos. Além disso, há uma implementação específica de Reducer para computações em grafos, como o sistema de enumeração de triângulos, chamado de Gusty, o qual implementa o contrato apresentado na Tabela 8 (a). Gusty é o principal componente do Gust[187], uma arcabouço para computação em grafos grandes que desenvolvemos como uma extensão do arcabouço MapReduce, como resultado de uma Tese de Doutorado . A Tabela 8(b) apresenta o contrato contextual vinculado a **reducer$_0$** no sistema de enumeração de triângulos, o qual orienta o Alite na seleção do componente Gusty como a implementação mais adequada, uma vez que apresenta os pressupostos de implementação mais fortes que são compatíveis com o contrato de **reducer$_0$**. Para isso, devido à contravariância dos parâmetros de contexto, é necessário que DataTriangle seja um subtipo de DataGraph e TC0 seja um subtipo de GustyFunction.

O sistema de resolução de contratos do Alite é capaz de ranquear pares formados por um componente de computação e a plataforma virtual sobre a qual encontra-se alocado em um sistema de computação paralela, os quais satisfazem ao mesmo tempo os requisitos de seus contratos no sistema, tanto da computação quanto da plataforma virtual, de acordo com uma política de alocação de recursos. Para isso, o Alite introduz três características importantes para garantir a expressividade na especificação de contratos contextuais e a

eficácia de seu sistema de resolução de contratos: a abstração de componente de sistema, classes de parâmetros de contexto e a espécie de componentes dos *quantificadores*.

Componentes de Sistema

O sistema de resolução de contratos contextuais é projetado para selecionar e ranquear componentes de sistema, os quais consistem de pares formados por um componente de computação e sua plataforma virtual hospedeira. Logo, é aplicado sobre um par formado pelo contrato contextual de um componente de computação e outro contrato contextual de uma plataforma virtual alvo. A idéia da resolução simultânea de um contrato de componente de computação e sua plataforma virtual hospedeira vem de que a implementação do componente de computação deve ser realizada de acordo com as características arquiteturais da plataforma virtual alvo de modo a utilizar de forma mais eficiente os recursos por ela oferecidos. No contexto da computação heterogênea, esse requisito torna-se ainda mais importante, bem como levar em consideração que os recursos dispostos em nuvens computacionais comerciais possuem um custo associado à sua utilização.

Classes de Parâmetros de Contexto

Componentes de computação são desenvolvidos para atender aos requisitos de uma aplicação em relação aos seus interesses, explorando as características de plataformas virtuais alvo a fim de satisfazer requisitos de QoS (*Quality-of-Service*) impostos pela aplicação, porém sujeito a restrições de custo associado aos recursos em nuvens computacionais comerciais. Por essa razão, cada parâmetro de contexto é classificado como de *aplicação*, *plataforma*, *qualidade* ou *custo*. Tal caracterização é levada em consideração pelo sistema de resolução de contratos. As aplicações podem forçar estaticamente restrições de qualidade (QoS) e custo através de contratos contextuais satisfeitos pelos componentes selecionados. Por sua vez, para os componentes selecionados pelo Alite, argumentos de qualidade e de custo podem ser calculados usando modelos analíticos de desempenho e custo, baseado nos outros argumentos especificados no contrato sob resolução.

Componentes Quantificadores

Com o propósito de suportar valorações numéricas em parâmetros de contexto, o Alite introduz a espécie dos componentes quantificadores, com quatro domínios predefinidos, representando números inteiros e reais como instâncias de componentes: Int $\downarrow$ and Real $\downarrow$, para relações de subtipagem direta, i.e., $n <: m \Leftrightarrow n \leq m$, onde n e m são quantificadores; Int $\uparrow$ e Real $\uparrow$, para relações de subtipagem inversa, i.e., $n <: m \Leftrightarrow m \leq n$. Dessa forma, sejam N e N' quantificadores do tipo Int $\downarrow$ (Real $\downarrow$). N' só pode ser usado quando um quantificador N é requisitado ($N' <: N$) se $N' \leq N$ (relação direta). De forma análoga, se são quantificadores do tipo Int $\uparrow$ (Real $\uparrow$), é possível apenas se $N' \geq N$ (relação inversa). Assume-se que Int $\downarrow \subset$ Real $\downarrow$ e Int $\uparrow \subset$ Real $\uparrow$. Além disso, dizemos que $+\infty$ e $-\infty$ são os quantificadores no topo de Int $\downarrow$ (Real $\downarrow$) and Int $\uparrow$ (Real $\uparrow$), respectivamente, ou seja, são supertipos de qualquer quantificador nos seus respectivos domínios. A seguir, apresentamos alguns exemplos intuitivos sobre o emprego de quantificadores como restrições de parâmetros e contexto.

Inicialmente, seja um parâmetro de contexto covariante, chamado ***efficiency***, que representa a eficiência de um algoritmo paralelo, medido no intervalo $[0.0, 1.0]$. Um componente de computação que garante eficiência de 0,7 (70%) pode ser selecionado em um contexto onde é requisitado um componente com eficiência de 0,6 (60%), uma vez que $0.7 <: 0.6$ pela covariância do parâmetro em questão. Logo, a restrição desse parâmetro de contexto deve ser 0.0 no domínio Real $\uparrow$. Porém, tal domínio poderia ser Int $\uparrow$ caso a eficiência fosse medida em termos percentuais, ou seja, por um valor no intervalo $[0, 100]$.

Agora, seja ***max_processing_nodes*** um parâmetro contravariante que representa o número máximo de nós de processamento recomendados para um componente de computação na sua plataforma virtual hospedeira. Se ***max_processing_nodes*** $= N$ para uma implementação de um componente de computação, ele poderia ser selecionado em um contexto pode é requisitado uma computação com ***max_processing_nodes*** $=$

M, onde $M \leq N$. Dessa forma, a restrição de ***max_ processing_ nodes*** seria $+\infty$ no domínio Int $\downarrow$. Analogamente, para o número mínimo de nós, e.g., ***min_ processing_ nodes***, a restrição e seu domínio seriam 1 e Int $\uparrow$, respectivamente.

Analogamente, o sistema de resolução deve selecionar uma plataforma virtual cujos nós de processamento tenham N núcleos em um contexto onde é requisitada uma plataforma virtual com menos que N núcleos de modo que haveriam núcleos suficientes para atender aos requisitos do contrato. Dessa forma, para um parâmetro de contexto representando o número de núcleos oferecidos pelos nós de processamento da plataforma virtual, o limite seria 1 no domínio Int $\uparrow$.

Um parâmetro de contexto para latência de interconexão entre os nós de uma plataforma virtual (e.g., ***network_ latency***), medido em nanosegundos, também possui Int $\downarrow$ como seu domínio e $+\infty$ como sua restrição, de modo que uma plataforma virtual com uma latência menor (melhor) que a requisitada pelo contrato possa ser selecionada. Entretanto, não seria razoável escolher sempre aquela plataforma virtual, dentre as compatíveis, que ofereça a menor latência possível ? Não necessariamente, se restrições de custo são consideradas dentre os parâmetros de contexto do contrato. Nesse caso, a escolha de uma plataforma virtual "melhor", com menor latência na interconexão entre os nós de processamento, porém com um custo maior, seria restringida de forma a não violar as restrições de custo expressas no contrato contextual.

Finalmente, seja um parâmetro para a capacidade de computação (*compute capability*) das GPUs oferecidas pelos nós de uma plataforma virtual, restrigindo a versão de CUDA suportado pelos componentes de computação que usam recursos de aceleração. Seu domínio poderia ser Int $\uparrow$, uma vez que uma plataforma virtual com GPUs de capacidade 5 (valor mais alto) poderia ser usada em um contexto onde uma capacidade 4 é requisitada (valor mais baixo). Um componente de computação selecionado para executar nessa plataforma virtual seria implementada sob suposições de capacidade de computação igual ou menor que 4, de modo que seria seguro selecionar uma plataforma virtual com capacidade de computação maior (e.g., 5).

Algoritmo de Resolução

O algoritmo de resolução de contratos contextuais não será detalhado nesse documento. Ele é executado para cada plataforma virtual, considerando os componentes de computação hospedados, e é composto de duas etapas. Na etapa de *seleção*, são selecionados componentes de sistemas que atendem aos requisitos especificados no contrato, chamados de *componentes de sistema candidatos*. Na etapa de *ranqueamento*, os candidatos são ordenados segundo um conjunto de parâmetros de ranqueamentos. No artigo que descreve o Alite [87], cuja leitura recomendamos para aprofundamenton neste tópico, são usados três parâmetros em um estudo de caso: ***execution_ time***, ***monetary_ cost***, and ***energy_ consumption***, cujos nomes são autoexplicativos. Tais parâmetros são associados a outros parâmetros que determinam suas ponderações, que afetam o ranqueamento priorizando um ou outro parâmetro de ranqueamento: ***execution_ time_ weight***, ***monetary_ cost_ weight***, and ***energy_ consumption_ weight***. Métodos de tomada de decisão multicritério (MCMD) [192] foram então empregados para o ranqueamento.

9.6.6 Swirls

Swirls é o nome de uma aplicação de propósito geral da HPC Shelf destinada à construção, implantação e execução interativa de sistemas de computação paralela por meio de uma interface REPL (*read-eval-print loop*). Tal interface encontra-se implementada sobre notebooks Jupyter[13] e terminais Linux (*shell*).

A Swirls oferece as seguintes características:

- acesso ao catálogo de componentes de qualquer serviço *Core*;
- uma linguagem de comandos para construção, implantação, execução e acompanhamento de execução de sistemas de computação paralela em geral;

[13] `http://www.jupyter.org`

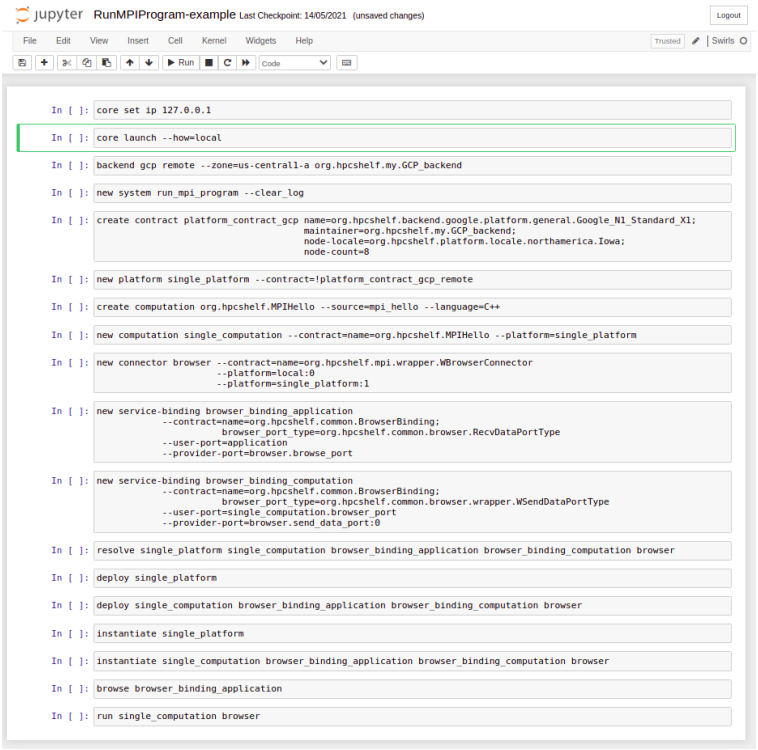

Figura 6: Exemplo de comandos Swirls para execução de um programa MPI simples

- serviços backend para dois provedores IaaS: Amazon Elastic Compute Cloud (EC2) and Google Cloud Platform (EC2);
- encapsulamento de programas MPI escritos em C, C++, e Python em componentes de computação;
- um conector de passagem de mensagens para comunicação entre programas MPI alocados a plataformas virtuais distintas (suporte ao paralelismo multicluster com programas MPI).

Dessa forma, é possível implantar sistemas de computação paralela formados por múltiplas plataformas virtuais (multicluster), possivelmente alocadas em infraestruturas de computação paralela distintas (multi-cloud), onde instâncias de componentes de computação, possivelmente encapsulando programas MPI pré-existentes, podem envolver-se em processamento paralelo de larga escala, agregando o poder computacional das plataformas virtuais onde encontra-se alocados. É possível também instanciar plataformas virtuais no próprio computador local onde a aplicação estiver sendo executada.

Para suportar EC2 e GCP, a Swirls oferece dois arcabouços de componentes para construção de plataformas virtuais a partir de tipos de instâncias/máquinas suportados por tais provedores IaaS. Usando essa mesma abordagem, é possível implementar, com relativa facilidade, o suporte a outros provedores IaaS que disponibilizem uma API[14] para acesso aos seus serviços. Além disso, é possível construir serviços Backend para clusters locais, inclusive para acesso não-virtualizado[15], consequência direta da HPC Shelf.

Instruções em como instalar e usar o Swirls podem ser encontrados em `https://www.hpcshelf.org/#h.8eglmidqf5uh`. Todos os códigos fontes são abertos para serem usados e modificados por qualquer interessado, com o único requisito de fazer referência aos trabalhos dos autores original em publicações.

[14] *Application Programming Interface*

[15] Atualmente, possuímos um serviço backend capaz de instanciar plataformas virtuais sobre clusters OpenStack, usado no cluster do LIA. Porém, os nós de processamento são núcleos de um multiprocessador, em memória compartilhada.

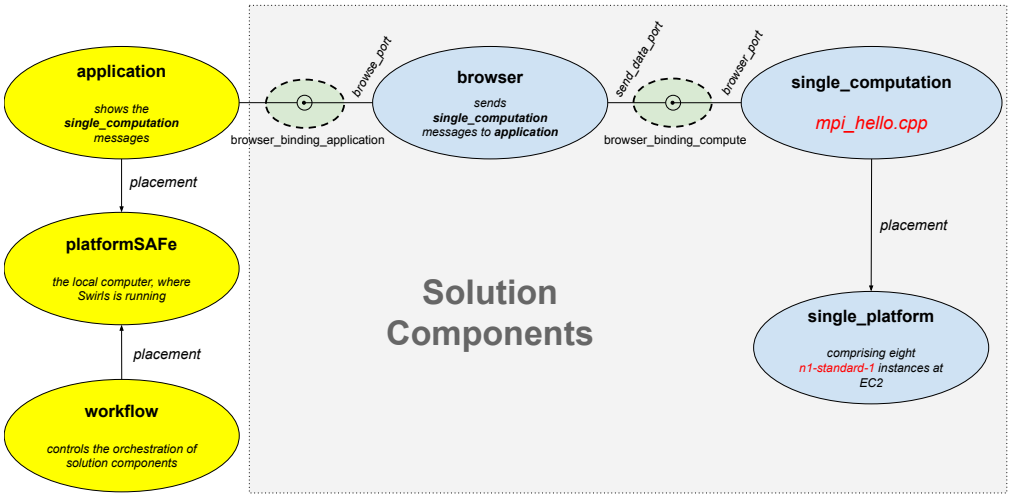

Figura 7: Arquiteura do Sistema "Hello World"

A Figura 6 apresenta uma captura de tela de um notebook Jupyter onde encontram-se comandos Swirls para criação, implantação e execução de um sistema de computação paralela capaz de executar um programa MPI simples escrito em C++ (`mpi_hello.cpp`) sobre uma plataforma virtual de 8 nós representados por instâncias de máquinas do tipo `n1-standard-1` do provedor GCP. A Figura 7 ilustra a arquitetura desse sistema. O nome do componente de computação que encapsula o código MPI é `single_computation`, enquanto a plataforma virtual sobre a qual encontra-se alocado é chamada `single_platform`. O connector `browser` é responsável por permitir que o programa MPI envie mensagens para o componente *application* do sistema, as quais são mostradas como saída do comando `browse` no notebook Jupyter. Para isso, são necessários duas ligações de serviço: `browser_binding_application`, que liga a porta `browse_port` do `browser` ao componente *application* (por *default*, o nome da porta é `application`), e `browser_binding_computation`, que liga a porta `send_data_port` do `browser` à porta `browser_port` do componente `single_computation`.

No exemplo, note o uso do comando `create contract` para especificar um contrato que mais a frente é referenciado no comando `new platform`, que cria a plataforma virtual **single_platform**. Observe o uso do parâmetro de contexto ***maintainer*** para selecionar o backend que será usado para criação da plataforma virtual (i.e., GCP), bem como o uso do parâmetro ***node-locale*** para especificar a região do provedor IaaS aonde será criada (Iowa). Finalmente, o parâmetro ***node-count*** especifica o número de nós de processamentos desejados (i.e., 8). De fato, existem uma grande quantidade de parâmetros de contexto suportados em contratos de plataformas virtuais para selecionar características desejadas da plataforma virtual alvo. Finalmente, note a especificação do contrato do componente **single_computation** no comando `new computation`. Esse é o contrato mais simples possível, apenas espeficando o tipo do componente usando o parâmetro ***name***. Entretanto, nos contratos de componentes de computação é possível usar todos os parâmetros de contexto também suportados por plataformas virtuais a fim de caracterizar sua plataforma virtual alvo.

Referências

[1] Abu-Khzam, F., Bazgan, C., Chopin, M., and Fernau, H. Approximation algorithms inspired by kernelization methods. In *Proc. of the 25th International Symposium on Algorithms and Computation (ISAAC)* (2014), vol. 8889 of *LNCS*, pp. 479–490.

[2] Adasme, P., Andrade, R., Leung, J., and Lisser, A. Improved solution strategies for dominating trees. *Expert Systems with Applications 100* (2018), 30 – 40.

[3] Agerwala, T. Exascale Computing: The Challenges and Opportunities in the Next Decade. In *The Sixteenth International Symposium on High-Performance Computer Architecture (HPCA-16)* (Jan. 2010), pp. 1–1.

[4] Ahadi, A., and Dehghan, A. The complexity of the proper orientation number. *Inf. Process. Lett. 113*, 19-21 (Sept. 2013), 799–803.

[5] Ahadi, A., Dehghan, A., and Saghafian, M. Is there any polynomial upper bound for the universal labeling of graphs? *Journal of Combinatorial Optimization 34*, 3 (Oct 2017), 760–770.

[6] Ahmed, R., Bodwin, G., Sahneh, F. D., Hamm, K., Latifi Jebelli, M. J., Kobourov, S., and Spence, R. Graph spanners: A tutorial review. *Computer Science Review 37* (2020), 100253.

[7] Ai, J., Gerke, S., Gutin, G., Shi, Y., and Taoqiu, Z. Proper orientation number of triangle-free bridgeless outerplanar graphs. *Journal of Graph Theory 95*, 2 (2020), 256–266.

[8] Aigner, M., and Fromme, M. A game of cops and robbers. *Discrete Applied Mathematics 8* (1984), 1–12.

[9] Alon, N., and Mehrabian, A. On a generalization of Meyniel's conjecture on the cops and robbers game. *The Electronic Jounal of Combinatorics 18*, 1 (2011).

[10] Andres, D., and Lock, E. Characterising and recognising game-perfect graphs. *Discrete Mathematics & Theoretical Computer Science vol. 21 no. 1, ICGT 2018* (2019).

[11] Appel, K., and Haken, W. Every planar map is four colorable. *Illinois Journal of Mathematics 21*, 3 (Sept. 1977), 429–567.

[12] Araujo, C., Araujo, J., Silva, A., and Cezar, A. Backbone coloring of graphs with galaxy backbones. *Electronic Notes in Theoretical Computer Science 346* (2019), 53–64. The proceedings of Lagos 2019, the tenth Latin and American Algorithms, Graphs and Optimization Symposium (LAGOS 2019).

[13] Araujo, J., Benevides, F., Cezar, A., and Silva, A. Circular backbone colorings: On matching and tree backbones of planar graphs. *Discrete Applied Mathematics 251* (2018), 69–82.

[14] Araújo, J., Bougeret, M., Campos, V., and Sau, I. Kernelization of maximum minimal vertex cover. *arXiv preprint arXiv:2102.02484* (2021).

[15] Araújo, J., Bougeret, M., Campos, V., and Sau, I. Parameterized complexity of computing maximum minimal blocking and hitting sets. *arXiv preprint arXiv:2102.03404* (2021).

[16] Araujo, J., Cezar, A., Lima, C., dos Santos, V., and Silva, A. On the proper orientation number of chordal graphs. *Theoretical Computer Science* (2021).

[17] Araujo, J., Cohen, N., de Rezende, S. F., Havet, F., and Moura, P. F. On the proper orientation number of bipartite graphs. *Theoretical Computer Science 566* (2015), 59 – 75.

[18] Araujo, J., Ducoffe, G., Nisse, N., and Suchan, K. On interval number in cycle convexity. *Discrete Mathematics & Theoretical Computer Science 20*, 1 (2018).

[19] Araujo, J., Havet, F., Sales, C. L., and Silva, A. Proper orientation of cacti. *Theoretical Computer Science 639* (2016), 14 – 25.

[20] Araujo, J., Havet, F., and Schmitt, M. Steinberg-like theorems for backbone colouring. *Discrete Applied Mathematics 245* (2018), 155 – 167. LAGOS'15 — Eighth Latin-American Algorithms, Graphs, and Optimization Symposium, Fortaleza, Brazil — 2015.

[21] Araújo, J., and Arraes, P. Hull and geodetic numbers for some classes of oriented graphs. *Discrete Applied Mathematics* (2021).

[22] Araújo, J., Campos, V., Girão, D., Nogueira, J., Salgueiro, A., and Silva, A. Cycle convexity and the tunnel number of links, 2020.

[23] Araújo, J., Sales, C. L., Sau, I., and Silva, A. Weighted proper orientations of trees and graphs of bounded treewidth. *Theoretical Computer Science 771* (2019), 39–48.

[24] Araújo, R., Sampaio, R., and Szwarcfiter, J. The convexity of induced paths of order three. *Electronic Notes in Discrete Mathematics 44* (2013), 109–114.

[25] Armstrong, R., Kumfert, G., McInnes, L. C., Parker, S., Allan, B., Sottile, M., Epperly, T., and Tamara, D. The CCA Component Model For High-Performance Scientific Computing. *Concurrency and Computation: Practice and Experience 18*, 2 (2006), 215–229.

[26] Balister, P., Bollobás, B., Narayanan, B., and Shaw, A. Catching a fast robber on the grid. *Journal of Combinatorial Theory, Series A 152* (2017), 341 – 352.

[27] Bang-Jensen, J., and Bessy, S. (Arc-)disjoint flows in networks. *Theoretical Computer Science 526* (2014), 28–40.

[28] Bang-Jensen, J., and Gutin, G. *Digraphs: theory, algorithms and applications*. Springer Verlag, 2010.

[29] Bang-Jensen, J., Havet, F., and Yeo, A. The complexity of finding arc-disjoint branching flows. *Discrete Applied Mathematics 209* (2016), 16–26.

[30] Baude, F., Caromel, D., Dalmasso, C., Danelutto, M., Getov, W., Henrio, L., and Prez, C. GCM: A Grid Extension to Fractal for Autonomous Distributed Components. *Annals of Telecommunications 64*, 1 (2009), 5–24.

[31] Bazgan, C., Brankovic, L., Casel, K., Fernau, H., Jansen, K., Klein, K., Lampis, M., Liedloff, M., Monnot, J., and Paschos, V. The many facets of upper domination. *Theoretical Computer Science 717* (2018), 2–25.

[32] Berman, K. A. Vulnerability of scheduled networks and a generalization of menger's theorem. *Networks 28* (1996), 125–134.

[33] Bessy, S., Hörsch, F., Maia, A. K., Rautenbach, D., and Sau, I. Fpt algorithms for packing k-safe spanning rooted sub (di) graphs. *arXiv preprint arXiv:2105.01582* (2021).

[34] Bhadra, S., and Ferreira, A. Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. In *ADHOC-NOW* (2003), pp. 259–270.

[35] Binkele-Raible, D., Fernau, H., Fomin, F., Lokshtanov, D., Saurabh, S., and Villanger, Y. Kernel(s) for problems with no kernel: On out-trees with many leaves. *ACM Transactions on Algorithms 8*, 4 (2012), 38:1–38:19.

[36] Biswas, A., Raman, V., and Saurabh, S. Approximation in (poly-) logarithmic space. In *Proc. of the 45th International Symposium on Mathematical Foundations of Computer Science (MFCS)* (2020), vol. 170 of *LIPIcs*, pp. 16:1–16:15.

[37] Blackmon, M. B., Boville, B., Bryan, F., Dickinson, R., Gent, P., Kiehl, J., Moritz, R., Randall, D., Shukla, J., Solomon, S., Bonan, G., Doney, S., Fung, I., Hack, J., Hunke, E., and Hurrell, J. e. a. The Community Climate System Model. *BAMS 82*, 11 (2001), 2357–2376.

[38] Blair, G., Coupaye, T., and Stefani, J.-B. Component-Based Architecture: The Fractal Initiative. *Annals of Telecommunications 64* (2009), 1–4.

[39] Bodlaender, H., Downey, R., Fellows, M., and Hermelin, D. On problems without polynomial kernels. *Journal of Computer and System Sciences 75*, 8 (2009), 423–434.

[40] Bodlaender, H., Jansen, B., and Kratsch, S. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics 28*, 1 (2014), 277–305.

[41] Bodlaender, H., Thomassé, S., and Yeo, A. Kernel bounds for disjoint cycles and disjoint paths. *Theoretical Computer Science 412*, 35 (2011), 4570–4578.

[42] Bodlaender, H. L. On the complexity of some coloring games. *International Journal of Foundations of Computer Science 2*, 2 (1991), 133–147. WG-1990, LNCS **484** (1990), pp 30-40.

[43] Bohman, T., Frieze, A., and Sudakov, B. The game chromatic number of random graphs. *Random Structures & Algorithms 32*, 2 (2008), 223–235.

[44] Bonato, A., Chiniforooshan, E., and Pralat, P. Cops and robbers from a distance. *Theoretical Computer Science 411*, 43 (2010), 3834–3844.

[45] Bonato, A., and Nowakovski, R. *The game of Cops and Robber on Graphs*. American Mathematical Society, 2011.

[46] Bondy, J. A., and Murty, U. S. R. *Graph theory*, vol. 244 of *Graduate Texts in Mathematics*. Springer, New York, 2008.

[47] Boria, N., Croce, F., and Paschos, V. On the max min vertex cover problem. *Discrete Applied Mathematics 196* (2015), 62–71.

[48] Braga, H. V. V. *Algoritmos exatos para problemas de spanner em grafos*. PhD thesis, Universidade de São Paulo, SP, Brasil, 2019.

[49] Brandstadt, A., Chepoi, V., and Dragan, F. Distance approximating trees for chordal and dually chordal graphs. *J. Algorithms 30*, 1 (1999), 166 – 184.

[50] Brandstadt, A., Dragan, F. F., Le, H.-O., and Le, V. B. Tree spanners on chordal graphs: complexity and algorithms. *Theor. Comput. Sci. 310*, 1 (2004), 329 – 354.

[51] Brandstadt, A., Dragan, F. F., Le, H.-O., Le, V. B., and Uehara, R. Tree spanners for bipartite graphs and probe interval graphs. *Algorithmica 47*, 1 (2007), 27 – 51.

[52] Broesma, H., Fomin, F., Golovach, P. A., and Woeginger, G. Backbone colorings for graphs : tree and path backbone. *Journal of graph theory 55* (2007).

[53] Bu, Y., Liu, D. D.-F., and Zhu, X. Backbone coloring for graphs with large girths. *Discrete Mathematics 313*, 18 (2013), 1799–1804.

[54] Bui-Xuan, B., Ferreira, A., and Jarry, A. Computing shortest, fastest, and foremost journeys in dynamic networks. *Int. J. Found. Comput. Sci. 14*, 2 (2003), 267–285.

[55] Buyya, R., Srirama, S. N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., Gelenbe, E., Javadi, B., Vaquero, L. M., Netto, M. A. S., Toosi, A. N., Rodriguez, M. A., Llorente, I. M., Vimercati, S. D. C. D., Samarati, P., Milojicic, D., Varela, C., Bahsoon, R., Assuncao, M. D. D., Rana, O., Zhou, W., Jin, H., Gentzsch, W., Zomaya, A. Y., and Shen, H. A manifesto for future generation cloud computing: Research directions for the next decade. *ACM Computing Surveys 51*, 5 (Nov. 2018), 105:1–105:38.

[56] Cai, L. *Tree spanners: spanning trees that approximate distances*. PhD thesis, University of Toronto, Toronto, 1992.

[57] Cai, L. Np-completeness of minimun spanner problems. *Discrete Applied Mathematics 48*, 2 (1994), 187 – 194.

[58] Cai, L., and Coneil, D. G. Tree spanners. *Journal on Discrete Mathematics 8*, 3 (1995), 359 – 387.

[59] Campos, V., Havet, F., Sampaio, R., and Silva, A. Backbone colouring: Tree backbones with small diameter in planar graphs. *Theoretical Computer Science 487*, 0 (2013), 50 – 64.

[60] Campos, V., Lopes, R., Marino, A., and Silva, A. Edge-disjoint branchings in temporal graphs. *The Eletronic Journal of Combinatorics* (2020). To appear.

[61] Campos, V., Lopes, R., Marino, A., and Silva, A. Edge-disjoint branchings in temporal graphs. In *31st International Workshop on Combinatorial Algorithms - IWOCA 2020* (2020).

[62] Carvalho, C., Costa, J., Sales, C. L., Lopes, R., Maia de Oliveira, A. K., and Nisse, N. On the characterization of networks with multiple arc-disjoint branching flows. Research report, UFC ; INRIA ; CNRS ; Université Côte d'Azur ; I3S ; LIRMM ; Université de Montpellier, Nov. 2020.

[63] Casteigts, A., Flocchini, P., Quattrociocchi, W., and Santoro, N. Time-varying graphs and dynamic networks. *IJPEDS 27*, 5 (2012), 387–408.

[64] Cerulli, R., Fink, A., Gentili, M., and Raiconi, A. The k-labeled spanning forest problem. *Procedia - Social and Behavioral Sciences 108* (2014), 153 – 163.

[65] Chalopin, J., Chepoi, V., Nisse, N., and Vaxès, Y. Cop and robber games when the robber can hide and ride. *SIAM Journal on Discrete Math 25*, 1 (2011), 333–359.

[66] Chang, R.-S., and Shing-Jiuan, L. The minimum labeling spanning trees. *Information Processing Letters 63*, 5 (1997), 277 – 282.

[67] Changat, M., and Mathew, J. On triangle path convexity in graphs. *Discrete Mathematics 206*, 1-3 (1999), 91 – 95.

[68] Charpentier, C., Hocquard, H., Éric Sopena, and Zhu, X. A connected version of the graph coloring game. *Discrete Applied Mathematics 283* (2020), 744–750.

[69] Chartrand, G., Fink, J. F., and Zhang, P. Convexity in oriented graphs. *Discrete Applied Mathematics 116*, 1 (2002), 115 – 126.

[70] Chartrand, G., Fink, J. F., and Zhang, P. The hull number of an oriented graph. *International Journal of Mathematics and Mathematical Sciences 2003*, 36 (2003), 2265–2275.

[71] Chartrand, G., Harary, F., and Zhang, P. On the geodetic number of a graph. *Networks 39*, 1 (2002), 1–6.

[72] Chartrand, G., Wall, C. E., and Zhang, P. The convexity number of a graph. *Graphs and Combinatorics 18* (2002), 209–217.

[73] Cheatham, T., Fahmy, A., Stefanescu, D., and Valiant, L. Bulk synchronous parallel computing-a paradigm for transportable software. In *Proceedings of the Twenty-Eighth Annual Hawaii International Conference on System Sciences* (1995), vol. 2, pp. 268–275 vol.2.

[74] Chen, J., Fernau, H., Kanj, I., and Xia, G. Parametric duality and kernelization: Lower bounds and upper bounds on kernel size. *SIAM Journal on Computing 37*, 4 (2007), 1077–1106.

[75] Chen, N. On the Approximability of Influence in Social Networks. *SIAM Journal on Discrete Mathematics 23*, 3 (2009), 1400–1415.

[76] Cohen, N., Martins, N. A., Mc Inerney, F., Nisse, N., Pérennes, S., and Sampaio, R. M. Spy-game on graphs: Complexity and simple topologies. *Theoretical Computer Science 725* (2018), 1 – 15.

[77] Cohen-Addad, V., Hebdige, M., Král', D., Li, Z., and Salgado, E. Steinberg's conjecture is false. *Journal of Combinatorial Theory, Series B 122* (2017), 452 – 456.

[78] Consoli, S., and Moreno Pérez, J. A. Variable neighbourhood search for the k-labelled spanning forest problem. *Electronic Notes in Discrete Mathematics 47* (2015), 29 – 36. The 3rd International Conference on Variable Neighborhood Search (VNS'14).

[79] Consoli, S., Moreno Pérez, J. A., and Mladenović, N. Comparison of metaheuristics for the k-labeled spanning forest problem. *International Transactions in Operational Research 24*, 3 (2017), 559–582.

[80] Consoli, S., Pérez, J. M., and Mladenović, N. Intelligent variable neighbourhood search for the minimum labelling spanning tree problem. *Electronic Notes in Discrete Mathematics 41* (2013), 399 – 406.

[81] Costa, E., Martins, N. A., and Sampaio, R. M. Spy game: Fpt-algorithm and results on graph products. In *COCOON 2021: The 27th International Computing and Combinatorics Conference, Tainan, Taiwan, October 24-26, 2021, Proceedings* (2021), Lecture Notes in Computer Science, Springer.

[82] Costa, E., Pessoa, V. L., Sampaio, R. M., and Soares, R. PSPACE-completeness of two graph coloring games. *Theoretical Computer Science 824-825* (2020), 36–45.

[83] Cugola, G., and Margara, A. Processing Flows of Information: From Data Stream to Complex Event Processing. *ACM Computing Surveys 44*, 3 (June 2012), 15:1–15:62.

[84] Cygan, M., Fomin, F., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. *Parameterized Algorithms*. Springer, 2015.

[85] Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. *Parameterized Algorithms*, 1st ed. Springer Publishing Company, Incorporated, 2015.

[86] Damaschke, P. Parameterized algorithms for double hypergraph dualization with rank limitation and maximum minimal vertex cover. *Discrete Optimization 8*, 1 (2011), 18–24.

[87] de Carvalho Junior, F. H., Al Alam, W. G., and Dantas, A. B. d. O. Contextual Contracts for Component-Oriented Resource Abstraction in a Cloud of High Performance Computing Services. *Concurrency and Computation: Practice and Experience 33*, 18 (2021), e6225.

[88] de Carvalho Junior, F. H., and Lins, R. D. Separation of Concerns for Improving Practice of Parallel Programming. *INFORMATION, An International Journal 8*, 5 (Sept. 2005), 621–638.

[89] de Carvalho Junior, F. H., Lins, R. D., Correa, R. C., and Araújo, G. A. Towards an Architecture for Component-Oriented Parallel Programming. *Concurrency and Computation: Practice and Experience 19*, 5 (2007), 697–719.

[90] de Carvalho Junior, F. H., and Rezende, C. A. A Case Study on Expressiveness and Performance of Component-Oriented Parallel Programming. *Journal of Parallel and Distributed Computing 73*, 5 (2013), 557–569.

[91] de Carvalho Junior, F. H., Silva, J. C., and Dantas, A. B. O. A Scientific Workflow Management System for Orchestration of Parallel Components in a Cloud of Large-Scale Parallel Processing Services. *Science of Computer Programming 173* (Mar. 2019), 95–127.

[92] Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering, 2017.

[93] Dehghan, A. On the in–out–proper orientations of graphs. *Discrete Applied Mathematics 302* (2021), 129–138.

[94] Dehghan, A., and Havet, F. On the semi-proper orientations of graphs. *Discrete Applied Mathematics 296* (2021), 9–25. 16th Cologne–Twente Workshop on Graphs and Combinatorial Optimization (CTW 2018).

[95] Dell, H., and Marx, D. Kernelization of packing problems. In *Proc. of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2012), pp. 68–81.

[96] Dell, H., and Melkebeek, D. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *Journal of the ACM 61*, 4 (2014), 23:1–23:27.

[97] Dourado, M. C., Protti, F., and Szwarcfiter, J. L. Complexity results related to monophonic convexity. *Discrete Applied Mathematics 158*, 12 (2010), 1268 – 1274. Traces from LAGOS'07 IV Latin American Algorithms, Graphs, and Optimization Symposium Puerto Varas - 2007.

[98] Downey, R., and Fellows, M. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[99] Dragan, F. F., Fomin, F. V., and Golovach, P. A. Spanners in sparse graphs. *J. Comput. Syst. Sci. 77*, 6 (2011), 1108 – 1119.

[100] Dublois, L., Hanaka, T., Ghadikolaei, M., Lampis, M., and Melissinos, N. (In)approximability of Maximum Minimal FVS. In *Proc. of the 31st International Symposium on Algorithms and Computation (ISAAC)* (2020), vol. 181 of *LIPIcs*, pp. 3:1–3:14. The cubic kernel appears in the full version, available at `https://arxiv.org/abs/2009.09971`.

[101] Dublois, L., Lampis, M., and Paschos, V. Upper dominating set: Tight algorithms for pathwidth and sub-exponential approximation. In *Algorithms and Complexity: 12th International Conference, CIAC 2021, Virtual Event, May 10–12, 2021, Proceedings 12* (2021), Springer, pp. 202–215.

[102] Edmonds, J. Edge-disjoint branchings. *Combinatorial Algorithms* (1973).

[103] Emek, Y., and Peleg, D. Approximating minimum max-stretch spanning trees on unweighted graphs. *SIAM Journal on Computing 38*, 5 (2008), 1761 – 1781.

[104] Erdös, P., Fried, E., Hajnal, A., and Milner, E. C. Some remarks on simple tournaments. *algebra universalis 2*, 1 (Dec 1972), 238–245.

[105] Faigle, U., Kern, U., Kierstead, H., and Trotter, W. On the game chromatic number of some classes of graphs. *Ars Combinatoria 35* (1993), 143–150.

[106] Fan, Z., Qiu, F., Kaufman, A., and Yoakum Stover, S. GPU Cluster for High Performance Computing. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing (SC'04)* (2004), IEEE Computer Society, pp. 47–47.

[107] Faragó, A., and Syrotiuk, V. R. Merit: A unified framework for routing protocol assessment in mobile ad hoc networks. In *MOBICOM 2001, Proceedings of the seventh annual international conference on Mobile computing and networking, Rome, Italy, July 16-21, 2001* (2001), pp. 53–60.

[108] Farber, M., and Jamison, R. E. Convexity in graphs and hypergraphs. *SIAM J. Algebraic Discrete Methods 7* (July 1986), 433–444.

[109] Farrugia, A. Orientable convexity, geodetic and hull numbers in graphs. *Discrete Appl. Math. 148* (June 2005), 256–262.

[110] Fekete, S. P., and Kremer, J. Tree spanners in planar graphs. *Discrete Applied Mathematics 108*, 1 (2001), 85 – 103.

[111] Fernau, H. *Parameterized algorithms: a graph-theoretic approach.* Habilitationsschrift, Universität Tübingen, 2005.

[112] Figueredo, P. J. O problema da floresta geradora k-rotulada. Master's thesis, Universidade Federal do Ceará, Fortaleza, CE, Brasil, 2020.

[113] Fomin, F., Golovach, P. A., and Pralat, P. Cops and robber with constraints. *SIAM Journal on Discrete Mathematics 26*, 2 (2012), 571–590.

[114] Fomin, F., Lokshtanov, D., Saurabh, S., and Zehavi, M. *Kernelization. Theory of Parameterized Preprocessing.* Cambridge University Press, 2019.

[115] Fomin, F. V., Golovach, P. A., Kratochvíl, J., Nisse, N., and Suchan, K. Pursuing a fast robber on a graph. *Theoretical Computer Science 411*, 7-9 (2010), 1167–1181.

[116] Fomin, F. V., Golovach, P. A., and Lokshtanov, D. Cops and robber game without recharging. In *12th Scandinavian Symp. and Workshops on Algorithm Theory (SWAT)* (2010), vol. 6139 of *LNCS*, Springer, pp. 273–284.

[117] Foster, I., and Kesselman, C. *The Grid 2: Blueprint for a New Computing Infrastructure.* M. Kauffman, 2004.

[118] Galbiati, G. On min-max cycle bases. *Algorithms and Computation, volume 2223 of Lecture Notes in Computer Science* (2001), 116 – 123.

[119] Gardner, M. Mathematical games. *Scientific American 244*, 4 (1981), 18–26.

[120] Giannopoulou, A., Lokshtanov, D., Saurabh, S., and Suchý, O. Tree Deletion Set has a polynomial kernel but no $\mathrm{OPT}^{O(1)}$ approximation. *SIAM Journal on Discrete Mathematics 30*, 3 (2016), 1371–1384.

[121] Goddard, W., Hedetniemi, S., and Hedetniemi, S. Eternal security in graphs. *Journal of Combinatorial Mathematics and Combinatorial Computing 52* (2005).

[122] Goldwasser, J. L., and Klostermeyer, W. Tight bounds for eternal dominating sets in graphs. *Discrete Mathematics 308*, 12 (2008), 2589–2593.

[123] Gonzalez, J. E., Low, Y., Gu, H., Bickson, D., and Guestrin, C. Powergraph: Distributed graph-parallel computation on natural graphs. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)* (Hollywood, CA, Oct. 2012), USENIX Association, pp. 17–30.

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

[125] Gu, R., Lei, H., Ma, Y., and Taoqiu, Z. Note on (semi-)proper orientation of some triangulated planar graphs. *Applied Mathematics and Computation 392* (2021), 125723.

[126] Guo, J., Kanj, I., and Kratsch, S. Safe approximation and its relation to kernelization. In *Proc. of the 6th International Symposium on Parameterized and Exact Computation (IPEC)* (2011), vol. 7112 of *LNCS*, pp. 169–180.

[127] Havet, F., King, A. D., Liedloff, M., and Todinca, I. (Circular) backbone colouring: Forest backbones in planar graphs. *Discrete Applied Mathematics 169* (2014), 119–134.

[128] Havet, F., and Zhu, X. The game grundy number of graphs. *Journal of Combinatorial Optimization 25*, 4 (2013), 752–765.

[129] Herbordt, M. C., VanCourt, T., Gu, Y., Sukhwani, B., Conti, A., Model, J., and DiSabello, D. Achieving High Performance with FPGA-Based Computing. *Computer 40* (March 2007), 50–57.

[130] Hermelin, D., and Wu, X. Weak compositions and their applications to polynomial lower bounds for kernelization. In *Proc. of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2012), pp. 104–113.

[131] Holme, P. Modern temporal network theory: a colloquium. *The European Physical Journal B 88*, 39 (2015), 234.

[132] Huang, S., Fu, A. W.-C., and Liu, R. Minimum spanning trees in temporal graphs. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2015), SIGMOD '15, ACM, pp. 419–430.

[133] Huerta, E. A., Khan, A., D., and et al., E. Convergence of artificial intelligence and high performance computing on NSF-supported cyberinfrastructure. *Journal of Big Data 7*, 88 (2020).

[134] Ibiapina, A., and Silva, A. Mengerian temporal graphs revisited. In *23rd International Symposium on Fundamentals of Computation Theory - FCT 2021* (2021).

[135] J., A., A., C. A., and A., S. On the existence of tree backbones that realize the chromatic number on a backbone coloring. *Journal of Graph Theory 85*, 4 (2016), 808–813.

[136] Jensen, T. R., and Toft, B. *Graph Coloring Problems*. Wiley-Interscience, New York, 1995.

[137] Jouppi, N., Young, C., Patil, N., and Patterson, D. Motivation for and Evaluation of the First Tensor Processing Unit. *IEEE Micro 38*, 3 (2018), 10–19.

[138] Karp, R. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, R. Miller and J. Thatcher, Eds. Plenum Press, 1972, pp. 85–103.

[139] Keiler, L., Lima, C. V. G. C., Maia, A. K., Sampaio, R., and Sau, I. Target set selection with maximum activation time. In *XI Latin and American Algorithms, Graphs and Optimization Symposium (LAGOS 2021)* (2021), Procedia Computer Science, Elsevier.

[140] Kempe, D., Kleinberg, J., and Kumar, A. Connectivity and inference problems for temporal networks. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing* (2000).

[141] Kierstead, H. A., and Trotter, W. T. Planar graph coloring with an uncooperative partner. *Journal of Graph Theory 18*, 6 (1994), 569–584.

[142] Kinnersley, W. B. Cops and robbers is exptime-complete. *Journal of Combinatorial Theory, Series B 111* (2015), 201–220.

[143] Kipf, T. N., and Welling, M. Semi-supervised classification with graph convolutional networks, 2017.

[144] Klostermeyer, W., and MacGillivray, G. Eternal dominating sets in graphs. *Journal of Combinatorial Mathematics and Combinatorial Computing 68* (2009).

[145] Klostermeyer, W., and Mynhardt, C. Graphs with equal eternal vertex cover and eternal domination numbers. *Discrete Mathematics 311*, 14 (2011), 1371–1379.

[146] Knox, F., Matsumoto, N., de la Maza, S. G. H., Mohar, B., and Sales, C. L. Proper orientations of planar bipartite graphs. *Graphs and Combinatorics 33*, 5 (Sep 2017), 1189–1194.

[147] Kratsch, S. Polynomial kernelizations for MIN $F^+\Pi_1$ and MAX NP. *Algorithmica 63*, 1-2 (2012), 532–550.

[148] Latapy, M., Viard, T., and Magnien, C. Stream graphs and link streams for the modeling of interactions over time. *Social Network Analysis and Mining 8* (2018), 61.

[149] Lecun, Y., and Bengio, Y. *Convolutional Networks for Images, Speech and Time Series*. The MIT Press, 1995, pp. 255–258.

[150] Li, R., Hu, H., Li, H., Wu, Y., and Yang, J. MapReduce Parallel Programming Model: A State-of-the-Art Survey. *International Journal of Parallel Programming 44*, 4 (2016), 832–866.

[151] Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. Gated graph sequence neural networks, 2017.

[152] Lin, J. Mapreduce is Good Enough?If All You Have is a Hammer, Throw Away Everything That's Not a Nail! *BigData 1*, 1 (Feb. 2013), 28–37.

[153] Lokshtanov, D., Panolan, F., Ramanujan, M., and Saurabh, S. Lossy kernelization. In *Proc. of the 49th Annual ACM Symposium on Theory of Computing (STOC)* (2017), pp. 224–237.

[154] Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M. Distributed graphlab: A framework for machine learning in the cloud, 2012.

[155] Lusk, E., and Yelick, K. Languages for High-Productivity Computing - The DARPA HPCS Language Support. *Parallel Processing Letters*, 1 (2007), 89–102.

[156] Ma, L., Yang, Z., Miao, Y., Xue, J., Wu, M., Zhou, L., and Dai, Y. Towards efficient large-scale graph neural network computing, 2018.

[157] Madanlal, M., Venkatesan, G., and Rangan, C. Tree 3-spanners on interval, permutation and regular bipartite graphs. *Information Processing Letters 59*, 2 (1996), 97 – 102.

[158] Mafort, R., and Protti, F. Vector domination in split-indifferent graphs. *Information Processing Letters 155* (2020).

[159] Mahjoub, A., and Mailfert, J. On the independent dominating set polytope. *European Journal of Combinatorics 27*, 4 (2006), 601–616.

[160] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2010), SIGMOD '10, Association for Computing Machinery, p. 135–146.

[161] Mamino, M. On the computational complexity of a game of cops and robbers. *Theoretical Computer Science 477* (2013), 48–56.

[162] Marcilon, T., Martins, N. A., and Sampaio, R. M. Hardness of variants of the graph coloring game. In *LATIN 2020: Theoretical Informatics - 14th Latin American Symposium, São Paulo, Brazil, January 5-8, 2021, Proceedings* (2020), Y. Kohayakawa and F. K. Miyazawa, Eds., vol. 12118 of *Lecture Notes in Computer Science*, Springer, pp. 348–359.

[163] Marino, A., and Silva, A. Coloring temporal graphs. *Journal of Computer and System Sciences* (2021). To appear.

[164] Marino, A., and Silva, A. Königsberg sightseeing: Eulerian walks in temporal graphs. In *32nd International Workshop on Combinatorial Algorithms - IWOCA 2021* (2021).

[165] Mertzios, G., Michail, O., and Spirakis, P. Temporal network optimization subject to connectivity constraints. *Algorithmica 81* (2019), 1416—-1449.

[166] Michail, O. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics 12*, 4 (2016), 239–280.

[167] Mittal, S. A Survey of Techniques for Architecting and Managing Asymmetric Multicore Processors. *ACM Compututing Surveys 48*, 3 (Feb. 2016), 45:1–45:38.

[168] Molloy, M. S., and Reed, B. A. *Graph colouring and the probabilistic method*, vol. 23. Springer Verlag, 2002.

[169] Narayanaswamy, N., and Ramakrishna, G. Tree t-spanners in outerplanar graphs via supply demand partition. *Discrete Applied Mathematics 195* (2015), 104 – 109.

[170] Nicosia, V., Tang, J., Musolesi, M., Russo, G., Mascolo, C., and Latora, V. Components in time-varying graphs. *Chaos 22*, 2 (2012).

[171] Noguchi, K. Proper 3-orientations of bipartite planar graphs with minimum degree at least 3. *Discrete Applied Mathematics 279* (2020), 195–197.

[172] Nowakowski, R. J., and Winkler, P. Vertex-to-vertex pursuit in a graph. *Discrete Mathematics 43* (1983), 235–239.

[173] Oliveira, C. A. S., and Pardalos, P. M. A survey of combinatorial optimization problems in multicast routing. *Comput. Oper. Res. 32*, 8 (2005), 1953 – 1981.

[174] Ore, O. *The four color problem.* Academic Press, New York, 1967.

[175] Papoutsakis, I. Tree 3-spanners of diameter at most 5. *CoRR, abs/1402.3573* (2014).

[176] Papoutsakis, I. Tree spanners of bounded degree graphs. *Discrete Appl. Math. 236* (2018), 395 – 407.

[177] Pelayo, I. M. *Geodesic Convexity in Graphs.* SpringerBriefs in Mathematics. Springer, 2013.

[178] Pelayo, I. M. *Geodesic Convexity in Graphs.* SpringerBriefs in Mathematics. Springer-Verlag, New York, 2013.

[179] Peleg, D., and Reshef, E. A variant of the arrow distributed directory with low average complexity. *Automata, Languages and Programming* (1999), 615 – 624.

[180] Peleg, D., and Schaffer, A. A. Graph spanners. *Journal of Graph Theory* (1989), 99 – 116.

[181] Peleg, D., and Upfal, E. A tradeoff between space and efficiency for routing tables. *Proceedings of the twentieth annual ACM symposium on Theory of computing* (1988), 43 – 52.

[182] Penso, L. D., Protti, F., Rautenbach, D., and Souza, U. Complexity analysis of p3-convexity problems on bounded-degree and planar graphs. *Theoretical Computer Science 607* (2015), 83–95.

[183] Pfaltz, J. L. Convexity in directed graphs. *Journal of Combinatorial Theory, Series B 10*, 2 (1971), 143 – 162.

[184] Pierce, B. *Types and Programming Languages*. The MIT Press, 2002.

[185] Post, D. E., and Votta, L. G. Computational Science Demands a New Paradigm. *Physics Today 58*, 1 (2005), 35–41.

[186] Post D. E. The Coming Crisis in Computational Science. In *IEEE International Conference on High Performance Computer Architecture (HPCA), Workshop on Productivity and Performance in High-End Computing (P-PHEC)* (2004), Madrid, Spain.

[187] Rezende, C. A., and de Carvalho Junior, F. H. MapReduce with Components for Processing Big Graphs. In *2018 Symposium on High Performance Computing Systems (WSCAD'2018)* (Oct. 2018), pp. 108–115.

[188] Rockafellar, R. T. *Convex analysis*. Princeton Mathematical Series, No. 28. Princeton University Press, Princeton, N.J., 1970.

[189] Röger, H., and Mayer, R. A Comprehensive Survey on Parallelization and Elasticity in Stream Processing. *ACM Computing Surveys 52*, 2 (Apr. 2019), 36:1–36:37.

[190] Santos, A. C., Duhamel, C., Belisário, L. S., and Guedes, L. M. Strategies for designing energy-efficient clusters-based wsn topologies. *Journal of Heuristics 18*, 4 (Aug 2012), 657–675.

[191] Sousa, G. Problema da árvore t-spanner de custo mínimo. Master's thesis, Universidade Federal do Ceará, CE, Brasil, 2021.

[192] Tzeng, G.-H., and Huang, J.-J. *Multiple Attribute Decision Making: Methods and Applications*. Chapman and Hall/CRC, 2011.

[193] van der Steen, A. J. Issues in Computational Frameworks. *Concurrency and Computation: Practice and Experience 18*, 2 (2006), 141–150.

[194] Varlet, J. C. Convexity in tournaments. *Bull. Société Royale des Sciences de Liège 45* (1976), 570 – 586.

[195] Venkatesan, G., Rotics, U., Madanlal, M., Makowsky, J., and Rangan, C. P. Restrictions of minimum spanner problems. *Information and Computation 136*, 2 (1997), 143 – 164.

[196] Vlajic, N., and Xia, D. Wireless sensor networks: to cluster or not to cluster? In *2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'06)* (2006), pp. 9 pp.–268.

[197] VoBs, S., Fink, A., and Duin, C. Looking ahead with the pilot method. *Annals of Operations Research 136*, 1 (Apr 2005), 285–302.

[198] Wasserman, S., and Faust, K. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[199] Wolsey, L. *Integer Programming*. John Wiley & Sons, [*s.l.*], 1998.

[200] Wu, H., Cheng, J., Huang, S., Ke, Y., Lu, Y., and Xu, Y. Path problems in temporal graphs. *PVLDB 7*, 9 (2014), 721–732.

[201] Xiao, W., Xue, J., Miao, Y., Li, Z., Chen, C., Wu, M., Li, W., and Zhou, L. Distributed graph computation meets machine learning. *IEEE Transactions on Parallel and Distributed Systems 31*, 7 (2020), 1588–1604.

[202] Yannakakis, M. The effect of a connectivity requirement on the complexity of maximum subgraph problems. *Journal of the ACM 26*, 4 (1979), 618–630.

[203] Zahran, M. Heterogeneous Computing: Here to Stay. *Communications of the ACM 60*, 3 (Feb. 2017), 42–45.

[204] Zermelo, E. Über eine anwendung der mengenlehre auf die theorie des schachspiels. In *Proc. Fifth International Congr. Math.* (1913), pp. 501–504.

[205] Zhu, X. The game coloring number of planar graphs. *Journal of Combinatorial Theory, Series B 75*, 2 (1999), 245–258.

[206] Zhu, X. Refined activation strategy for the marking game. *Journal of Combinatorial Theory, Series B 98*, 1 (2008), 1–18.