



CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM MATEMÁTICA E COMPUTACIONAL

**EFEITO DA HIBRIDIZAÇÃO
DINÂMICA DE
META-HEURÍSTICAS EM UM
SISTEMA MULTIAGENTES
DISTRIBUÍDO E ESCALÁVEL**

Felipe Duarte dos Reis

Orientador: Henrique Elias Borges

Coorientador: Rogério Martins Gomes

BELO HORIZONTE

MARÇO DE 2022

Felipe Duarte dos Reis

EFEITO DA HIBRIDIZAÇÃO DINÂMICA DE META-HEURÍSTICAS EM UM SISTEMA MULTIAGENTES DISTRIBUÍDO E ESCALÁVEL

Dissertação de mestrado apresentada ao Programa de Pós Graduação em Modelagem Matemática e Computacional do Centro Federal de Educação Tecnológica de Minas Gerais, como requisito parcial para obtenção do título de Mestre em Modelagem Matemática e Computacional apresentada ao Programa de Pós-graduação em Modelagem Matemática e Computacional do Centro Federal de Educação Tecnológica de Minas Gerais, como requisito parcial para a obtenção do título de em Modelagem Matemática e Computacional.

Orientador: Henrique Elias Borges

Coorientador: Rogério Martins Gomes

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM MATEMÁTICA E COMPUTACIONAL
BELO HORIZONTE
MARÇO DE 2022

R375e Reis, Felipe Duarte dos
Efeito da hibridização dinâmica de meta-heurísticas em um sistema multiagentes distribuído e escalável / Felipe Duarte dos Reis. – 2022.
96 f.

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional.

Orientador: Henrique Elias Borges.

Coorientador: Rogério Martins Gomes.

Dissertação (mestrado) – Centro Federal de Educação Tecnológica de Minas Gerais.

1. Sistemas inteligentes de controle – Teses. 2. Agentes inteligentes – Teses. 3. Otimização combinatória – Teses. 4. Sistemas distribuídos – Teses.
I. Borges, Henrique Elias. II. Gomes, Rogério Martins. III. Centro Federal de Educação Tecnológica de Minas Gerais. IV. Título.

CDD 519.6



SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
COORDENAÇÃO DO CURSO DE MESTRADO EM MODELAGEM MATEMÁTICA E COMPUTACIONAL

“EFEITO DA HIBRIDIZAÇÃO DINÂMICA DE META-HEURÍSTICAS EM UM SISTEMA MULTIAGENTES DISTRIBUÍDO E ESCALÁVEL”

Dissertação de Mestrado apresentada por **Felipe Duarte dos Reis**, em 25 de março de 2022, ao Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG, e aprovada pela banca examinadora constituída pelos professores:

Prof. Dr. Henrique Elias Borges (Orientador)
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. Rogério Martins Gomes (Coorientador)
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. Bruno André Santos
Centro Federal de Educação Tecnológica de Minas Gerais

Prof^ª. Dr^ª. Elizabeth Fialho Wanner
Centro Federal de Educação Tecnológica de Minas Gerais

Visto e permitida a impressão,

Prof^ª. Dr^ª. Elizabeth Fialho Wanner
Presidenta do Colegiado do Programa de Pós-Graduação em
Modelagem Matemática e Computacional

Agradecimentos

Não é possível começar este agradecimento sem antes trazer o contexto em que este trabalho se passou. Quando iniciei o curso do PPGMMC em 2019, jamais esperaria que o mundo fosse se deparar com uma pandemia no ano de 2020 que se estende até o presente momento. Presto aqui as minhas homenagens às tantas vítimas que esta doença fez, em especial no Brasil. Agradeço ao Eterno por ninguém próximo a mim ter se deparado com esta doença e padecido para ela, mas também rezo para que os que perderam entes queridos tenham o coração consolado.

Agradeço aos meus pais, Renato e Janaina, que me incentivaram a estudar, e que jamais puseram limites até onde eu poderia ir. Espero sempre surpreendê-los e ir mais longe.

Agradeço a minha noiva, Júlia, por todo afeto e compreensão, por todo apoio e energia. Por me socorrer nas horas mais difíceis. Por ter lido este texto incansáveis vezes. Juntos sonharemos sempre mais alto. Agradeço também aos meus sogros, José Maria e Ângela por todo o apoio ao longo do desenvolvimento deste trabalho.

Agradeço ao professor Henrique, que já se tornou a muito um querido amigo, com quem aprendo sempre. Obrigado pelo suporte ao longo deste trabalho, por todos os comentários cirúrgicos, pela gentil disposição em ensinar. Agradeço também ao professor Rogério, que também a muito tenho o privilégio de chamar de amigo, e que se juntou a este trabalho em um momento crítico em que tudo estava a ir pelos ares. Obrigado pela injeção de ânimo em cada reunião, ainda que demoradas, mas sempre com muitas risadas.

Agradeço ao CEFET-MG, enquanto aluno do PPGMMC pelo suporte, e enquanto funcionário dele pela compreensão. Passei os últimos 13 anos da minha vida acadêmica nesta instituição, como aluno do técnico e da graduação, como funcionário e agora aluno da pós-graduação. Tenho imenso carinho pelo CEFET, que me abriu portas e mudou minha vida através da educação.

Agradeço aos amigos que o CEFET me deu, em especial Carolina Marcelino, Gustavo Borba, Amanda Borba, Bárbara Jaber, Sinval Júnior, Vitor Peixoto, Lara Loures, Lucas Marioza, Paulo Hoffmam. Vocês acompanharam este trabalho de perto de alguma maneira, nos cafés, enquanto presencial, nas videochamadas durante o isolamento social.

Agradeço aos amigos que o Porto me deu, em especial Michelly, que chegou a tempo de testemunhar o desfecho deste trabalho. Obrigado, pelo alívio cômico e pela companhia nos almoços de domingo.

Sumário

1 – Introdução	1
1.1 Motivação	6
1.2 Justificativa	7
1.3 Objetivos	8
1.4 Organização do Trabalho	9
2 – Revisão bibliográfica	10
2.1 Arquiteturas Multi-Agentes para Otimização	10
2.1.1 AgE	11
2.1.2 CMA	11
2.1.3 JABAT	11
2.1.4 MACS	12
2.1.5 MANGO	12
2.1.6 AMAM	13
2.2 Histórico da Arquitetura D-Optimas	13
2.3 Considerações Finais	20
3 – Metodologia	22
3.1 Hipóteses	22
3.2 Método	23
3.3 Premissas	26
3.4 Experimentos	27
3.5 Considerações Finais	29
4 – Desenvolvimento	30
4.1 Da topologia do sistema utilizando <i>akka-cluster</i>	31
4.2 Da dinâmica do espaço de busca e interações entre regiões e o SimulationActor	35
4.3 Da dinâmica interna do agente	39
4.4 Adição de novos algoritmos populacionais	41
4.5 Considerações finais	46
5 – Avaliação da escalabilidade	47
5.1 Avaliação de desempenho da arquitetura	47
5.2 Resultados do experimento	49
5.3 Síntese do resultado	53
6 – Avaliação do efeito da diversidade na qualidade das soluções produzidas	55

6.1	Experimentos preparatórios	57
6.2	Experimentos de <i>baseline</i>	58
6.3	Avaliando o efeito da hibridização de um agente populacional e uma busca local	60
6.4	Avaliando o efeito do aumento do número de regiões na presença de um agente populacional e uma busca local	63
6.5	Efeito do aumento do número dos agentes e variações em suas configurações	65
6.6	Efeito da combinação de diferentes agentes populacionais	70
6.7	Síntese do resultado e considerações finais	73
7	– Conclusão	76
7.1	Principais contribuições	77
7.2	Trabalhos futuros	79
	Referências	81
	 Apêndices	 86
	APÊNDICE A – Avaliação do efeito da diversidade	87
A.1	Número médio de regiões	87

Resumo

Metaheurísticas híbridas reportam uma melhora significativa comparadas com as respectivas implementações clássicas. Apesar da técnica de hibridização de metaheurísticas ser promissora, ainda assim é possível encontrar problemas técnicos e fundamentalmente teóricos. As principais dificuldades técnicas são a necessidade do conhecimento do problema, e uma vez desenvolvida, é necessário calibrar os parâmetros numéricos do algoritmo. Uma alternativa ao desenvolvimento de novas meta-heurísticas, ou a hibridização manual delas, é utilizar os mecanismos de colaboração e comunicação próprios da modelagem de sistemas multi-agentes (MMAS) para promover uma hibridização dinâmica de diferentes estratégias de busca. A arquitetura D-Optimas é um MMAS baseado no modelo de atores, onde cada agente encapsula uma meta-heurística diferente e, dotado de um mecanismo de aprendizagem colabora com os demais agentes para encontrar a melhor solução para um problema de otimização. Os agentes interagem no espaço de busca que é dividido em regiões, que possuem um comportamento independente, podendo receber novas soluções, se particionar ou se fundir. O presente trabalho evoluiu a arquitetura D-Optimas comparada a sua última versão, adicionando mais algoritmos de otimização, atualizando a implementação para a biblioteca *akka-cluster* e simplificando a sua execução em um *cluster*. Este trabalho avaliou experimentalmente tanto a escalabilidade quanto o efeito da diversidade na qualidade das soluções. A arquitetura se mostrou escalável em um cluster de até seis nós, mantendo o número de agentes. A diversidade não se mostrou um fator relevante em todos os casos estudados.

Palavras-chave: Sistemas multi-agente; Otimização; Sistemas distribuídos.

Abstract

Hybrid metaheuristics report a significant improvement compared to their classical implementations. Although the hybridization technique of metaheuristics is promising, it is still possible to encounter technical and fundamentally theoretical problems. The main technical difficulties are the need to know the problem, and once developed, it is necessary to calibrate the numerical parameters of the algorithm. An alternative to the development of new meta-heuristics, or their manual hybridization, is to use the collaboration and communication mechanisms typical of multi-agent systems (MMAS) modeling to promote a dynamic hybridization of different search strategies. The D-Optimas architecture is an MMAS based on the actors model, where each agent encapsulates a different meta-heuristic and, equipped with a learning mechanism, collaborates with the other agents to find the best solution for an optimization problem. The agents interact in the search space, which is divided into regions, which have an independent behavior, being able to receive new solutions, partition or merge. The present work has evolved the D-Optimas architecture compared to its last version, adding more optimization algorithms, updating the implementation to the *akka-cluster* library and simplifying its execution in a *cluster*. This work experimentally evaluated both scalability and the effect of diversity on the quality of solutions. The architecture proved to be scalable in a cluster of up to six nodes, maintaining the number of agents. Diversity was not a relevant factor in all the cases studied.

Keywords: Multi-agent systems; Optimization; Distributed systems.

Capítulo 1

Introdução

Mestre não é quem sempre ensina, mas quem de repente aprende.

Riobaldo, em "Grande Sertão: Veredas"

Ao observar o comportamento de diferentes populações de seres vivos, *e.g.*, insetos, aves e mamíferos, dois fatores do seu modo de vida parecem estar correlacionados: a diversidade dos indivíduos e a maneira como colaboram. A colaboração, sem dúvida, contribui para a sobrevivência da população, principalmente se esta é mais fraca que o seu predador natural ou está em desvantagem em relação ao ambiente. Tomando como exemplo o caso das formigas: não parece razoável imaginar que uma única formiga, ou ainda um grupo muito pequeno, conseguiria cortar folhas, carregá-las para o formigueiro, defender o seu lar, enquanto procriam para perpetuar a espécie. Entretanto, colaborando na casa dos milhares de indivíduos, elas alcançam sucesso em sobreviver e se procriar.

Em sua obra intitulada "*Árvore do Conhecimento*", Maturana e Varela (2001) dão alguns exemplos deste tipo de fenômeno social. Os autores apresentam, como exemplo, a estratégia dos rebanhos de antílopes para se comunicar e fugir de uma ameaça de coordenada, sempre ficando um indivíduo para trás em um local mais alto, de modo a observar a retaguarda do grupo. Na sequência, o autor aborda a interação de uma alcateia de lobos que colabora para caçar um animal.

"Essa forma peculiar de conduta, em que animais distintos cumprem papéis distintos, permite que os membros do rebanho relacionem-se em atividades que não lhes seriam possíveis como indivíduos isolados. Além desse exemplo de fuga, há muitos outros no sentido inverso; Por exemplo, os lobos vivem também em grupos, coordenando suas condutas mediante várias interações olfativas, faciais e corporais, como o mostrar dos dentes, o abaixar das orelhas e o mover da cauda, como vemos nos cães domésticos. Tal grupo, como unidade social,

é capaz de perseguir e matar um gigantesco alce, façanha que não estaria à altura de nenhum lobo solitário.” (MATURANA; VARELA, 2001)

Aqui, Maturana e Varela (2001) ressaltam o papel da diversidade na maneira como os animais colaboram, quando falam em animais que cumprem papéis distintos, com comportamentos e habilidades diferentes. As formigas, por exemplo, tem papéis muito específicos dentro do formigueiro para conseguirem cooperar com sucesso. Algumas trabalham em coletar alimento, outras defendem o formigueiro, e os machos juntamente com a rainha são responsáveis pela reprodução (MATURANA; VARELA, 2001). Cada indivíduo na população tem uma morfologia adaptada e específica para aquela tarefa. Mas ainda assim, não são suficientes sozinhos para desempenha-la com sucesso. Uma formiga com as presas mais afiadas do que as demais poderia ser muito mais rápida para cortar uma folha, mas ainda assim não conseguiria abastecer o formigueiro inteiro sozinha.

Maturana e Varela (2001) sublinham que outro ponto importante para o surgimento de um comportamento colaborativo é a comunicação. Espécies que possuem um repertório comunicacional maior conseguem se organizar de maneira mais complexa. Um exemplo disto é a própria espécie humana, que possui um repertório de comunicação vasto comparado com o dos lobos ou antílopes. O historiador Yuval Noah Harari, no seu popular *"Sapiens: Uma Breve História da Humanidade"* argumenta que a capacidade de falar e argumentar sobre coisas abstratas, de realizar comércio, de compartilhar crenças, levou a espécie *H. Sapiens* a sobressair sobre as demais espécies do gênero *Homo*.

”Em uma briga de um para um, provavelmente um neandertal teria derrotado um sapiens. Mas em um conflito de centenas, os neandertais não teriam uma chance sequer. Os neandertais podiam partilhar informações sobre o paradeiro de leões, mas provavelmente não podiam contar - e revisar - histórias sobre espíritos tribais. Sem a capacidade de criar ficção, os neandertais não conseguiam cooperar efetivamente em grande número nem adaptar o seu ambiente social para responder aos desafios em rápida transformação. [...]

Se Sapiens arcaicos que acreditavam em tais ficções trocaram conchas e obsidiana, é razoável pensar que também podem ter trocado informações, criando assim redes de conhecimento muito mais amplas e mais densas do que a que serviu aos neandertais e a outros humanos arcaicos. [...]

Cinquenta neandertais cooperando em padrões tradicionais e estáticos não eram páreo para cinco centenas de Sapiens versáteis e inovadores.” (HARARI, 2015)

Novamente o conceito de diversidade aparece em Harari (2015). O autor argumenta que o ser humano consegue cooperar de uma maneira dinâmica e diversa e que essa não

era uma capacidade presente em outras espécies do genero *Homo*. Baseado nos excertos apresentados, é razoável dizer que na natureza há uma correlação entre a complexidade da cooperação e a complexidade das interações entre os agentes da população. Dito de outra maneira, quanto maior a diversidade dos agentes, e quanto maior as possibilidades de comunicação entre esses agentes, maior a chance dessa população conseguir se adaptar a novas situações.

Estes fenômenos populacionais apresentados acima são frequentemente usados na ciência da computação como inspiração para o desenvolvimento de novas técnicas e algoritmos. Tomando como exemplo a área específica da otimização, são vários os algoritmos chamados bioinspirados: algoritmos genéticos, colônia de formigas, enxame de partículas, entre outros (BINITHA; SATHYA et al., 2012). Essas técnicas mapeiam características particulares de um fenômeno biológico, *e.g.* os algoritmos genético (GA, da sigla em inglês *genetic algorithm*), são inspirados na deriva genética das espécies. Nesta técnica, uma solução para um problema de otimização é encarada como um indivíduo inserido em uma população na qual ele pode se reproduzir e gerar descendentes que sofrem mutações. Esta população sofre uma pressão seletiva, de modo que os indivíduos mais adaptados tem maiores chances de sobreviver e gerar outros descendentes. No contexto dos GAs, dizer que uma solução A é mais adaptada do que uma outra solução B significa dizer que $f(A) < f(B)$ dado um mapeamento f , denominado função objetivo do problema. Esse tipo de técnica é amplamente aplicado em problemas do dia-a-dia, do estabelecimento de rotas de transporte ao despacho de energia elétrica. Na literatura eles são classificados como problemas do tipo \mathcal{NP} . Este conceito será esclarecido mais adiante, mas por enquanto, dizer que um problema p pertence à classe \mathcal{NP} significa que resolvê-lo requer testar todas as possíveis soluções exaustivamente.

Uma outra área da ciência da computação que se inspira em conceitos populacionais e dinâmicas sociais é o estudo de sistemas complexos, principalmente apoiado pela modelagem de sistemas multi-agentes (MAS). Neste tipo de sistema agentes podem interagir com o ambiente, o que inclui outros agentes, aprender a desempenhar uma determinada tarefa e se adaptar ao mundo em que estão inseridos (HOLLAND, 2014). Este tipo de abordagem é usada tanto para simular sistemas biológicos e estudar os modelos populacionais baseados na modelagem das interações entre os indivíduos, como para simular sistemas físicos, redes de tráfego, e sistemas distribuídos Dorri, Kanhere e Jurdak (2018).

Como ambas as áreas de pesquisa mencionadas acima se baseiam no mesmo conjunto de fenômenos, é possível traçar paralelos entre a área de otimização, em especial a dos algoritmos bioinspirados, os sistemas multi-agentes, e os conceitos de colaboração e diversidade que introduzem o assunto deste capítulo. Retomando o exemplo do algoritmos genéticos, esta técnica é munida de um mecanismo de mutação, que mantém a diversidade

na população. Uma população pouco diversa pode levar o algoritmo a ficar preso em soluções ótimas locais, que em sua vizinhança são melhores do que as outras soluções mas não são ótimos globais. O mecanismo de cruzamento do GA pode ser visto como uma estratégia de colaboração entre os indivíduos da população. Essas interações de mutação e cruzamento podem se dar de diversas maneiras, a depender da implementação do algoritmo, porém elas são codificadas de maneira estática. Por sua vez, a técnica de modelagem baseada em agentes pode ser usada para modelar uma rede de tráfego, por exemplo. Esta rede pode ter uma diversidade de veículos, com diferentes tamanhos, capacidades motoras (*drones*, carros, caminhões, trens). Os veículos podem colaborar entre si trocando mensagens usando sinais luminosos ou de rádio. O sistema de comunicação entre os agentes pode ser tão complexo quanto possível, mas no caso de sistemas de transporte, a diversidade do sistema também é estática e limitada.

Como já observado, os padrões de colaboração e diversidade presentes nos algoritmos bioinspirados são limitados de modo que, normalmente, eles são desenvolvidos e calibrados para uma família de problemas. Dificilmente um algoritmo projetado para problemas combinatórios tem um bom desempenho em problemas contínuos (BURKE et al., 2003). Este fato, na verdade, é uma consequência do "*Não há almoço grátis*" para otimização, proposto inicialmente por Wolpert e Macready (1997). Neste sentido Burke et al. (2003) indica que a composição de várias heurísticas para problemas combinatórios, em um algoritmo denominado hiper-heurística, é uma alternativa promissora para contornar as limitações de heurísticas isoladas.

Dokeroglu et al. (2019), em sua revisão recente da literatura, indica que metaheurísticas híbridas reportam uma melhora significativa comparadas com as respectivas implementações clássicas. Segundo o autor, a combinação de meta-heurísticas populacionais com técnicas de busca local é um dos mais bem sucedidos métodos presentes na literatura. Apesar da técnica de hibridização de metaheurísticas ser promissora, ainda assim é possível encontrar problemas técnicos e fundamentalmente teóricos. As principais dificuldades técnicas são a necessidade do conhecimento do problema, e uma vez desenvolvida, é necessário calibrar os parâmetros numéricos do algoritmo, que podem ser muitos (STÜTZLE; LÓPEZ-IBÁÑEZ, 2018). Além disso, são incontáveis as metaheurísticas que surgiram nos últimos 20 anos na literatura (DOKEROGLU et al., 2019). Combiná-las todas manualmente e verificar o seu desempenho em diferentes tipos de problemas não parece uma abordagem simples. Por sua vez, há uma dificuldade teórica no ponto em que metaheurísticas híbridas ainda tem um padrão de interação e comunicação estático. Normalmente, a busca local é executada após um número determinado de iterações da busca populacional com o objetivo de refinar as soluções encontradas até o momento. Este tipo de comunicação entre uma busca local e um algoritmo populacional poderia ser encarado como uma comunicação estática, o que parece dificultar, do ponto de vista teórico, o sistema a se adaptar a diferentes problemas.

Uma alternativa ao desenvolvimento de novas meta-heurísticas, ou a hibridização manual delas, é utilizar os mecanismos de colaboração e comunicação próprios da modelagem de sistemas multi-agentes (MMAS) para promover uma hibridização dinâmica de diferentes estratégias de busca (GONG et al., 2015; ZHENG; WANG, 2015; FERNANDES et al., 2009; MILANO; ROLI, 2004). São várias as arquiteturas e *frameworks* disponíveis na literatura que alcançaram bons resultados utilizando a MMAS, entre elas destacam-se: AGE, CMA, JABAT, MACS, MANGO, AMAM (SILVA et al., 2018) e a D-Optimas, objeto de estudo do presente trabalho.

A arquitetura D-Optimas é uma evolução da BIMASCO (*BIO-Inspired Multi-Agent System for Combinatorial Optimization*, proposta inicialmente por Junior (2010)). À época, o objetivo do autor era encapsular diferentes estratégias de busca em agentes, dotados de um mecanismo de aprendizagem bio-inspirada, de modo que eles pudessem colaborar e aprender a investigar o espaço de busca. A primeira versão da arquitetura era baseada em *threads*, utilizando comunicação assíncrona, onde os agentes interagem com espaço de busca de maneira não determinística. Outro conceito interessante apresentado no trabalho é a segmentação do espaço de busca em regiões de interesse. As interações entre os agentes e regiões permitiria uma hibridização dinâmica das meta-heurísticas, voltadas para um problema específico, sem um grande esforço de configuração e calibração de parâmetros.

Souza (2014) tornou a arquitetura genérica, propondo as abstrações necessárias para desacoplar o funcionamento das meta-heurísticas das particularidades de cada problema. Dando continuidade ao trabalho, Santos (2015) criou o modelo de comunicação entre os agentes, permitindo que eles pudessem colaborar, e verificou que de fato essa colaboração leva a arquitetura a encontrar soluções de melhor qualidade. Além disso, o autor propôs uma primeira implementação das regiões, dando ao espaço de busca uma dinâmica temporal. Entretanto, os autores tiveram dificuldades em executar experimentos de larga escala e produzir grandes volumes de dados. Isso impediu que análises estatísticas mais robustas fossem conduzidas. Essas dificuldades estavam ligadas principalmente à uma limitação tecnológica do modelo de concorrência utilizado, baseado em *threads* e compartilhamento de memória.

Pacheco (2017) então propôs remodelar a arquitetura BIMASCO, mudando o mecanismo de concorrência utilizado, a fim de contornar a dificuldade de execução em problemas de larga escala. Essa nova versão da arquitetura, chamada de D-Optimas, foi implementada utilizando a biblioteca *akka*, que implementa o modelo de atores (HEWITT, 2013). Este é um modelo de concorrência mais moderno, baseado em troca de mensagens assíncronas, e não bloqueantes. Esse novo paradigma permitiu a execução de centenas de agentes/regiões em um *cluster* com 2 nós. Entretanto, não foi possível verificar o desempenho da arquitetura em *clusters* maiores. O principal impedimento de estudos mais complexos era principalmente

a falta de resiliência a falhas, que são mais comuns à medida que os sistemas distribuídos crescem.

Sendo assim, a execução da arquitetura D-Optimas ficou restrita à um número limitado de nós em um *cluster*, bem como à uma pequena diversidade de agentes. Estendê-la, permitindo a execução em um *cluster* com um número indeterminado de nós, é essencial para a resolução de problemas em larga escala, bem como para extração de dados confiáveis necessários na análise de desempenho da arquitetura. Além disso, incluir novos algoritmos é essencial na criação de um sistema multiagente distribuído aplicado na solução de problemas de otimização.

1.1 Motivação

Existem vários problemas da vida prática que, apesar de parecerem complexos, de larga escala e difíceis de resolver, podem ser resolvidos em milésimos de segundos, mesmo em computadores cuja as configurações são muito básicas. Por exemplo, para navegar por uma cidade desconhecida, em que o tráfego varia muito nos horários de pico, hoje é possível abrir um aplicativo no celular que calcula a rota com o menor custo até o destino em questão de segundos. Para calcular essa rota, um algoritmo de caminho mínimo (XU et al., 2007) investiga as várias possíveis ruas, analisa dados obtidos de outros usuários, dados fornecidos pelo sistema de trânsito da cidade, e chega ao resultado ótimo com um tempo muito pequeno, que é normalmente uma função polinomial do tamanho da entrada. Neste caso, o tamanho da entrada pode ser considerado o como o tamanho da cidade, a quantidade de ruas e de cruzamentos.

Este problema clássico apresentado acima é considerado um problema de otimização, no qual o objetivo é encontrar um subconjunto dos dados de entrada que levam a um valor mínimo de uma função. Quando a relação entre o tempo para resolver o problema e o tamanho da entrada é uma função polinomial, a literatura classifica esse problema como \mathcal{P} (KARP, 1972). Entretanto, nem todo problema pode ser resolvido em tempo polinomial por um computador convencional. Por exemplo, é possível modificar ligeiramente o problema acima tornando-o mais complexo e difícil de ser resolvido. O motorista do veículo poderia ser um entregador que precisa parar em uma lista de destinos, e quer saber a rota com menor distância que passe por todos os destinos, sem repetir nenhuma rua ou avenida. Neste caso, a literatura desconhece uma estratégia melhor do que investigar sistematicamente cada uma das possíveis combinações de ruas da cidade. A relação entre o tempo e o tamanho da entrada neste caso é exponencial. Resolver este problema de forma exata para uma entrada pequena, por exemplo, um bairro com cinquenta ruas, é computacionalmente impraticável, podendo levar aproximadamente nove décadas para ser resolvido. A literatura classifica problemas deste tipo como \mathcal{NP} (LI, 2015).

Para resolver problemas da classe \mathcal{NP} a literatura propõe o uso de heurísticas, estratégias de busca focadas para resolver um problema específico, e meta-heurísticas que são estratégias de busca genéricas. Essas estratégias podem ter inspiração biológica, *e.g.* o algoritmo genético (WHITLEY, 1994), ou serem baseadas em outros fenômenos ou propriedades do problema (GRASP, recozimento simulado, são dois exemplos de metaheurísticas sem qualquer inspiração biológica). As metaheurísticas normalmente utilizam procedimentos estocásticos para produzir e modificar soluções para o problema e caminhar em direção a um valor ótimo, que pode ser um ótimo global ou não. Algumas destas estratégias de busca possuem vários parâmetros de configuração, o que as vezes dificulta a sua utilização, pois os parâmetros afetam diretamente a convergência do algoritmo.

Por essas características estocásticas inerentes às metaheurísticas, não existe nenhum método que seja bom em todos os problemas de otimização (EIBEN; SMITH, 2015, p. 30-32)(WOLPERT; MACREADY, 1997). Uma alternativa a essa limitação das metaheurísticas é unir características positivas de diferentes algoritmos em um único. Esta técnica, conhecida como hibridização, mostra-se ser mais robusta na solução de problemas mais complexos. De toda forma, a hibridização de meta-heurísticas normalmente leva a um número maior de parâmetros de configuração, o que torna o ajuste de parâmetros mais complexo e necessário.

1.2 Justificativa

A arquitetura D-Optimas tem um grande potencial para ser aplicada em problemas de otimização de larga escala. Ela se consolidou como um sistema multi-agentes baseado em um modelo de concorrência mais moderno ¹ e foi construída sobre uma biblioteca consolidada pelo mercado, a plataforma Akka. Além disso, possui vários algoritmos implementados e adaptados para funcionarem em qualquer problema em um ambiente distribuído. Conta com mecanismos de aprendizagem, de memória e de colaboração entre os agentes. Todos esses mecanismos trabalhando em conjunto permitem uma hibridização dinâmica das estratégias de busca, criando uma hiper-heurística sob demanda para um dado um problema de otimização.

Sobretudo, Santos (2015) obteve resultados interessantes em seu trabalho, podendo observar uma melhora da performance da arquitetura quando os agentes colaboram entre si. Entretanto uma das dificuldades enfrentadas no trabalho estava relacionada a performance da arquitetura, que naquele momento só podia executar em um único nó. Pacheco (2017) obteve sucesso em desacoplar o funcionamento das regiões dos agentes em uma primeira versão da arquitetura D-Optimas utilizando o modelo de atores. Entretanto a execução da arquitetura nessa versão ainda estava confinada a dois nós, e não contava com nenhum

¹<https://www.reactivemanifesto.org/>

mecanismo de tolerância a falhas. Desta forma, justifica-se a melhoria do modelo distribuído, uma vez que ela permitirá aumentar a escala dos problemas estudados na arquitetura D-Optimas, bem como ter mais confiança nos resultados obtidos.

Por sua vez, a literatura reporta sucesso na solução de problemas complexos utilizando sistemas multi-agentes que colaboram e formam times. Por exemplo, Marcolino, Jiang e Tambe (2013) estudou o efeito da diversidade de agentes que votam em um problema complexo, conseguindo resultados melhores que um agente especialista no problema (no caso estudado, os agentes deviam colaborar para jogar o jogo de tabuleiro *Go*). Deste modo, é possível levantar também a hipótese de que a diversidade de metaheurísticas no contexto da arquitetura D-Optimas possa ser um fator relevante para encontrar boas soluções para um problema de otimização.

Deste modo, estender a arquitetura D-Optimas como um sistema distribuído resiliente a falhas, permitirá executá-la para solucionar diferentes problemas de otimização nos quais a maioria dos algoritmos clássicos enfrenta dificuldade em encontrar bons resultados. As técnicas de hibridização de meta-heurísticas obtém melhores resultados neste tipo de problema, mas normalmente requerem algum conhecimento prévio do problema e um ajuste fino de parâmetros. Estudar um modelo automático capaz de hibridizar metaheurísticas configura-se um tópico de pesquisa amplo na área de otimização. Além disso, esta pesquisa permitirá a execução da arquitetura em infraestruturas modernas, como os provedores de nuvem, sendo possível disponibilizá-la como um *Software as a Service* (SaaS).

1.3 Objetivos

O objetivo geral deste trabalho é consolidar a arquitetura D-Optimas do ponto de vista de um sistema distribuído, tolerante a falhas, com balanceamento de carga e transparência de localidade, tornando-a resiliente e escalável horizontalmente. Com isso será possível executar simulações com problemas de larga escala em um *cluster* com uma variedade de agentes e estratégias de busca. Espera-se observar a adaptação da arquitetura ao problema, com os agentes colaborando e, dessa forma, verificar, neste comportamento, o surgimento de uma hibridização dinâmica das meta-heurísticas.

Para atingir este objetivo geral, é necessário cumprir os seguintes objetivos específicos:

- Compreender a arquitetura D-Optimas, seus fundamentos teóricos e seus mecanismos de generalização de problemas
- Estender arquitetura permitindo sua execução em um *cluster* com um número arbitrário de nós

- Aprimorar os algoritmos da arquitetura, em especial os que definem o comportamento das regiões, para que não dependam de um conhecimento prévio do problema
- Estender a D-Optimas, adicionando novas meta-heurísticas
- Submetê-la a uma bateria de experimentos com diferentes problemas, a fim de obter resultados confiáveis
- Submeter os dados a uma análise estatística adequada

1.4 Organização do Trabalho

Situado o contexto deste trabalho, o Capítulo 2 fará uma revisão das arquiteturas de software encontradas na literatura que possuem características em comum com a D-Optimas. As principais características levadas em conta nesta seleção foram o uso da modelagem multi-agentes e a possibilidade de execução em sistemas multi-cores ou *clusters*. Este capítulo também revisita o histórico da arquitetura D-Optimas, desde a sua origem na arquitetura BIMASCO, apresentando detalhes dos seus componentes e funcionamento.

O Capítulo 3 apresenta a metodologia adotada no desenvolvimento deste trabalho. Neste capítulo, a principal hipótese que fundamentou a construção da arquitetura D-Optimas é retomada, a saber: que a diversidade de estratégias melhora o desempenho da arquitetura em diferentes problemas de otimização. As premissas que contornam o problema de pesquisa são delineadas. Entre elas, a principal, de que a estocasticidade da arquitetura impede na maioria das vezes o uso de testes estatísticos paramétricos. É também apresentado o procedimento experimental e o esquema de coleta de dados.

A nova organização da arquitetura D-Optimas é apresentada no Capítulo 4. Esta nova versão utiliza outras ferramentas importantes da biblioteca *akka*, a saber, o *akka-cluster*, *akka-sharding* e *akka-persistence*, que permitem aos agentes manterem um estado distribuído e persistente da arquitetura. O novo funcionamento das regiões é detalhado, e a adaptação de novas meta-heurísticas à arquitetura é descrita.

Por fim, o Capítulo 5 apresenta os resultados obtidos em experimentos preliminares, para avaliar a escalabilidade e funcionamento do novo modelo da arquitetura e o Capítulo 6 apresenta os resultados dos experimentos realizados para avaliar o efeito da diversidade de agentes na qualidade das soluções obtidas. O Capítulo 7 apresenta as conclusões do trabalho e os próximos passos para este projeto de pesquisa.

Capítulo 2

Revisão bibliográfica

Conto ao senhor é o que eu sei e o senhor não sabe; mas principal quero contar é o que eu não sei se sei; e que pode ser que o senhor saiba.

Riobaldo, em "Grande Sertão: Veredas"

2.1 Arquiteturas Multi-Agentes para Otimização

Silva et al. (2018) fazem uma detalhada revisão da literatura apresentando 25 arquiteturas de software voltadas para otimização. Os autores apresentam os trabalhos em duas categorias, a saber: *frameworks* para otimização que utilizam metaheurísticas e *frameworks* multi-agentes que utilizam metaheurísticas. Entre essas duas principais categorias, são analisadas características gerais e avançadas, características multi-agentes, e os recursos ao processo de otimização.

O que se propõe neste trabalho é desenvolver uma arquitetura de software multi-agente, distribuída, baseada no modelo de atores, e escalável. Neste sentido, há que se comparar este trabalho com trabalhos semelhantes apresentados na literatura. Dos 25 trabalhos apresentados por Silva et al. (2018), foram selecionados aqueles que são categorizados como um sistema distribuído em suas características avançadas, e que são sistemas multi-agentes.

Deste modo, as próximas subseções apresentam as arquiteturas AgE, CMA, JABAT, MACS e MANGO, com suas principais características. Adiante também haverá uma apresentação da arquitetura D-Optimas, objeto de estudo do presente trabalho, a seção seguinte fará um comparativo dos trabalhos apresentados anteriormente e como a versão atual do D-Optimas se destaca dentre as outras.

2.1.1 AgE

AgE, abreviatura para *Agent Evolution*, foi proposto por Cetnarowicz, Kisiel-Dorohinicki e Nawarecki (1996) e é definido como uma plataforma baseada em componentes, configurável, para a resolução de diferentes problemas (PIĘTAK et al., 2009). O autor apresenta um formalismo matemático para definir um modelo baseado em agentes e descreve como esse formalismo pode ser implementado utilizando componentes de software reutilizáveis. A implementação é feita em Java, baseado no *Pico Container*¹, um *middleware* para inversão de controle e injeção de dependências. O autor defende que é possível utilizar várias técnicas de otimização (PIĘTAK; KISIEL-DOROHINICKI, 2013), entretanto a única implementação concreta do *framework* é baseada em algoritmos genéticos e sistemas multi-agentes evolucionários (KISIEL-DOROHINICKI, 2004). Há outras implementações do AgE em diferentes linguagens, mas não foram encontrados trabalhos que apresentem resultados experimentais, ou que demonstrem o funcionamento ou vantagens do modelo.

2.1.2 CMA

Malek (2009) propôs o Algoritmo para Colaboração de Meta-heurísticas (CMA). Este é um *framework* para resolução de problemas de otimização combinatória. O sistema conta com quatro tipos diferentes de agentes. O primeiro deles, são agentes do tipo Problema, responsável por receber os dados de entrada do usuário e iniciar o processo de otimização. Há um agente responsável por manter o conjunto de soluções encontradas até o momento e fornecer essas soluções para os agentes otimizadores. A comunicação entre os agentes acontece de maneira indireta através desse agente chamado *SolutionPool*, não havendo de fato uma colaboração efetiva entre os agentes. Os otimizadores encapsulam as meta-heurísticas e, por fim, há um agente consultor que é responsável por calibrar e fornecer os parâmetros para cada agente. O autor apresenta resultados para algumas instâncias do Problema do Caixeiro Viajante (MALEK, 2010), mas sem uma análise estatística aprofundada, exibindo somente o melhor valor encontrado para cada instância.

2.1.3 JABAT

JABAT é um sistema multi-agentes no qual os agentes colaboram para gerar soluções de problemas de otimização combinatória. Ele é baseado no A-Teams e implementando com o suporte da biblioteca **JADE** (*Java Agent Development Framework*). Foi proposto inicialmente por Jdrzejowicz e Wierzbowska (2006) e a última atualização do trabalho, feita por Barbucha et al. (2010), apresenta duas novas versões da arquitetura: eJABAT que oferece uma interface Web e a cJABAT que inclui um mecanismo de aprendizagem por reforço positivo. Estas novas versões, de acordo com o autor, são escaláveis, portáteis e em conformidade com a **FIPA** (*Foundation of Intelligent Physical Agents*).

¹<http://picocontainer.com>

O autor apresenta também os problemas resolvidos pela arquitetura JABAT, a saber: *Resource Constrained Project Scheduling* (RCPSP) com e sem restrição de tempo, *Euclidean Planar TSP*, *Clustering* e distribuído e não distribuído, *Vehicle Routing Problem*, e *Data Reduction Problem* distribuído e não distribuído. Apesar da argumentação de que a arquitetura é genérica, cada problema é apresentado com um conjunto específico de heurísticas para resolvê-lo. O autor não deixa claro se é possível mudar a estratégia de solução de cada problema.

A arquitetura conta com um mecanismo de aprendizagem por reforço, que nesta configuração passa a funcionar de maneira síncrona. O mecanismo atribui pesos a cada agente que são atualizados à medida que cada um deles produz uma solução. Os pesos são utilizados em um sorteio ponderado para escolher que agente executará a cada instante de tempo. Entretanto, a arquitetura não é preparada para solucionar problemas multi-objetivo.

2.1.4 MACS

Martin et al. (2016) propôs um sistema multi-agente para busca cooperativa (do inglês *Multi-Agent Cooperative Search*, MACS) para a resolução de problemas de roteamento e fluxo. O sistema foi baseado no *JADE* e possui somente dois tipos de agente, denominados inicializador e agente meta-heurístico. O primeiro tipo é responsável por ler os dados do problema e convertê-los numa representação compatível com o protocolo de comunicação dos agentes do segundo tipo, que encapsulam um par de meta-heurística/busca local. A colaboração acontece entre os agentes através de um protocolo de comunicação denominado conversação. O autor compara os resultados utilizando um, quatro, oito e dezesseis agentes colaborando, e foi possível perceber diferença significativa (95% de confiança) no resultado da busca sempre que se dobra o número de agentes. Além disso, os resultados obtidos foram tão bons quanto os da literatura, sem a necessidade de ajuste de parâmetros. Entretanto, foram utilizados somente as heurísticas RandNEH (JUAN et al., 2010) e RandCWS (PÉREZ et al., 2010).

2.1.5 MANGO

O Ambiente Multi-Agente para Otimização Global (MANGO) apresentado por Kerçelli et al. (2008) é uma arquitetura baseada também na plataforma *JADE*. Cada agente encapsula uma meta-heurística e pode se comunicar com outros agentes, compartilhando e solicitando soluções, ou informando sobre áreas do espaço de busca que são propícias para exploração. A comunicação entre os agentes acontece de maneira assíncrona e não-determinística através de troca de mensagens. A implementação da arquitetura é fortemente acoplada ao problema de otimização e só possui a heurística EM (*Eletromagnetism-like Mechanism*, Mecanismo Eletromagnético em português) (BIRBIL; FANG, 2003). O autor apresenta

apenas uma execução para ilustrar o funcionamento da arquitetura, não apresentando um procedimento experimental planejado.

2.1.6 AMAM

O *framework* AMAM, proposto inicialmente por Silva et al. (2007) é um MAS composto por 5 tipos de agentes, a saber: os construtores, agentes de busca local, agentes que encapsulam meta-heurísticas, o agente coordenador e o analisador de soluções. Os três primeiros agentes são responsáveis por explorar o espaço de busca, enquanto o coordenador, que centraliza a troca de mensagens, é responsável por intermediar a comunicação entre os outros agentes. A primeira versão da AMAM foi proposta por Oliveira (2008) e estendida por Fernandes et al. (2009), que a equipou com uma memória adaptativa compartilhada. Nesta versão, a comunicação entre os agentes se dá por meio de uma área de memória compartilhada que mantém as melhores soluções a arquitetura em um dado momento. O acesso à essa área pelos agentes é feito de maneira síncrona.

A versão mais recente do AMAM abandonou o conceito de agente coordenador, se alinhando a melhor prática de modelagem de sistemas multi-agentes, em que a independência dos agentes é um princípio importante. Nesta versão, os agentes são dotados de uma memória de aprendizagem por reforço (SILVA et al., 2015). Os resultados neste trabalho indicam que há diferença significativa nos resultados encontrados pela arquitetura quando há cooperação entre os agentes, mesmo em uma simulação de pequeno porte, com somente quatro agentes.

O *framework* AMAM e a arquitetura BIMASCO tiveram a mesma origem, como relata a seção a seguir. A principal diferença entre os dois trabalhos é que a AMAM não faz uso de mecanismos de programação distribuída ou paralela e, em suas primeiras versões, utiliza um modelo de comunicação determinístico. Além disso, a arquitetura BIMASCO já suportava na sua última versão, diferentes tipos de problemas de otimização, enquanto a AMAM oferece suporte somente para problemas de otimização combinatória.

2.2 Histórico da Arquitetura D-Optimas

Um projeto mais recente, no qual o presente trabalho se baseia, é a arquitetura D-Optimas. Essa arquitetura foi apresentada por Pacheco (2017), mas é fundada num projeto mais antigo, chamado BIMASCO, que possui um longo histórico e uma bibliografia densa. Esta seção é dedicada a fazer uma pequena revisão do projeto, desde sua origem no trabalho de Junior (2010), passando pelos trabalhos de Souza (2014) e Santos (2015), até a sua transformação em uma arquitetura distribuída para solução de problemas de otimização. Serão apresentados os principais conceitos que compõem a arquitetura, os resultados obtidos

em cada uma de suas versões, e as suas eventuais falhas conceituais.

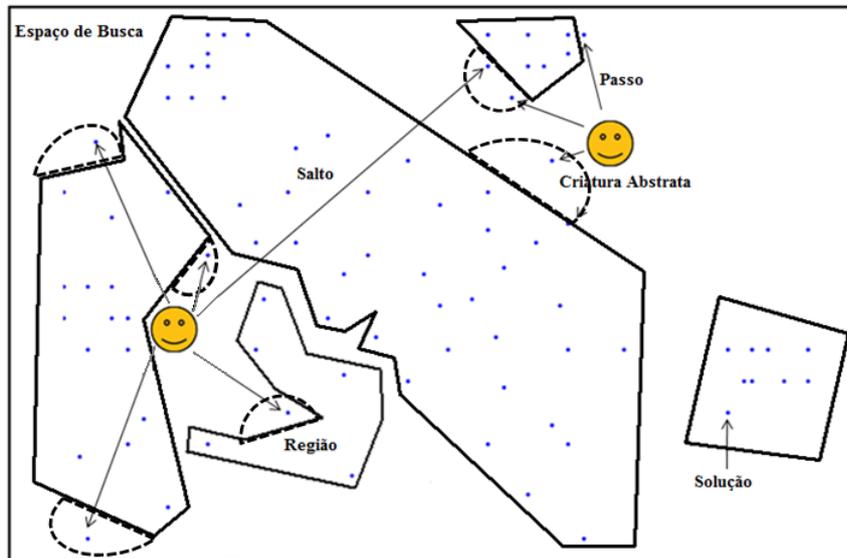
A arquitetura BIMASCO (*Bio Inspired Multi Agent System for Combinatorial Optimization*) foi primeiramente apresentada por Junior (2010) como uma completa reconcepção da arquitetura AMAM, segundo o próprio autor. Foi desenvolvida no Laboratório de Sistemas Inteligentes do CEFET-MG e a justificativa para a sua criação foi baseada no fato de que a AMAM, como já apresentado na seção anterior, feria alguns princípios da modelagem de sistemas multi-agentes, principalmente o princípio da autonomia, prevendo em seu modelo um agente controlador que determinava o comportamento dos demais agentes. O objetivo do trabalho àquele momento foi implementar um sistema multi-agente que funcionasse por troca de mensagens assíncrona, de modo que os agentes pudessem interagir entre si e com o espaço de busca, tomando decisões de maneira autônoma, funcionando como um sistema auto-organizável (JUNIOR, 2010).

Para o desenvolvimento do seu trabalho o autor se baseou numa outra arquitetura de software, essa voltada para a simulação de vida artificial, chamada ARTÍFICE. Esse projeto foi desenvolvido por cerca de 15 anos no mesmo laboratório. Seu objetivo era modelar, construir e simular criaturas artificiais com sistema nervoso central, e era fortemente fundamentado em conceitos biológicos e da cognição situada. Essas criaturas deviam ser capazes de aprender, se adaptar, bem como viver em um mundo artificial (SANTOS, 2003). Uma lista não exaustiva dos principais trabalhos que compõe a arquitetura ARTÍFICE são: Campos (2006), quem propôs a primeira implementação, Silva (2008), que apresentou um mecanismo de aprendizagem por reforço e Mapa (2009), que adicionou ao software um sistema de memória episódica. Destes projetos, o BIMASCO herda sua característica mais inovadora à época, seu caráter bio-inspirado. Ainda hoje, olhando para a literatura mais recente, em especial a revisão feita por Silva et al. (2018) não se tem relato de nenhum trabalho que dialogue com áreas do conhecimento tão diversas.

Feita esta pequena introdução sobre o histórico do projeto, passemos às contribuições da primeira implementação do BIMASCO proposta por Junior (2010). Além da inovação de um sistema multi-agente bio-inspirado para otimização, baseado em programação *multi-threading*, que permitia escalar o sistema verticalmente², o autor apresenta a arquitetura como uma metáfora biológica composta de 3 entidades: agentes, regiões e o mundo. O **mundo artificial** é o espaço de busca, composto das **soluções** de um problema de otimização, que podem ser agrupadas em **regiões** de interesse. Essas regiões não são necessariamente estáticas, podendo contrair, expandir, particionar e se unir a outras regiões. A Figura 1 ilustra didaticamente o sistema proposto pelo autor.

²Entende-se aqui por escalabilidade vertical a capacidade de aumentar o sistema adicionando mais recursos na mesma máquina. Mais adiante será mencionado o conceito de escalabilidade horizontal que é a capacidade de um sistema crescer a medida que se adicionam mais máquinas à configuração, como em um *cluster*, por exemplo.

Figura 1 – Representação gráfica do projeto BIMASCO, onde é possível distinguir os agentes, regiões, soluções no espaço de busca, bem como os possíveis movimentos de um agente



Fonte: Junior (2010)

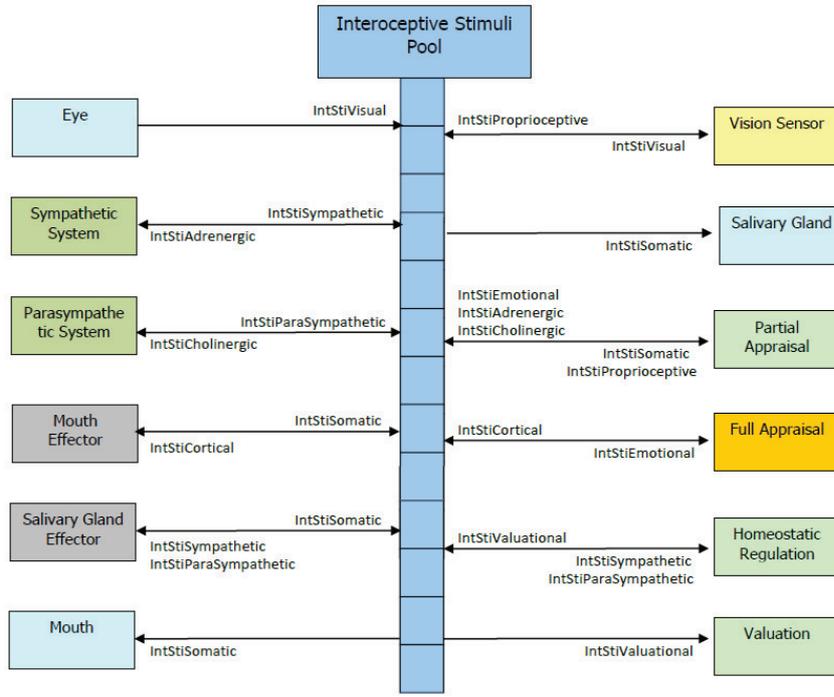
Neste ponto é necessário ressaltar que apesar do autor prever a divisão do espaço em regiões, e introduzir um comportamento básico para elas, ele não entra em detalhes do seu funcionamento e também não as implementa.

Os **agentes** são formados pela união de vários componentes. Entre os principais estão os mecanismos de aprendizagem, componentes emocionais, efetores e sensores. Além disso, encapsulam as heurísticas e meta-heurísticas e interagem com esse espaço de busca, produzindo e consumindo soluções do mundo, aprendendo a buscar melhores soluções. Cada componente do agente executava em uma *thread* separada, que se comunicavam por troca de mensagens, por meio de uma área de memória compartilhada, chamada de *Interoceptive Pool*. A Figura 2 ilustra os diversos componentes que compõe o agente, incluindo os sensores, efetores, componentes do sistema nervoso artificial, bem como a comunicação entre eles por meio de memória compartilhada.

O objetivo é que os agentes aprendam a encontrar boas soluções para um problema de otimização baseado nessa interação e na colaboração entre eles. A interação com o espaço de busca acontece também por meio de troca de mensagens, baseada em compartilhamento de estado, por meio do *Environmental Pool*. A Figura 3 ilustra o funcionamento do *Environmental Pool*, que é basicamente uma lista compartilhada onde agentes e regiões podem colocar mensagens endereçadas a outras entidades, bem como consultar periodicamente se há mensagens endereçadas para si.

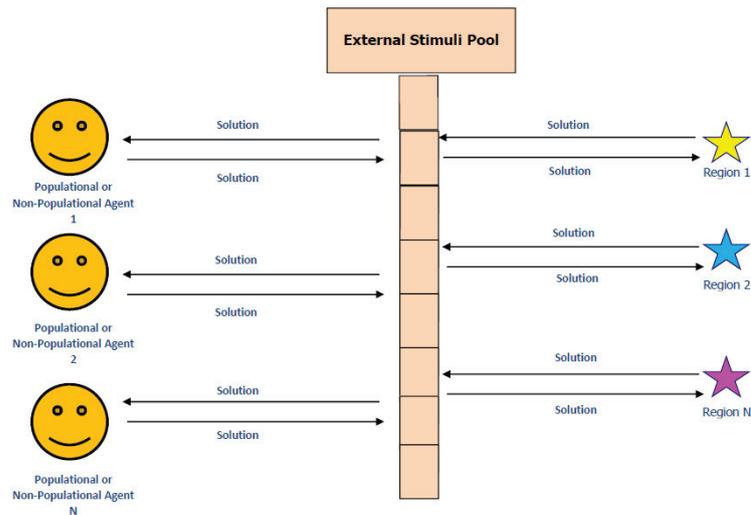
Os agentes podem ser divididos em construtores, populacionais e não-populacionais,

Figura 2 – Esquema dos componentes internos do agente, os estímulos, ou mensagens que eles trocam entre si, e o *Interoceptive Pool*, memória compartilhada utilizada para a troca de mensagens



Fonte: Junior (2010)

Figura 3 – Mensagens trocadas entre agentes e regiões através do *Environmental Pool*.



Fonte: Junior (2010)

baseado no funcionamento de cada um e em suas interações com o espaço de busca. Um agente construtor, por exemplo, não precisa consumir nenhuma solução inicial do mundo, pois constrói uma solução, elemento a elemento, baseado em alguma estratégia, que pode ser gulosa ou randômica. Um agente populacional encapsula uma estratégia de busca baseada em populações, *e.g.*, um algoritmo genético, e por isso necessita de um conjunto inicial de soluções para funcionar. Assim, o agente busca soluções iniciais no espaço de busca, processa e entrega um conjunto melhorado de soluções ao mundo. Agentes não-populacionais são aqueles que realizam buscas locais, e necessitam de uma solução para iniciar a busca, produzindo uma solução que pode ser um ótimo local, ou um ótimo global.

Nesta primeira versão da arquitetura estavam presentes algumas estratégias de busca conhecidas da literatura, são elas: a heurística construtiva GRASP (*Greedy Randomized Adaptive Search Procedure*), os métodos de busca local ILS (*Iterated Local Search*) e VNS (*Variable Neighborhood Search*) e a meta-heurística populacional GA (*Genetic Algorithm*). Para avaliar o funcionamento da sua arquitetura, Junior (2010) executou alguns experimentos com instâncias do Problema de Roteamento de Veículos com Janela de Tempo (PRVJT), propostas por Solomon (1987), e compara os seus resultados com os da arquitetura AMAM. Entretanto, ele não apresenta um procedimento experimental adequado para um sistema não-determinístico, como é o BIMASCO, e muito menos uma análise estatística rigorosa, prejudicando uma comparação entre as arquiteturas.

Apesar de Junior (2010) ter projetado o BIMASCO para ser facilmente extensível e flexível tanto para os métodos de busca quanto para os problemas, foi somente no trabalho de Souza (2014) que a arquitetura foi estendida para que as heurísticas fossem completamente independentes da representação e dados dos problemas. Para tanto, a autora tornou as soluções genéricas quanto ao tipo de dado, podendo ele ser booleano, inteiro ou real. Também criou artefatos de software para abstrair as estratégias que atuam sobre as soluções e que são comuns a vários algoritmos de busca. Esses artefatos foram nomeados **movimentos**, que consistem na alteração de uma posição do vetor no espaço de objetivos, levando a solução para um vizinho próximo, e **modificadores**, que podem ser vistos como um conjunto de movimentos. Um movimento ou um modificador pode ser aplicado a uma ou mais soluções, e foram implementados para as principais heurísticas e para os tipos de dados propostos. A autora deixa claro que "[...] os modificadores são totalmente dependentes do tipo de representação do problema e não do problema em questão"(SOUZA, 2014).

Nesta versão do BIMASCO novos problemas foram adicionados à arquitetura, a saber: o problema da otimização irrestrita de função real, da diversidade máxima, da mochila e o problema da partição binária. Foi adicionada a heurística *Simulated Annealing* e as

heurísticas existentes foram adaptadas para funcionarem de maneira independente da representação das soluções, utilizando somente os movimentos e modificadores propostos. Em seguida, foram apresentados três conjuntos de experimentos para validar a generalização da arquitetura executando diferentes combinações de problemas e heurísticas, bem como o correto funcionamento das heurísticas. A arquitetura alcançou bons resultados para a maioria dos problemas, apesar do efeito dos mecanismos de aprendizagem não ter sido avaliado, e a implementação dos comportamentos para as regiões foi elencada somente na seção de trabalhos futuros.

Dando prosseguimento no trabalho de Souza (2014), Santos (2015) simplificou a arquitetura, removendo a maioria dos componentes herdados da arquitetura ARTÍFICE que oneravam a simulação, tornando-a mais lenta e menos competitiva que outras arquiteturas. Deste modo, o autor propõe um mecanismo simplificado baseado em aprendizagem por reforço, um protocolo de comunicação entre os agentes, e uma primeira implementação da dinâmica das regiões. Nesta nova versão, os agentes passam a ser compostos somente de 3 *threads*: um componente sensor, efetor e o *Evaluator*. Por meio do componente sensor, os agentes podem solicitar soluções a outros agentes e às regiões, com uma probabilidade associada. Essa probabilidade é alterada a medida que o agente interage com o mundo e produz soluções melhores do que as produzidas anteriormente. Essas soluções são avaliadas pelo *Evaluator*, que altera as probabilidades na memória do agente.

Para escolher, dentre os agentes e regiões na memória, de qual solicitar soluções, o algoritmo funciona basicamente em duas etapas. Inicialmente, ele faz uma roleta entre todos os agentes e reserva uma fatia dessa roleta para as regiões. Caso algum agente seja sorteado, é enviado uma mensagem diretamente para ele. Mas caso a escolha recaia sobre as regiões, é feita uma nova roleta entre as regiões, e o agente pede soluções para a região escolhida. O agente utiliza as soluções recebidas para executar a meta-heurística que encapsula e produzir novas soluções, que retornam para o mundo. O componente efetor é responsável por enviar essas soluções produzidas para uma região. Entretanto, Santos (2015) não dá detalhes sobre como um agente escolhe para qual região enviar essas novas soluções.

A estratégia proposta para a fusão e fissão das regiões ficou baseada na análise da variância e covariância dos valores de função objetivo dos conjuntos de soluções que formam as regiões. Para se dividir, uma região verifica se a variância da função objetivo está acima de um limiar que é dado como parâmetro no arquivo de configuração da simulação. Se a σ^2 for maior que este parâmetro, a região se particiona em 3 novas regiões contendo as soluções com valor de função objetivo que estavam abaixo de $\mu - \sigma$, entre $\mu - \sigma$ e $\mu + \sigma$ e as que estavam acima de $\mu + \sigma$. Para duas regiões fazerem fusão, a covariância entre as duas tem de ser maior que um limiar, que também é um parâmetro da configuração da arquitetura. É importante observar que o autor não mostra detalhadamente como essa análise é feita.

Por exemplo, a fórmula da covariância entre duas amostras X e Y de tamanho N com médias \bar{X} e \bar{Y} pode ser escrita da seguinte maneira:

$$cov(X, Y) = \sum_{i=1}^{i < N} \frac{(x_i - \bar{X})(y_i - \bar{Y})}{N - 1} \quad (1)$$

Aplicando a Equação (1) ao contexto da fusão entre duas regiões, seria necessário que ambas tivessem o mesmo número N de soluções. Entretanto, pela característica estocástica do sistema, muito raramente duas regiões terão o mesmo número de soluções. Esse caso de borda não foi abordado na modelagem do autor, muito menos nos experimentos realizados.

Neste trabalho foram executadas 5 baterias de experimentos dos quais 4 estavam voltados para avaliar o desempenho da arquitetura com e sem colaboração entre os agentes (permitindo ou não que os agentes troquem soluções) e 1 voltado para avaliar o efeito de utilizar uma ou mais regiões na simulação, alterando somente os limiares de fusão e fissão. Na maioria dos experimentos foi observado que a interação entre os agentes e a aprendizagem favorecem a busca por melhores soluções. No entanto, não foi possível tirar alguma conclusão sobre o efeito da separação do espaço em regiões para o funcionamento da arquitetura.

A falta de experimentos em larga escala com a arquitetura BIMASCO motivou Pacheco (2017) a propor uma nova implementação da arquitetura, desta vez baseada num modelo de concorrência mais moderno, denominado modelo de atores. Proposto inicialmente por Hewitt, Bishop e Steiger (1973), o modelo de atores se contrapõe ao modelo de concorrência por *threads* e compartilhamento de estado. Os atores, unidades básicas de computação, não compartilham o estado interno e se comunicam somente por meio de troca de mensagens assíncronas. O autor utilizou como ferramenta o *framework Akka*³, uma implementação do modelo de atores para linguagens baseadas na JVM, como o Java.

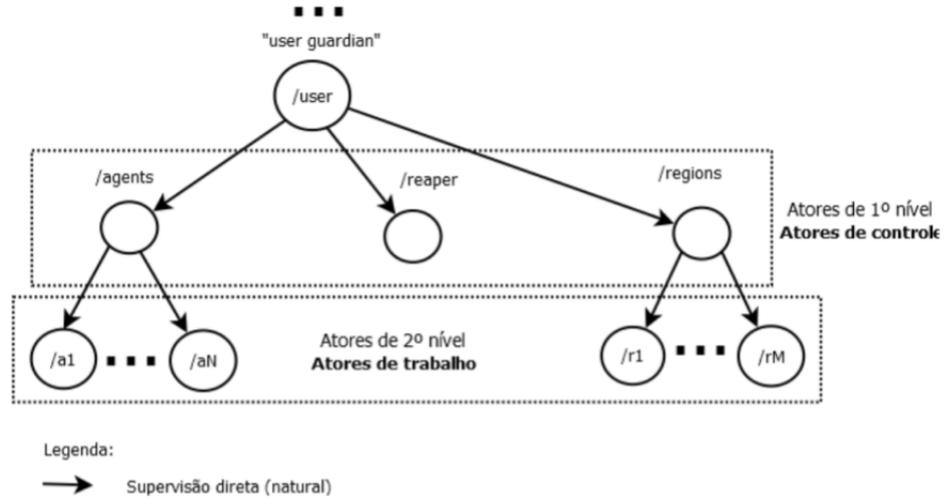
A mudança de paradigma oferece à arquitetura remodelada a vantagem de poder escalar horizontalmente, como um sistema distribuído, o que inclusive leva à mudança do nome do projeto, que deixa de se chamar BIMASCO e passa a se chamar D-Optimas. Os agentes e regiões também foram reduzidos a um único ator que executa a regra de negócio principal, mas sem perderem as suas funcionalidades descritas nos trabalhos anteriores.

A Figura 4 ilustra a hierarquia de atores proposta por Pacheco (2017). O ator */user* faz parte do *toolkit Akka* e supervisiona todos os atores criados pelo programa, os atores */agents* e */regions* criam, supervisionam e finalizam a execução de agentes e regiões, respectivamente. O ator */reaper* é o responsável por finalizar a simulação quando atingido o critério de parada. Uma maneira possível de executar essa arquitetura em um *cluster*⁴ seria alocar o ator */agents* em uma máquina com o endereço *192.168.0.1*, e o */regions* no

³<https://akka.io/>

⁴No contexto da computação distribuída, um *cluster* nada mais é do que um conjunto de computadores

Figura 4 – Hierarquia de atores proposta para a arquitetura D-Optimas.



Fonte: Pacheco (2017)

computador endereçado no IP *192.168.0.2*, ambos na porta *2551*. Para o agente *a1* se comunicar com a região *r1* é necessário que ele saiba o endereço completo do agente, que seria *akka://192.168.0.2:2551/user/regions/r1*.

O autor apresenta dois experimentos para avaliar o resultado da mudança: o primeiro, para verificar a corretude do sistema e certificar de que a mudança de paradigma não alterou nenhuma funcionalidade até aqui desenvolvida, e um segundo que avaliou a escalabilidade da arquitetura, coletando dados de 30 execuções com mais de 1000 agentes. Entretanto, o segundo e principal experimento não avaliou o funcionamento da arquitetura em mais de 2 nós, não exibindo nenhum dado sobre a escalabilidade horizontal do sistema.

2.3 Considerações Finais

O presente capítulo se dedicou a fazer uma revisão da literatura de arquiteturas multi-agentes para otimização. Foram apresentadas alguns trabalhos que compartilham características importantes, baseado principalmente na revisão mais recente realizada por Silva et al. (2018). Entre essas arquiteturas, foi apresentada também a arquitetura AMAM, proposta por Silva (2019) e por fim, a arquitetura D-Optimas. As suas principais características, resultado de diversas contribuições ao longo dos últimos 10 anos, foram exploradas, bem como suas eventuais deficiências, que se convertem em boas oportunidades de trabalho. O capítulo seguinte apresenta os esforços que foram feitos no presente trabalho para reparar

conectados através de uma rede, os quais podem ser chamados de membros, nós, *workers*, *slaves*, a depender do autor. O presente trabalho utiliza o termo nó, por não haver qualquer distinção entre o papel dos computadores em si.

essas deficiências e estender o projeto.

Capítulo 3

Metodologia

Pica pau voa é duvidando do ar.

Riobaldo, em "Grande Sertão: Veredas"

O capítulo anterior apresentou algumas arquiteturas e *frameworks* para resolução de problemas de otimização que podem ser executadas em um *cluster*. Entre elas, se destaca a arquitetura D-Optimas, baseada no sistema de atores. Do ponto de vista de sistemas distribuídos, essa arquitetura apresenta algumas dificuldades para executá-la em um *cluster* com mais de dois nós. Além disso, é possível observar também alguns aspectos da modelagem que tornam a configuração da arquitetura muito acoplada ao problema de otimização, principalmente no que diz respeito aos parâmetros das regiões.

Deste modo, o presente trabalho se dedica a propor melhorias na arquitetura D-Optimas, a fim de tornar o comportamento das regiões menos dependente de algum conhecimento prévio do problema e também possibilitar a escalabilidade horizontal da arquitetura. O presente capítulo é dedicado a expor a metodologia utilizada para atingir os objetivos propostos no Capítulo 1. A próxima seção apresenta as hipóteses que guiam este trabalho. Em seguida, na Seção 3.2 é apresentado o método utilizado para por as hipóteses à prova. A Seção 3.3 apresenta as premissas que podem ser assumidas no decorrer da metodologia, e por fim, a Seção 3.4 apresenta os experimentos que serão executados para verificar as hipóteses.

3.1 Hipóteses

Dentre os trabalhos que envolveram a arquitetura D-Optimas e sua antecessora, o Bimasco, se destacam as contribuições mais recentes de Santos (2015) e Pacheco (2017). As hipóteses levantadas e testadas por Santos (2015) foram de que a diversidade de agentes poderia favorecer a exploração do espaço de busca e que a segregação das soluções em regiões em

conjunto com um mecanismo de aprendizagem levaria os agentes a buscarem soluções de melhor qualidade. Por sua vez, Pacheco (2017) levantou a hipótese de que a mudança para o modelo de atores permitiria que a arquitetura escalasse de maneira vertical, em um único computador, e de maneira horizontal, ao longo de um *cluster*. De maneira mais geral, a hipótese que motiva a construção da arquitetura é de que nenhuma meta-heurística particular é capaz de encontrar bons resultados para todos os tipos de problemas, porém uma combinação adequada delas poderia gerar resultados melhores.

Sendo assim, torna-se necessário conduzir um estudo mais detalhado do comportamento da arquitetura ao longo de uma simulação, com a presença de uma maior diversidade de meta-heurísticas. É também necessário verificar o desempenho da D-Optimas em um *cluster* com vários computadores, onde a influência da rede é crítica e onde a escalabilidade horizontal poderia ser melhor verificada. Além disso, a implementação de um conjunto muito restrito de meta-heurísticas impediu uma investigação mais detalhada do efeito da diversidade no processo de busca. Espera-se que, corrigidas as falhas discutidas no capítulo anterior e adicionando novas meta-heurísticas, seja possível avaliar com maior robustez algumas destas hipóteses.

A primeira hipótese que se pode formular no presente trabalho é que a versão da arquitetura produzida exhibe um comportamento convergente em torno do valor conhecido da função objetivo dos problemas propostos na literatura. A segunda hipótese formulada é que o aumento do número de nós não acarreta em perda de desempenho ou diminuição da taxa de convergência da arquitetura. E por fim, é possível revisitar a hipótese levantada por Santos (2015), de que, ao inserir diversidade de estratégias de busca, combinada com um sistema de memória e um mecanismo de colaboração entre os agentes, a arquitetura é capaz de produzir soluções de melhor qualidade para diferentes problemas.

3.2 Método

Como já foi dito, o objetivo do presente trabalho é construir uma nova versão da arquitetura D-Optimas, baseado na versão proposta por Pacheco (2017). Este sistema multi-agentes deve ser construído sem um controle central, utilizando técnicas e ferramentas de sistemas distribuídos, que permitam a sua execução em um *cluster* com um número arbitrário de nós. Nesta nova versão, as regiões devem funcionar de maneira independente dos dados do problema objeto de estudo, necessitando o mínimo de conhecimento prévio acerca dele para configurar a arquitetura. Também pretende-se adicionar novas meta-heurísticas à arquitetura, adaptando-as para funcionar num contexto distribuído, a fim de poder obter resultados confiáveis para avaliar o desempenho do sistema em diferentes problemas de otimização.

Dito isto, a metodologia estabelecida para atingir os objetivos e por as hipóteses à prova consiste em, inicialmente, compreender a atual versão da arquitetura, desenvolvida por Pacheco (2017) sobre o *toolkit akka*, e verificar quais pontos têm dificultado sua execução em um *cluster* com mais de dois nós.

Para suplantar essas dificuldades, é necessário, em seguida, estudar os recursos do *toolkit akka*, em especial, as ferramentas que compõem o *akka-cluster*, o qual implementa todas as funcionalidades e recursos essenciais para o desenvolvimento de um sistema distribuído. A biblioteca *akka-cluster* foi escolhida em detrimento de várias outras alternativas pois a última versão da arquitetura já está implementada no modelo de atores, adotado na D-Optimas. Assim, adotar esta implementação dos algoritmos clássicos de sistemas distribuídos, tais como descobrimento de nós e eleição de líder, parece uma escolha natural e que leva ao menor retrabalho.

Após compreender a arquitetura D-Optimas e as ferramentas necessárias para a implementação de um sistema distribuído, é necessário refatorar a arquitetura para incluir os recursos do *akka-cluster*. As modificações nos protocolos de funcionamento serão inevitáveis, pois apesar de alguns atores supervisores já existirem, eles não foram preparados para chegar a um consenso sobre, por exemplo, para qual região uma solução deveria ir em um sistema distribuído. Nesta nova versão da arquitetura, atores supervisores do mesmo tipo precisarão manter uma comunicação frequente e um estado distribuído da aplicação, permitindo ao sistema como um todo ser resiliente a falhas.

Para além de refatorar a comunicação entre os atores supervisores, será necessário detalhar e implementar o funcionamento das regiões em concordância com os objetivos aqui propostos. A primeira versão das regiões implementada por Santos (2015) estava fortemente amarrada a um conhecimento prévio do problema. O que se propõe aqui é aprimorar a implementação, desvinculando-a dos dados do problema, e estudar o efeito macro dos comportamentos de fissão e fusão no comportamento/resultados gerados pela arquitetura. Além disto, Santos (2015) apontou como uma oportunidade de trabalho futuro a inserção de novas meta-heurísticas a arquitetura, uma vez que a escassez delas pode ter influenciado na performance de seus resultados. Assim, é necessário revisar a literatura, encontrar meta-heurísticas que sejam relevantes para este trabalho, estudar o seu funcionamento, adaptá-las ao contexto de um sistema multi-agente genérico no que diz respeito à representação dos problemas, e implementá-las neste sistema.

Para validar as hipóteses levantadas na seção anterior, experimentos serão executados e dados serão coletados, tratados e analisados. Todavia, num sistema distribuído e assíncrono, que opera por meio de troca de mensagens, a coleta de dados confiável, que preserva a sequência temporal dos mesmos está longe de ser tarefa trivial, como é em outros cenários. Neste contexto, será necessário propor e implementar um sistema de persistência e coleta

de dados adequado à um sistema distribuído. Neste tipo de sistema as entidades podem e, de fato migram de um nó para outro. Portanto, manter ativa e funcional uma conexão com um banco de dados centralizado, e muitas das vezes remoto, representa não apenas uma tarefa árdua, como também um possível gargalo/ponto único de falha. Além disto, como a escrita no banco de dados é uma operação muito mais dominante na D-Optimas em comparação com a leitura, é possível abrir mão dos princípios de ACID¹ e utilizar um banco de dados distribuído não-SQL. Ao longo da execução, os agentes e regiões escrevem no banco de dados os eventos relevantes (por exemplo, a produção ou recebimento de uma nova solução, uma operação de fusão ou fissão), e ao final da execução, esses dados são extraídos pela arquitetura, recebendo neste ponto um primeiro tratamento, como por exemplo, a ordenação, agrupamento e remoção de eventuais dados duplicados.

Ao final deste tratamento, os dados são salvos em um conjunto de arquivos texto do tipo *csv*. Para cada série de dados extraída, e para cada entidade, um novo arquivo é criado, *i.e.*, há pelo menos um arquivo para cada região/agente na simulação. Os dados extraídos são: as soluções produzidas por cada agente e o intervalo de tempo para produzir cada solução, as soluções recebidas por cada região, os parâmetros estatísticos das regiões, a melhor solução de cada região, a melhor solução global, as estatísticas globais e as mensagens trocadas. Todos esses dados são séries temporais discretas. Quanto ao sistema de banco de dados, nas versões anteriores era utilizado o Sistema Gerenciador de Banco de Dados (SGDB) **PostgreSQL**. Para a versão atual foi escolhido o **Cassandra**². Essa escolha se deu pois a biblioteca *akka-cluster* já possui um *plugin* para *journaling* e persistência do estado distribuído da aplicação, e também por ser um SGDB adequado à implementação de sistemas distribuídos. Por simplicidade de configuração e manutenção, foi feita a opção por utilizar um único sistema de banco de dados.

Para executar a arquitetura no *HPC* do CEFET-MG é necessário preparar um ambiente que é complexo, alocar os recursos computacionais, prover arquivos de configuração para o banco de dados, limpar os arquivos temporários que permaneceram nos nós de execuções passadas, iniciar e popular o banco com as tabelas, executar o experimento, extrair esses dados e, por fim, copiar os *logs* da execução. Sobretudo, é necessário garantir que todo experimento inicie sob as mesmas condições, de modo a evitar qualquer viés na coleta dos dados. Para configurar o ambiente, executar e armazenar os dados é necessário produzir uma série de *scripts* de configuração que seguem o padrão do escalonador do *HPC*, no caso do CEFET-MG, o SLURM.

¹Sigla para Atomicidade, Consistência, Isolamento e Durabilidade. Estas quatro propriedades são fortemente garantidas em bancos de dados relacionais, o que permite o desenvolvimento de aplicações confiáveis para o usuário final. No contexto do desenvolvimento de sistemas distribuídos, é comum abandonar a premissa de consistência e adotar uma consistência eventual, onde, em algum momento no tempo, todas as instâncias do banco de dados terão o mesmo conjunto de dados.

²<https://cassandra.apache.org/>

Uma vez alcançados todos estes passos que dizem respeito ao desenvolvimento tecnológico do problema proposto, há que efetivamente executar a bateria de experimentos propostos para por à prova cada uma das hipóteses. Antes disso é necessário formular as hipóteses estatísticas e verificar as premissas para cada um dos conjuntos de dados. Baseado nas premissas, é possível escolher os testes estatísticos apropriados (paramétricos ou não), o nível de significância, o poder do teste e o tamanho da amostra. Baseado nesse planejamento é possível executar os experimentos, coletar e caracterizar os dados utilizando o arcabouço da estatística descritiva e, finalmente, submeter cada uma das hipótese formuladas a um teste estatístico visando refutá-las.

3.3 Premissas

Nos experimentos conduzidos por Santos (2015) o efeito da cooperação entre os agentes foi avaliado. O autor comparou os melhores valores de função objetivo que a arquitetura produziu em 30 execuções em que os agentes não colaboravam com outras 30 execuções em que a colaboração estava habilitada. Pacheco (2017) repetiu parte desses experimentos e realizou comparações para se certificar que as alterações no modelo de comunicação da arquitetura não alterariam o seu comportamento. Ambos os autores se limitaram a fazer comparações visuais utilizando o *boxplot*, e por isso não fizeram nenhum teste de hipótese sobre a amostra. Entretanto, por mera inspeção destes mesmos *boxplot*, é possível perceber que os valores de função objetivo não seguem uma distribuição normal. Portanto, não se pode assumir normalidade da amostra de valores ótimos obtidos pela arquitetura.

Pacheco (2017) avaliou também a escalabilidade da arquitetura, e verificou que utilizando 2 nós no *cluster*, é possível aumentar o número de agentes e observar um aumento proporcional da quantidade de soluções obtidas em um mesmo intervalo de tempo. Entretanto, essa escalabilidade não foi avaliada quando outros nós são adicionados à simulação. Espera-se que o aumento do número de nós possa intensificar a comunicação pela rede, gerando algum atraso nas mensagens. Como será avaliada a latência média, espera-se que a distribuição das médias das latências siga uma distribuição normal pelo teorema do limite central.

Ao executar uma bateria de experimentos no *HPC* do CEFET-MG, espera-se que uma execução sempre inicie em um ambiente limpo, sem influência de execuções passadas. Parte disso é garantido pelo próprio *script* de execução, que remove todos os arquivos temporários antes de iniciar uma nova execução, e também inicia o banco de dados em um estado inicial.

Dado que as execuções são realizadas em um ambiente isolado, gerenciadas por um escalonador de recursos (no caso do *cluster* do CEFET-MG, é utilizado o SLURM³), é

³<https://slurm.schedmd.com/>

possível presumir que outras tarefas não irão influenciar a execução do D-Optimas, pois não haverá competição por recursos computacionais, tais como memória e processador. Em outras palavras, dado que o *script* da simulação reserva uma certa quantidade de núcleos de processamento e memória, uma execução iniciará e terminará com os mesmos recursos, não sendo escalonada com outras tarefas de outros usuários, dividindo espaço somente com as tarefas gerenciais do escalonador e do sistema operacional.

Os experimentos serão executados em uma fila do *HPC* do CEFET-MG reservada à pesquisa do Laboratório de Sistemas Inteligentes do CEFET-MG. Os computadores dessa fila estão isolados da rede do restante do *cluster*, portanto, a última premissa que este trabalho assume é que as únicas mensagens que trafegarão na rede com alguma relevância serão as da arquitetura D-Optimas. Além disso, fez-se o melhor esforço para minimizar outros tipos de comunicação (por exemplo, as do sistema de arquivos, que utiliza o protocolo NFS). Assume-se que as mensagens de gerenciamento e monitoramento do escalonador trafegam na mesma rede em que trafegam as mensagens específicas da D-Optimas, não obstante, elas não influenciam de maneira crítica o resultado dos experimentos. Isso é possível de ser verificado utilizando o utilitário de monitoramento do *HPC* do CEFET-MG, o **Ganglia**⁴. Nele é possível verificar que o tráfego na rede quando o super-computador está ocioso é mínimo. Além disso, todo o tráfego é direcionado ao nó mestre do *HPC*, que não é diretamente utilizado durante as simulações da arquitetura D-Optimas.

Por fim, é possível presumir que os relógios das máquinas estejam sincronizados. Essa premissa está baseada no fato de que no *HPC* do CEFET-MG os nós estão configurados para sincronizar os relógios com o nó mestre utilizando o *NTP* (*Network Time Protocol*). Eventualmente, pode haver algum atraso entre os relógios de duas máquinas particulares, mas espera-se que esse efeito não seja significativo, uma vez que os resultados dos experimentos serão amostrados considerando as médias das latências.

3.4 Experimentos

Para colocar à prova as hipóteses propostas neste trabalho, três experimentos computacionais foram previstos. O primeiro visa avaliar a corretude da arquitetura, verificando se as alterações no comportamento das regiões e na comunicação entre os atores supervisores produzem um comportamento de convergência em torno dos valores conhecidos de função objetivo para cada problema. Em outras palavras, deseja-se verificar a hipótese nula de que a média populacional dos valores de função objetivo obtidos pela arquitetura é igual ao valor ótimo conhecido pela literatura, contra a hipótese alternativa de que os valores

⁴É possível acessá-lo na rede interna do CEFET-MG através do endereço <http://cluster.decom.cefetmg.br>

de função objetivo obtidos pela atual versão da arquitetura sejam diferentes dos valores conhecidos da literatura.

Para isso, será necessário utilizar um teste de hipótese não paramétrico para uma amostra, uma vez que não é possível assumir normalidade dos valores de função objetivo obtidos pela arquitetura. Neste ponto há uma dificuldade significativa, pois é sabido que o poder dos testes não paramétricos é menor, o que requer amostras maiores, e pela complexidade tecnológica da arquitetura D-Optimas, coletar amostras grandes é muito custoso, ainda que utilizando um *HPC*.

Além de avaliar o correto comportamento da arquitetura, é necessário avaliar a escalabilidade da arquitetura ao longo de um *cluster*, executando-a em vários nós ao mesmo tempo. Espera-se que ao aumentar a quantidade de nós na simulação, preservando o número de agentes e o limite de regiões, a latência não aumente de maneira significativa, bem como não prejudique a quantidade de soluções geradas da arquitetura e nem afete a sua qualidade. Dito de outra maneira, é necessário fazer duas análises de variância para um fator. Deseja-se avaliar a média da latência das trocas de mensagens e a quantidade de soluções produzidas por intervalo de tempo, para diferentes valores do fator número de nós em que a simulação executa.

Neste experimento o resultado esperado para o modelo proposto de comunicação entre os atores supervisores e os agentes é que não haja diferença estatisticamente significativa para os diferentes níveis do fator número de nós no *cluster*, *i.e.*, não há diferença entre executar a arquitetura com dois, três ou mais nós. Um resultado positivo neste quesito daria um indício da resiliência da arquitetura a diferentes cargas e do seu potencial de escalabilidade.

Por fim, dadas as novas meta-heurísticas que foram adicionadas na atual versão, procura-se verificar, com um número grande de agentes com diferentes estratégias de busca, se os resultados encontrados são melhores que os obtidos por Santos (2015). Em outras palavras, deseja-se verificar a hipótese nula de que a média dos resultados da arquitetura com várias heurísticas e diferentes configurações é melhor do que a média dos resultados com pouca diversidade de estratégias de busca.

Novamente, há um elemento expressivo de estocasticidade neste experimento que tem origem em diferentes fontes. Entre elas destaca-se o fato de cada agente iniciar com um gerador de números aleatórios diferentes, que é utilizado para a tomada de decisão dos algoritmos; as regiões se agruparem de maneira não-determinística; e finalmente o conjunto de soluções dos algoritmos populacionais não ter sido gerado aleatoriamente, mas escolhido de uma ou mais regiões particulares do espaço de busca. Esses elementos de aleatoriedade dificultam a identificação da função de densidade de probabilidade dos valores de função

objetivo obtidos pela arquitetura, e portanto impedem o uso de testes paramétricos. Esses fatores levam à necessidade de se coletar amostras grandes, por longos períodos de tempo, para avaliar a arquitetura com um valor de significância razoável (abaixo de 5%).

3.5 Considerações Finais

O presente capítulo se dedicou a expor a metodologia científica proposta para a realização deste trabalho. Foram apresentadas três hipóteses que fundamentam esse projeto de pesquisa. Destas três, a última hipótese é a mais relevante do ponto de vista científico. A arquitetura D-Optimas pode ser caracterizada como uma hiper-heurística, um sistema auto-organizável, dotado de um mecanismo de hibridização dinâmica, capaz de gerar combinações e encadeamentos de meta-heurísticas adaptadas às propriedades particulares de cada problema. Além disso, ela é configurável e extensível, permitindo a fácil adição de problemas combinatórios, de otimização irrestrita ou restrita. Além disso, há o fato do sistema ser construído sobre um arcabouço complexo de sistemas distribuídos. Todos esses fatores tornam a tarefa de estudá-lo, compreendê-lo e testar hipóteses sobre ele um trabalho relevante.

Fez-se aqui também um esforço em levantar as várias premissas que contornam a arquitetura. Um sistema distribuído sofre influência de diversos fatores, incluindo as possíveis falhas na rede, a sincronia entre os relógios dos nós, a carga em cada computador. Todos esses fatores tornam a atividade de replicar e depurar erros bastante complexa.

Deste modo, o desenvolvimento do trabalho foi feito de maneira criteriosa, testando cada nova funcionalidade, cada meta-heurística separadamente. Algumas destas validações estarão presentes no Capítulo 5 e Capítulo 6, ao se fazer a análise da estatística descritiva dos resultados do experimento. O capítulo seguinte se dedicará a apresentar os detalhes do desenvolvimento, principalmente no que diz respeito a nova topologia da arquitetura, adequada para funcionar como um sistema distribuído, e a nova dinâmica das regiões.

Capítulo 4

Desenvolvimento

Maluqueiras - é o que não dá certo. Mas só é maluqueiras depois que se sabe que não acertou.

Riobaldo, em "Grande Sertão: Veredas"

Algumas arquiteturas para resolução de problemas de otimização baseadas em agentes foram apresentadas no Capítulo 2, em especial as que possuem alguma característica de paralelismo, seja em um único computador, utilizando *multi-threading*, ou em um *cluster*. Dentre essas se destaca a arquitetura D-Optimas, objeto de estudo do presente trabalho.

Nesta arquitetura, os agentes encapsulam metaheurísticas e interagem entre si trocando soluções através de um ambiente virtual, buscando aprender nesse ambiente a encontrar soluções de boa qualidade para qualquer problema de otimização. O ambiente virtual, ou mundo virtual, é composto de soluções que são abrigadas em conjuntos disjuntos, chamados de regiões. O objetivo é que os agentes interajam com as regiões, e não com as soluções diretamente, "compreendendo", de maneira genérica, partes do espaço de busca que sejam mais ou menos interessantes. Da mesma forma, espera-se que as regiões possam apresentar uma dinâmica, podendo se fundir ou se particionar, a se adaptem à geometria do espaço de busca de qualquer problema de otimização.

A arquitetura D-Optimas foi construída baseada no modelo de atores para ser um sistema distribuído. Entretanto, alguns aspectos importantes que são inerentes à um sistema distribuído foram negligenciados na sua primeira versão, principalmente nas questões relacionadas ao balanceamento de carga, transparência de localidade e resiliência a falhas.

Em um sistema que cresce com o passar do tempo, com agentes produzindo soluções, regiões se fundindo, sendo criadas e se particionando, é importante que o sistema possa criar os agentes e regiões nos nós do *cluster* de maneira equilibrada, de modo a não onerar nenhum nó em particular, fazendo mal uso dos recursos computacionais. A transparência de localidade também deve ser uma característica de um sistema que se propõe a resolver

problemas de grande escala, sendo necessário executá-lo em um *cluster* computacional, onde é difícil conhecer previamente os endereços *IPs* das máquinas que executarão o experimento. A resiliência a falhas num sistema distribuído é importante pois falhas na rede são frequentes, um nó pode sair no meio da simulação, e perder as soluções que estavam nele não é aceitável, uma vez que a função objetivo pode ser complexa e custosa para se calcular. Outro fator importante é a necessidade de uma definição mais precisa da dinâmica das regiões, que não foi apresentada nos trabalhos anteriores.

Deste modo, neste capítulo a Seção 4.1 descreve de maneira mais detalhada os conceitos de sistemas distribuídos e os problemas encontrados na arquitetura, bem como apresenta uma nova topologia para os agentes e regiões, baseada na biblioteca *akka-cluster*. A Seção 4.2 apresenta os detalhes da dinâmica das regiões, como elas são criadas e quais as condições que permitem à elas fazer fissão e fusão. Por fim, a Seção 4.4 apresenta os algoritmos que foram adicionados à arquitetura, e como esse processo se deu no contexto das abstrações propostas por Souza (2014) para a generalização de metaheurísticas.

4.1 Da topologia do sistema utilizando *akka-cluster*

Como discutido na Seção 2.2, Pacheco (2017) propôs uma remodelagem da arquitetura BIMASCO, utilizando o modelo de atores e o *framework Akka*. Essa remodelagem deu origem à arquitetura D-Optimas, que funciona como um sistema distribuído, podendo ser executada em um *cluster* com dezenas de nós e milhares de agentes. Entretanto, o autor utilizou somente os recursos mais básicos do *framework*, a saber, o recurso de *akka remote*. Este recurso permite que atores que executam em um processo se comuniquem com atores em outro processo. Entretanto, só a mera comunicação entre processos não é suficiente para caracterizar um sistema como distribuído. Há três propriedades importantes para que um sistema seja considerado distribuído que estão ausentes na arquitetura D-Optimas: tolerância a falhas, balanceamento de carga e transparência de localidade. Os próximos parágrafos argumentam, baseados na literatura, a razão dessas três propriedades serem importantes e apresentam uma solução para esses três problemas, apoiada na biblioteca *akka*.

Segundo Tanenbaum e Steen (2007), um sistema distribuído pode ser definido da seguinte maneira:

"A distributed system is a collection of independent computers that appears to its users as a single coherent system" (TANENBAUM; STEEN, 2007, p. 2)

Dito isto, o autor ressalta que dois aspectos relevantes dessa definição são a característica autônoma e cooperativa dos componentes do sistema, fazendo com que o usuário, ao lidar

com o sistema, perceba-o como um sistema unificado. Este aspecto em especial não está presente na primeira versão do D-Optimas. Um exemplo disso é o caso em que um dos atores supervisores falhe. Neste caso é necessário reiniciar a simulação, o que deixa claro para o usuário que os componentes não colaboram de maneira efetiva.

Além da falta de resiliência a falhas do sistema, na versão proposta por Pacheco (2017) os agentes e regiões precisam saber o endereço físico uns dos outros para poderem se comunicar. É um ponto que também diverge da definição de Tanenbaum e Steen (2007), uma vez que sistemas distribuídos devem ter algum nível de transparência, sendo uma das mais básicas a transparência de localidade, *i.e.*, dois componentes de um sistema distribuído serem capazes de se comunicar sem saber exatamente o endereço físico de cada um (TANENBAUM; STEEN, 2007, p.4).

Por fim, um último ponto que é crítico para a escalabilidade de um sistema é o balanceamento de carga. Na primeira versão da arquitetura, os processos que executavam as regiões e agentes eram especializados, ou continham somente regiões, ou somente agentes. Essa decisão de projeto não contribuiu para o melhor uso dos recursos do *cluster*. Enquanto os agentes fazem uso intenso do processador (*CPU-bound*), para executar os algoritmos e encontrar boas soluções, as regiões fazem uso intensivo da rede (*I/O-bound*), trocando mensagens entre si para decidir entre fazer fissão ou fusão. Segregar as entidades da simulação pelo seu papel pode gerar uma sobrecarga no processador de um nó, enquanto o outro nó está ocioso.

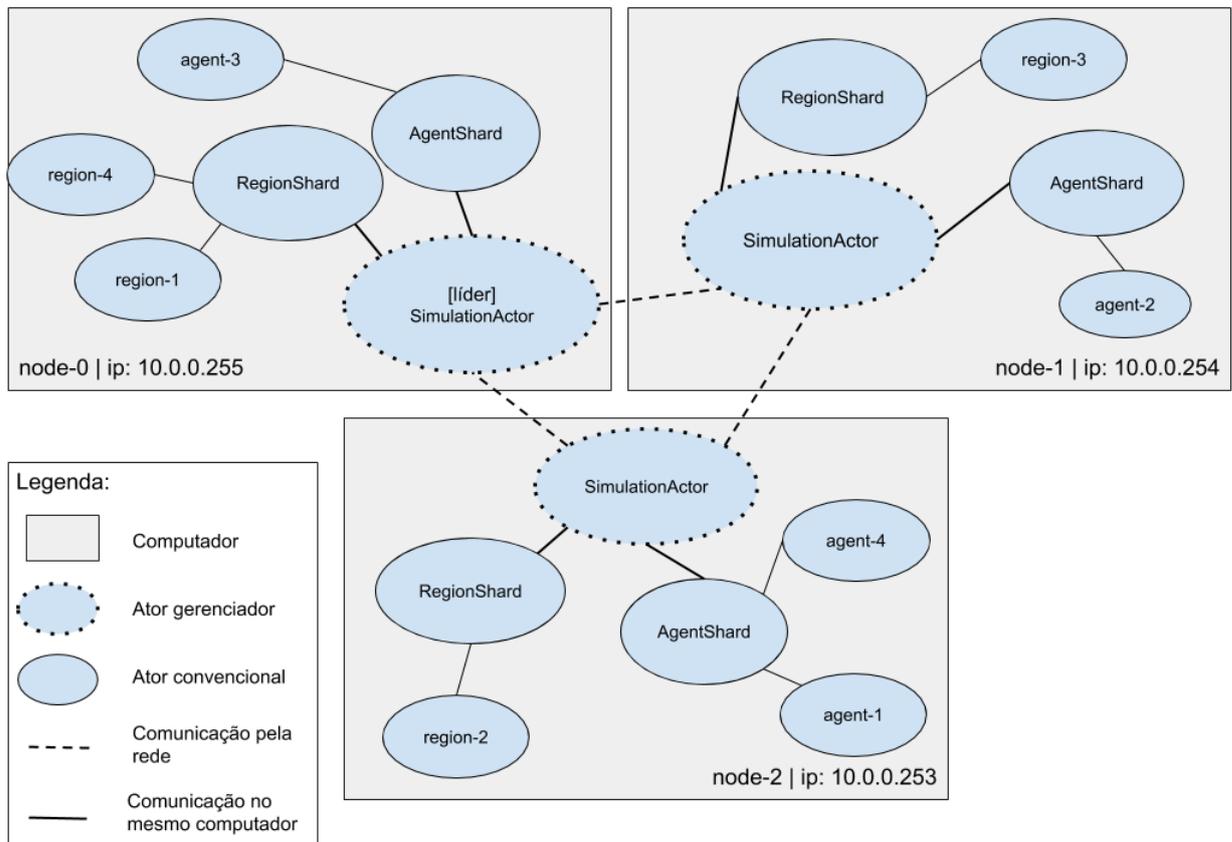
Levantadas essas três deficiências, a saber, a falta de resiliência a falhas, de transparência de localidade e do balanceamento de carga entre os nós da simulação, o que se propôs inicialmente neste trabalho foi uma correção da arquitetura para abrigar essas características importantes do ponto de vista de sistemas distribuídos. Para solucionar esses três problemas o *toolkit Akka* dispõe de uma funcionalidade importante, a biblioteca *akka-cluster*¹.

Esse pacote dispõe de ferramentas uteis para a criação de um *cluster* entre os atores, oferecendo algoritmos eficientes, sem ponto único de falha ou ponto único de sobrecarga. As principais ferramentas utilizadas neste trabalho é a descoberta de novos membros, detecção de falha, e eleição de líder, persistência, migração e transparência de localidade. A explicação detalhada da implementação de cada um desses conceitos pode ser encontrada na documentação da biblioteca, e por isso não serão abordados aqui. A seguir será abordado como esses mecanismos foram utilizados para construir a arquitetura e como eles facilitam a abordagem dos problemas apresentados.

A Figura 5 ilustra a nova organização proposta da arquitetura. Cada ator é representado por

¹<https://doc.akka.io/docs/akka/current/cluster-usage.html>

Figura 5 – Nova organização dos atores da arquitetura D-Optimas.



Fonte: O próprio autor

uma elipse, os responsáveis por gerenciar a simulação tem a borda pontilhada, e os agentes e regiões, por sua vez, têm as bordas contínuas. As linhas contínuas indicam que os atores se comunicam no mesmo nó, e as linhas tracejadas indicam que os atores se comunicam através da rede. A imagem exemplifica um *cluster* composto por 3 computadores.

A arquitetura é agora composta por um único ator supervisor por processo, chamado **SimulationActor**, responsável por gerenciar tanto as regiões quanto os agentes. Quando um nó começa a executar, este é o primeiro ator a ser criado, e há um único para cada processo em execução. O **SimulationActor** subscreve nos eventos do *cluster* de novo membro (**MemberUp**), saída de um membro (**MemberDown**) e de mudança de líder (**LeaderChanged**), de modo que ele recebe uma mensagem quando qualquer um desses eventos acontece. Quando um segundo nó começa a executar, ambos recebem uma mensagem de que há um novo membro na simulação. Neste ponto, a biblioteca *akka-cluster* elege um líder. Doravante, sempre que o **SimulationActor** for mencionado, este deve ser associado imediatamente com o líder da simulação, e caso seja necessário falar de um outro ator do mesmo tipo em um nó qualquer, essa distinção será feita.

O líder tem um papel importante ao longo da execução da simulação. Dentro da especificação da biblioteca *akka-cluster*, ele é o responsável por tarefas administrativas relacionadas ao *cluster*, como por exemplo permitir a entrada de novos membros ou remover um nó que está em estado inacessível. No caso da simulação do D-Optimas, o **SimulationActor** também é o responsável por iniciar e finalizar a simulação, criar e destruir regiões e manter os agentes atualizados sobre o estado do espaço de busca. Ele também mantém dados estatísticos das regiões, que serão detalhados na seção a seguir.

Ao longo do texto pode parecer que o líder tem um papel crítico na execução da simulação, e que isso pode caracterizar algum tipo de gargalo, ponto único de falha ou controle central. Entretanto, o líder exerce fundamentalmente um papel de coordenador, podendo inclusive ser mudado ao longo da simulação. Executar algumas das tarefas no líder é uma mera questão de escolha, por ser uma maneira mais simples de implementar um algoritmo num ambiente distribuído, sem a adoção um controle central. Isso tão pouco caracteriza um controle central, pois caso um líder fique inoperante, outro nó assume a liderança quase que imediatamente, sem causar prejuízo ou interrupção da simulação. Este comportamento já é um importante passo dado na direção de promover um sistema resiliente a falhas, desde que seja possível restaurar os agentes/regiões que estavam subordinados ao nó falho.

Como foi dito, para cada processo criado há um único ator do tipo **SimulationActor**, que supervisiona outros dois atores, o *agentShard* e o *regionShard*. Um ator do tipo **Shard**, disponibilizado na biblioteca *akka-sharding*, um sub-pacote do *akka-cluster*, é o responsável por oferecer transparência de localidade dentro de um *cluster*. Isso permite que os agentes e regiões troquem mensagens pelo seu endereço lógico, em geral, o nome do ator, em detrimento de seu endereço completo. Caso o *agent-1* queira enviar uma mensagem para a *region-4*, essa mensagem é enviada juntamente com o identificador do destinatário à *shard* mais próxima, que sabe como encaminhar a mensagem ao destinatário.

Além da transparência de localidade, a biblioteca *akka-sharding* oferece persistência aos atores, utilizando um banco de dados interno e distribuído, baseado em CRDTs (*Conflict-free Replicated Data Types*)(SHAPIRO et al., 2011). Isso permite que atores migrem de um *Shard* para outro, caso o primeiro nó esteja mais sobrecarregado que o segundo. A utilização do banco de dados distribuído também permite que, caso um nó falhe, os agentes e regiões sejam restaurados em outro nó, no último estado registrado no banco. Portanto, a arquitetura possui neste ponto algum nível de resiliência a falhas, e também o balanceamento de carga entre os nós.

4.2 Da dinâmica do espaço de busca e interações entre regiões e o *SimulationActor*

Uma vez construído o sistema, com os agentes e regiões comunicando entre si por meio dos *shards* e conectados ao líder da simulação com auxílio do *akka-cluster*, o próximo passo é entender o comportamento das regiões no sistema. São quatro os algoritmos associados à essa dinâmica: a criação de novas regiões, a inserção de uma solução em uma região, a fissão de uma região e a fusão de duas regiões.

O primeiro cenário que será abordado é quando um agente produz a primeira solução do sistema e quer enviá-la à uma região. Como o agente tem um conhecimento parcial do mundo, ele sempre envia essa solução para o ator gerenciador da simulação. Sendo as regiões conjuntos não-vazios de soluções, a simulação não possui nenhuma região até que a primeira solução seja criada. O *SimulationActor* é o responsável por criar esta nova região que abrigará aquela solução. Para criar uma região, o *SimulationActor* envia uma mensagem do tipo **CreateRegion** ao **RegionShard** subordinado a ele. Internamente, o *akka-cluster* decide em qual nó da simulação essa região vai ser criada, não sendo necessariamente no mesmo nó. Essa decisão será tomada com base no balanceamento das **Shards**. Uma nova região é criada para cada nova solução recebida até que o sistema atinja o número mínimo de regiões em toda a simulação, que é um parâmetro de configuração do sistema fornecido pelo usuário no arquivo de configuração da simulação. Quando atingido esse limite mínimo, o sistema passa a escolher aleatoriamente a região que receberá a próxima solução.

Esse procedimento aleatório se repetirá até que as regiões tenham pelo menos duas soluções. A partir desse momento, o **SimulationActor** passa a manter alguns dados relativos às regiões, que são: o valor médio, a variância e o coeficiente de variação das funções objetivo de todas as soluções, bem como o número de soluções para cada região. As fórmulas para o cálculo de cada um desses parâmetros estão descritas na Tabela 1. Ele também agrega os dados de cada região para calcular a estatística global do sistema, que compreende a média, desvio padrão e o coeficiente de variação dos valores de função objetivo de todas as soluções existentes, aqui denotados por \bar{x}_g , s_g e CV_g , respectivamente.

No momento em que cada região tem ao menos duas soluções e o CV de todas as regiões é diferente de zero, para cada nova solução recebida, o **SimulationActor** sorteia a região que receberá a próxima solução baseado no coeficiente de variação das regiões: regiões com maior CV tem menor probabilidade de receber soluções, enquanto regiões com menor CV tem chances maiores chances de serem escolhidas. Quando uma região recebe uma solução, ela envia uma mensagem do tipo **UpdateRegionSummary** para o líder, que atualiza a estatística da região, recalcula a estatística global e envia-a para todas as regiões, através

Tabela 1 – Fórmulas utilizadas para calcular os parâmetros de uma região r com n soluções x_i , e uma função objetivo é $f(\cdot)$

Parâmetro	Fórmula
Média (\bar{x})	$\sum_{i=1}^{i \leq n} f(x_i)$
Variância (s^2)	$\frac{1}{n-1} \sum_{i=1}^{i \leq n} (\bar{x} - f(x_i))^2$
Coefficiente de variação (CV)	$\frac{s}{\bar{x}}$

de uma mensagem **UpdateGlobalSummary** contendo os valores de \bar{x}_g , s_g e CV_g .

Sempre que o **SimulationActor** recebe uma nova solução ele também incrementa um contador, que é considerado o tempo discreto da simulação. Esse tempo é propagado para todos os elementos do **cluster**, incluindo os agentes e regiões, utilizando a estratégia de *pig backing* sobre a mensagem de **UpdateGlobalSummary**. Apesar de funcionar como um relógio lógico global (TANENBAUM; STEEN, 2007, p. 244), o tempo discreto não tem o objetivo de criar qualquer mecanismo de sincronização entre os membros da simulação. As mensagens que transmitem soluções são marcadas com esse tempo que é utilizado posteriormente para forçar uma ordenação parcial entre os eventos, facilitando a extração e análise de dados.

No que diz respeito à estratégia de escolha das regiões baseada no CV , há outras diferentes formas de se fazer essa escolha, *e.g.*, sortear aleatoriamente uma região para receber uma solução. Entretanto, o presente trabalho não pretende avaliar o mérito de uma estratégia, ou comparar esta com qualquer outra. O sorteio probabilístico foi escolhido por ser um algoritmo simples, mas não trivial. O algoritmo proposto não oferece nenhuma garantia sobre como a adição de uma nova solução a uma região particular afetará a distribuição dos valores de função objetivo, podendo fazer essa região tanto se tornar mais especializada, com o CV menor, menos especializada, aumentando o CV . A única premissa do procedimento adotado é que regiões muito especializadas tem mais chances de crescerem em quantidade de soluções.

As regiões, apesar de não escolherem quais soluções receberão, tem autonomia para se particionar ou se fundir, sendo assim entidades ativas do sistema. Como atores da plataforma

akka são entidades reativas que somente respondem a mensagens, que podem ter sido enviadas por um outro ator ou pelo próprio receptor, foi utilizado um artifício já proposto por Pacheco (2017). As regiões recebem uma mensagem chamada **InternalBehaviour**, que é enviada periodicamente pela própria região. Ao receber essa mensagem, a receptora deve escolher, com uma probabilidade de 50% entre se separar em três outras regiões ou se fundir com uma outra região.

Caso uma região r escolha fazer a fissão, ela deve se certificar de que duas condições sejam satisfeitas. A primeira delas é que a região possua um número mínimo de soluções, cujo valor é dado pelo usuário no início da simulação. Esse parâmetro existe para evitar que uma região se particione prematuramente. A segunda condição é que o coeficiente de variação dos valores de função objetivo da região r seja maior que o coeficiente de todas as soluções do sistema, *i.e.*, $CV_r > CV_g$.

Caso essas duas condições sejam satisfeitas a região r se particionará em outras 3 regiões, a saber, as soluções cujo valor de função objetivo estão abaixo de $\bar{x}_r - s_r$ (cauda inferior), as soluções que estão acima de $\bar{x}_r + s_r$ (cauda superior) e as soluções que estão entre $\bar{x}_r - s_r$ e $\bar{x}_r + s_r$, onde \bar{x}_r e s_r são a média e o desvio padrão dos valores de função objetivo das soluções que pertencem a região r , respectivamente.

Ao se particionar, a região envia ao *SimulationActor* uma mensagem do tipo *RegionSplit* contendo dois conjuntos de soluções, as da cauda superior e inferior da distribuição. As soluções do centro da distribuição ficam com a região r . Ao receber uma mensagem do tipo, o **SimulationActor** verifica se o número máximo de regiões permitidas no sistema não foi atingido e, em caso negativo, cria as novas regiões com as soluções recebidas. Caso o número máximo já tenha sido atingido, o sistema redistribui aleatoriamente as soluções entre as regiões já existentes. Esse parâmetro do número máximo de regiões também é fornecido no arquivo de configuração do sistema e é necessário para evitar o crescimento exacerbado do número de regiões, levando o sistema a consumir memória de maneira desordenada.

Caso a região r_1 decida por tentar fazer a fusão, ela enviará uma mensagem do tipo **MergeRequest** para todas as regiões contendo os seus parâmetros estatísticos. Ao receber uma mensagem desse tipo, uma outra região r_2 verifica se a união dos dois conjuntos de solução produzirá uma nova região r_3 cujo CV_{r_3} é menor do que o parâmetro CV_{r_2} . Caso essa condição seja satisfeita, a região r_2 responde à região r_1 com uma mensagem do tipo **MergeResponse**. Recebendo a primeira mensagem desse tipo, a região r_1 imediatamente envia uma mensagem do tipo **MergeResult** contendo a sua lista de soluções, e se destrói. Outras regiões que tenham respondido à solicitação de r_1 terão suas respostas ignoradas, não causando nenhum prejuízo ao sistema.

Quadro 1 – Mensagens trocadas na arquitetura D-Optimas

Remetente	Destinatário	Mensagem	Finalidade
SimulationActor	RegionActor e AgentActor	SetLeader	Informar a mudança do líder
		UpdateGlobalSummary	Atualizar o sumário global
	RegionActor	CreateRegion	Inicializar um nova região
	AgentActor	CreateAgent	Inicializar um novo agente
AgentActor	SimulationActor	AgentRegister	Registrar em um líder
		AgentRelease	Informar ao líder a saída da simulação
		SolutionResult	Inserir uma nova solução no sistema
	AgentActor	AgentInternalBehaviour	Atualizar o estado interno do agente
	RegionActor	SolutionRequest	Solicitar soluções à uma região
RegionActor	SimulationActor	RegionRegister	Registrar em um líder
		RegionRelease	Informar ao líder a saída da simulação
		UpdateRegionSummary	Atualizar os dados da região no líder
		RegionSplit	Informar a divisão de uma região ao líder
	RegionActor	RegionInternalBehaviour	Mudança do estado interno de uma região
		MergeResponse	Informar a uma região que aceitou a fusão
		MergeResult	Adicionar as soluções do remetente ao destinatário
	AgentActor	SolutionResponse	Fornecer soluções ao agente

O Quadro 1 apresenta um resumo de mensagens do sistema, bem como a finalidade de cada mensagem.

4.3 Da dinâmica interna do agente

Descrita a topologia geral do sistema, bem como a dinâmica interna das regiões, esta seção se dedica a escrutinar o funcionamento interno do agente. Serão aqui apresentados os seus componentes internos, as mensagens recebidas e enviadas pelo agente, o algoritmo de seleção de ação do agente e o funcionamento do seu componente de memória.

Três componentes integram o funcionamento do agente, entre eles: o mecanismo de seleção de ação, o mecanismo de memória e o algoritmo de otimização. O mecanismo de seleção de ação é acionado periodicamente pelo recebimento da mensagem **AgentInternalBehaviour**. O envio dessa mensagem é agendado no momento da criação do agente, pela mensagem do tipo **CreateAgent**, enviada pelo **SimulationActor**. A mensagem que dispara o comportamento é enviada a cada intervalo de 5 segundos.

Quando acionado, o mecanismo de seleção de ação é responsável por escolher como solicitar soluções para que o agente possa otimizar. Para enviar uma requisição por novas soluções, o agente pode ou utilizar a memória para escolher uma região, ou enviar uma mensagem de *broadcast* a todas as regiões. Essa escolha é ajustada pela probabilidade de utilização da memória, que é fornecida como parâmetro pelo usuário no início da simulação. Ao receber uma mensagem do tipo **SolutionResponse** contendo as soluções enviadas pela região remetente, o agente executa de maneira síncrona a sua metaheurística, oferecendo o conjunto de soluções como entrada. Após a execução da metaheurística, o agente atualiza a memória e envia a melhor solução encontrada nesta iteração para o **SimulationActor** utilizando a mensagem do tipo **SolutionResult**.

O sistema de memória do agente é baseado na técnica de aprendizagem por reforço *Q-Learning*. Essa é uma técnica de aprendizagem por reforço, baseada em uma tabela de mapeamento entre ações e valores de qualidade para cada uma das ações (NORVIG, 2018). A medida em que o agente interage com o ambiente a sua volta, e recebe recompensas pelas suas ações, essa tabela é atualizada. A regra de atualização do *Q-learning* está representada na Equação (2). A sequência de ações desempenhadas pelo agente ao longo do tempo, *i.e.*, o seu comportamento, é frequentemente chamado na literatura de *policy*. Dito isto, o aprendizado por qualidade é uma técnica do tipo *off-policy* pois ela explora os possíveis comportamentos do agente enquanto aprende o comportamento ótimo (SUTTON; BARTO, 2018).

$$Q_a(t_{i+1}) = Q_a(t_i) + \alpha * (R + \gamma * (\max(Q(t_i)) - Q_a(t_i))) \quad (2)$$

A regra de atualização de qualidade Q para cada ação a da tabela está descrita na Equação (2), conforme descrita por NORVIG (2018). O valor de qualidade da ação Q_a é

definido segundo seu valor no tempo anterior acrescido pela diferença entre o valor máximo da recompensa futura esperada dentre todos os itens na tabela $\max(Q(t_i))$ e o valor atual $Q_a(t_i)$. Esse valor é descontado e ajustado por uma taxa de aprendizado γ . Essa taxa de aprendizado é utilizada para balancear a recompensa imediata e futura. Finalmente, R é o termo utilizado para a recompensa após completar uma ação do algoritmo.

No contexto da arquitetura D-Optimas, a única ação possível ao agente é solicitar soluções a uma determinada região. Deste modo, a tabela descrita na Equação (2) pode ser definida como $Q_r(t)$, que representa o valor da qualidade de solicitar soluções à região r no instante de tempo t . O agente a recebe uma recompensa R sempre que consegue encontrar no tempo t um valor de função objetivo f_t^a relação à última solução por ele produzida, aqui denominado f_{best}^a . Caso o agente não consiga melhorar o último resultado encontrado essa recompensa deve ser negativa. Em ambos os casos, o valor de R está diretamente correlacionado com a diferença $\Delta f = f_{best}^a - f_t^a$.

É fácil perceber que os valores de Δf dependem do conjunto imagem da função objetivo. Atribuí-lo diretamente à tabela Q_a causaria um acoplamento entre uma característica do problema de otimização e a arquitetura, e isso não é desejável. Deste modo, os valores de Δf são normalizados entre $[-1, 1]$. A função sigmoide foi escolhida para normalizar os valores de recompensa, e a equação final que define o valor da recompensa R está descrita na Equação (3).

$$Reward(\Delta f) = 1 - \frac{2}{1 + e^{\Delta f}} \quad (3)$$

Quando o agente decide utilizar a memória para escolher uma região para solicitar soluções, ele executa uma roleta sobre os valores da tabela Q_a . Esta estratégia foi adotada para dar possibilidade ao agente explorar regiões que a princípio não levaram a soluções de melhor qualidade. É importante ressaltar que os algoritmos de otimização implementados na arquitetura D-Optimas são estocásticos, e a sua convergência para um valor ótimo não é garantida. Mesmo as regiões mudam ao longo do tempo, e uma região que antes levou a soluções de pior qualidade, pode no futuro levar o agente a encontrar melhores soluções.

Como já descrito na seção anterior, o espaço de busca está em constante mudança, dada a dinâmica de fusão e fissão das regiões. A cada nova solução criada por um agente, potencialmente uma nova região pode surgir, ou duas regiões podem se unir, alterando assim a configuração do sistema. Entretanto, o mecanismo *Q-learning* tal qual descrito na literatura é estático. Uma memória que tivesse todas as r regiões possíveis, uma vez que o número de regiões é limitado por um parâmetro da simulação, frequentemente poderia levar o agente a solicitar soluções de uma região que não existe. Outrossim, o número de regiões escolhido pode ser arbitrariamente grande, o que faria a memória Q_a do agente ser

grande.

Sendo assim, optou-se por limitar o tamanho da memória do agente à um tamanho M , que é dado como parâmetro no início da simulação. O agente também foi munido de um algoritmo para atualizar as regiões presentes na memória. Este algoritmo é executado sempre que um agente recebe uma mensagem do tipo **UpdateGlobalSummary**. Como já descrito, essa mensagem é enviada pelo **SimulationActor** a todos os atores da simulação sempre que alguma mudança acontece no sistema. O Algoritmo 1 exhibe o procedimento de atualização da memória dos agentes.

Algoritmo 1 Método de atualização da tabela de qualidade utilizada pela memória *Q-learning*

- 1: Recebe uma lista R_w das regiões presentes no espaço de busca
 - 2: $R_{stay} \leftarrow R_w \cap Q_a$
 - 3: $R_{remove} \leftarrow Q_a - R_{stay}$
 - 4: $R_{add} \leftarrow Q_a - R_w$
 - 5: $Q_a \leftarrow Q_a - R_{remove}$
 - 6: $Q_a \leftarrow Q_a \cup \{ (r, Reward(0)) \mid \forall r \in R_{add} \}$
 - 7: Ordene os pares $(id, R) \in Q_a$ decrescente em R
 - 8: Mantenha as M primeiras entradas da tabela Q_a
-

O algoritmo adota uma estratégia elitista na escolha das regiões que ficarão na memória. Ele recebe o conjunto de regiões R_w que existe atualmente no espaço de busca. O primeiro passo do algoritmo é calcular as regiões que ficam na memória, as que serão removidas e as que serão adicionadas (R_{stay} , R_{remove} e R_{add} , respectivamente). O passo seguinte é remover da memória do agente Q_a as regiões que não existem mais no mundo R_{remove} (linha 5), e adicionar as regiões que ainda não estão na memória (linha 6). As novas regiões são adicionadas com o valor de qualidade correspondente a uma recompensa nula. Por fim, a tabela Q_a é ordenada de maneira decrescente e as regiões com menor qualidade que excedem o tamanho M da tabela são removidas.

4.4 Adição de novos algoritmos populacionais

Um princípio fundamental na construção da arquitetura D-Optimas é de que a diversidade de estratégias de busca pode, não só levar a arquitetura a encontrar soluções de maior qualidade, quanto melhorar o desempenho geral em diferentes tipos de problema. A cada nova versão da arquitetura, alguns novos algoritmos foram adicionados, em especial na versão de Souza (2014), que adicionou vários algoritmos construtores e de busca local. Entretanto, pouco esforço foi empreendido na adição de algoritmos evolutivos ou populacionais, técnica consolidada para solução de problemas complexos, dos quais a única

premissa necessária é a de localidade fraca (GASPAR-CUNHA; TAKAHASHI; ANTUNES, 2012).

Neste sentido, fez-se um esforço no presente trabalho de trazer ao menos mais duas técnicas populacionais/evolutivas à arquitetura, de modo que experimentos pudessem ser conduzidos para avaliar o efeito da diversidade de estratégias de busca, algo que não foi testado em trabalhos passados. Os algoritmos inseridos foram o *Differential Evolution* (**DE**) (STORN; PRICE, 1997) e o *Particle Swarm Optimization* (**PSO**) (Kennedy; Eberhart, 1995; Shi; Eberhart, 1998). O algoritmo DE foi escolhido principalmente pela sua simplicidade, uma vez que possui poucos parâmetros de configuração. Por sua vez, o PSO foi escolhido por ser um dos algoritmos de otimização mais citados na literatura, tendo sido aplicado com sucesso em diversos problemas de otimização contínua (BONYADI; MICHALEWICZ, 2017).

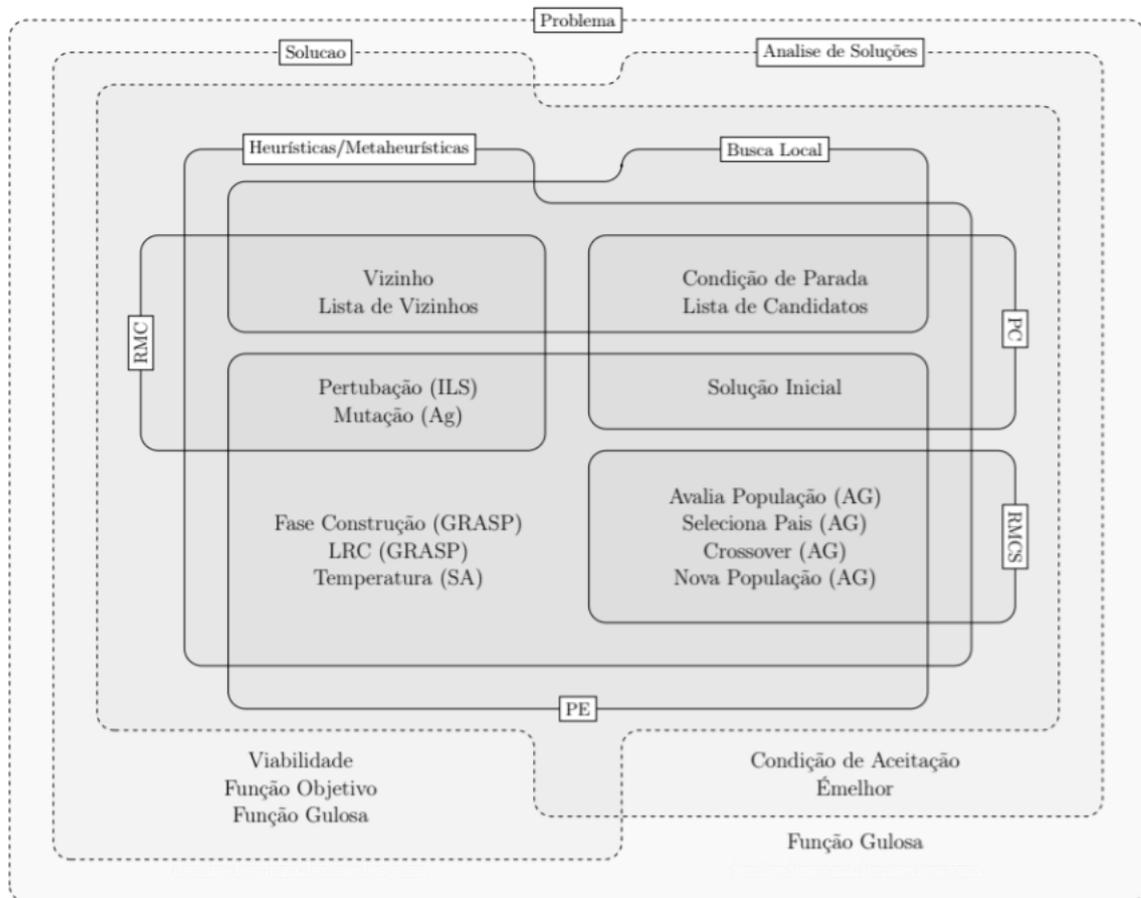
Ambos os algoritmos foram implementados somente para o problema de otimização irrestrita de função real, para o qual esses algoritmos foram inicialmente propostos. Entretanto, ambos foram implementados usando o arcabouço de generalização proposto por Souza (2014), o que torna fácil a extensão deles para problemas de natureza combinatória, bastando implementar os modificadores adequados. Os parágrafos a seguir apresentam os pseudocódigos das estratégias de busca e como eles foram implementados e adaptados para a arquitetura D-Optimas, utilizando os mecanismos de generalização.

Como já abordado no capítulo anterior, Souza (2014) criou um conjunto de classes abstratas para implementar algoritmos de otimização que sejam independentes dos dados do problema, bem como da representação das soluções. A Figura 6 apresenta essas estruturas abstratas e suas relações com os algoritmos já implementados na arquitetura. Um exemplo é o *Genetic Algorithm* (**GA**), que utiliza a estrutura de Modificador de Conjunto de Soluções (RMCS) para encapsular a avaliação da população, a seleção dos pais, o cruzamento e a seleção da nova população.

Portanto, para incluir um novo algoritmo utilizando essas ferramentas foi necessário estudá-las e identificar em cada algoritmo quais os pontos em comum que poderiam depender de um problema específico e encapsulá-los na estrutura apropriada. Deste modo, serão apresentados à seguir os pseudocódigos dos dois algoritmos selecionados, do **DE** e do **PSO**, a ideia central das metaheurísticas e quais trechos foram encapsulados em estruturas abstratas.

A primeira metaheurística abordada será o **DE**, cujo pseudocódigo é apresentado no Algoritmo 2. Essencialmente o algoritmo aplica uma regra simples de seleção e mutação em cada indivíduo da população, sucessivamente, até que uma condição de parada seja atingida. Na implementação original, uma população inicial é gerada aleatoriamente. Esse passo foi

Figura 6 – Diagrama das estruturas abstratas proposta por Souza (2014) e como elas foram utilizadas na implementações dos algoritmos já existentes na arquitetura. As estruturas Problema, Solução e Análise de Solução são comuns a todas implementações de metaheurística. As siglas RMS e RMCS significam, respectivamente, as estruturas de Realiza Modificação de Solução e Realiza Modificação de Conjunto de Solução. Elas são as principais estruturas utilizadas no presente trabalho para implementação dos algoritmos escolhidos.



substituído, evidentemente, pelo comportamento padrão dos agentes na arquitetura que é solicitar um conjunto de soluções às regiões (linha 2). A condição de parada (linha 3) segue o padrão das outras implementações na arquitetura D-Optimas, sendo informada pelo usuário no arquivo de configuração da simulação. Há atualmente algumas condições de parada implementadas na arquitetura que estendem a classe abstrata **StopCondition**. As implementações possíveis são a parada por número máximo de iterações, número máximo de iterações sem melhora, valor da melhor avaliação de função objetivo, tempo máximo de execução ou uma combinação qualquer das condições anteriores.

A classe **DE** é composta de dois modificadores, o primeiro responsável por sortear os indivíduos para o cruzamento, o que corresponde a linha 5 do pseudocódigo, e o segundo corresponde a regra de mutação. Como esse modificador recebe uma lista de soluções, que é a população atual, e retorna uma lista contendo três soluções, este modificador é do tipo

ModifiesSolutionCollection. O componente recebeu o nome de **DERandSelection**.

Algoritmo 2 Evolução Diferencial

```
1:  $t \leftarrow 1$ 
2: Recebe uma população inicial  $X_t$  de tamanho  $n$  e dimensão  $D$ 
3: while Condição de parada não for atingida do
4:   for  $i \leftarrow 1$  até  $n$  do
5:     Sorteie  $s_1, s_2, s_3 \in 1, \dots, n$ 
6:     Sorteie  $i_{rand} \in 1, \dots, n$ 
7:     for  $j \leftarrow 1$  até  $D$  do
8:       if  $rand(0, 1) \leq CR \vee i = i_{rand}$  then
9:          $u_{t,i,j} = x_{t,i,s_3} + F * (x_{t,i,s_1} - x_{t,i,s_2})$ 
10:      else
11:         $u_{t,i,j} = x_{t,i,j}$ 
12:      end if
13:    end for
14:  end for
15:  for  $i \leftarrow 1$  até  $n$  do
16:    if  $f(u_{t,i}) < f(x_{t,i})$  then
17:       $x_{t+1,i} \leftarrow u_{t,i}$ 
18:    else
19:       $x_{t+1,i} \leftarrow x_{t,i}$ 
20:    end if
21:  end for
22:   $t \leftarrow t + 1$ 
23: end while
```

As linhas 7 à 14 do **DE** foram encapsuladas em o segundo modificador de conjunto de soluções, que recebe uma lista contendo quatro soluções e aplica a regra de mutação descrita no algoritmo. Esse modificador foi implementado na classe **DEMutationMSC**.

Há algumas variações para as regras de seleção e mutação do algoritmo **DE**, que podem ser expressas de uma maneira sintética, de acordo com Gaspar-Cunha, Takahashi e Antunes (2012). A variação implementada foi **DE/rand/1/bin**. Entretanto, outras variações podem ser facilmente implementadas sem grandes modificações da classe **DE**, simplesmente provendo novos modificadores.

A implementação do **DE** apresenta muito menos desafios do que o algoritmo **PSO**, cujo pseudocódigo é apresentado no Algoritmo 3. O primeiro problema é a manutenção de duas populações, um vetor de posições, que são as soluções do problema de otimização, e uma segunda população, que são os vetores de velocidades associados às posições. O comportamento básico do algoritmo é atualizar, a cada iteração, o vetor de velocidade de cada partícula e logo em seguida atualizar a posição até que uma condição de parada seja atingida. Novamente, foi utilizada a estrutura abstrata de **StopCondition**, cuja

implementação concreta é informada no arquivo de configuração da simulação. Para isso, o algoritmo além de receber as soluções iniciais das regiões (linha 1), gera um conjunto de soluções do mesmo tamanho que o primeiro conjunto. Esse novo conjunto é inicializado aleatoriamente pelo método *generateInitialSolution()*, definido na classe abstrata **Solution**. Novamente, foi utilizada a condição de parada genérica **StopCondition**, cujo tipo concreto é informado no arquivo de configuração da simulação.

Algoritmo 3 Otimização por Enxame de Partículas

```

1: Recebe uma população inicial  $X$  de tamanho  $n$  e dimensão  $D$ 
2: Gere um conjunto de soluções aleatórias  $V$  de tamanho  $n$  e dimensão  $D$ 
3: while Condição de parada não for atingida do
4:   for  $i \leftarrow 1$  até  $n$  do
5:     if  $f(x_i) < f(pb_i)$  then
6:        $pb_i \leftarrow x_i$ 
7:     end if
8:     if  $f(x_i) < f(gb)$  then
9:        $gb \leftarrow x_i$ 
10:    end if
11:  end for
12:  for  $i \leftarrow 1$  até  $n$  do
13:    for  $d \leftarrow 1$  até  $D$  do
14:       $v_{i,d} \leftarrow v_{i,d} + C_1 * rand(0, 1) * (pb_{i,d} - x_{i,d}) + C_2 * rand(0, 1) * (gb_d - x_{i,d})$ 
15:       $x_{i,d} \leftarrow x_{i,d} + v_{i,d}$ 
16:    end for
17:  end for
18: end while

```

O algoritmo mantém uma terceira lista de soluções que armazena a melhor posição de cada indivíduo do enxame ao longo da execução do algoritmo, que corresponde no pseudocódigo à variável pb_i . Além desta memória, ele também mantém melhor partícula ao longo da execução, que corresponde à variável gb . A cada iteração essas variáveis são atualizadas, e para fazer a comparação, utilizamos a classe abstrata **SolutionAnalyzer**, que fornece métodos para efetuar a comparação entre duas soluções. Essa classe deve ser estendida e o seu tipo concreto também é informado no arquivo de configuração.

Os passos de alteração dos vetores velocidade e posição foram implementados utilizando dois modificadores de coleção diferentes. Para modificar as velocidades, é necessário passar um vetor de tamanho $2 * n + 1$ contendo nas n primeiras posições os vetores de velocidade, nas n posições seguintes o vetor pb e na última posição da lista a melhor solução até a t -ésima iteração gb . Esse modificador foi nomeado **PSOVelocityUpdateMSC** e corresponde a linha 14 do Algoritmo 3. Por fim, o modificador **PSOPositionUpdateMSC** recebe uma lista de tamanho $2n$ contendo nas n primeiras posições o vetor X e nas n posições seguintes o vetor V , que atualiza a lista de posições, encapsulando a linha 15 do pseudocódigo.

4.5 Considerações finais

O presente capítulo se dedicou a apresentar as alterações feitas na arquitetura D-Optimas, proposta inicialmente por Pacheco (2017). Essas alterações tiveram o principal objetivo de melhorar aspectos teóricos e evoluir a arquitetura, por exemplo, conferindo resiliência à simulação, do ponto de vista de sistemas distribuídos, e dando uma definição precisa para o comportamento das regiões. Foram adicionadas também mais estratégias de busca, para cumprir o propósito da D-Optimas, como um sistema multi-agente, no qual a diversidade de estratégias é um fator importante.

O capítulo seguinte apresentará os experimentos realizados para avaliar as mudanças que foram feitas até o momento, principalmente no que diz respeito ao desempenho computacional e a escalabilidade da arquitetura. Em seguida, o Capítulo 6 apresenta a avaliação do efeito da diversidade na qualidade das soluções produzidas pela arquitetura.

Capítulo 5

Avaliação da escalabilidade

Vou longe. Se o senhor já viu disso, sabe: se não sabe, como vai saber? São coisas que não cabem em fazer idéia.

Riobaldo, em "Grande Sertão: Veredas"

Apresentadas as correções e aprimoramentos feitos na arquitetura D-Optimas, é necessário avaliar os impactos no seu funcionamento e comportamento. Neste capítulo será apresentada uma bateria de experimentos, com o objetivo de avaliar a escalabilidade da nova implementação baseada em *akka-cluster*.

A Seção 5.1 descreve o experimento para avaliar o impacto do aumento da quantidade de nós do *cluster* na execução da simulação, principalmente na latência entre as mensagens trocadas e a taxa de produção de soluções dos agentes. A Seção 5.2 apresenta os resultados obtidos e a análise estatística para cada tipo de mensagem analisada. Espera-se que, mantendo o número de agentes e o limite de regiões, mas aumentando o número de nós, a carga seja melhor distribuída entre os nós, e que o tempo médio entre um envio de uma mensagem por um ator e o seu respectivo processamento em outro ator diminua. Por fim, a Seção 5.3 apresenta uma síntese dos resultados.

5.1 Avaliação de desempenho da arquitetura

Esse experimento avaliou a média da latência das trocas de mensagens entre os agentes, regiões e atores supervisores da simulação. O número de nós do *cluster* em que a arquitetura executou foi o único parâmetro que sofreu variação.

Foram executadas 4 instâncias, nomeadas S3, S4, S5 e S6, utilizando de 3 a 6 nós. Para cada instância do experimento foram obtidas 14 réplicas da mesma configuração, que é exibida no Tabela 2. Cada execução contou com 18 agentes, sendo 6 agentes construtores, 6 populacionais e 6 agentes de busca local. Os agentes construtores tiveram o tempo de vida

Tabela 2 – Parâmetros das meta-heurísticas

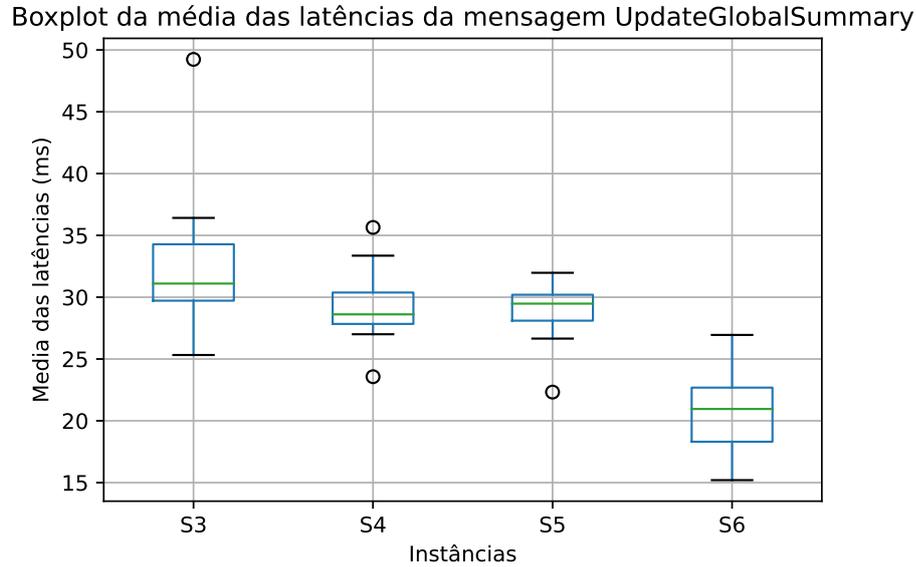
Nº de agentes	Metaheurística	Parâmetro	Valor
6	GRASP	iterações	100
		iterações busca local	100
		alpha	0.5
3	ILS	iterações	500
		nível de distúrbio	10
3	ILS	iterações	500
		nível de distúrbio	7
3	GA	iterações	300
		tamanho da população	20
		taxa de mutação	0.2
		taxa de cruzamento	0.7
3	GA	iterações	500
		tamanho da população	30
		taxa de mutação	0.1
		taxa de cruzamento	0.8

limitado à 200 unidades pois as soluções por eles geradas servem somente como soluções iniciais. Para este experimento os mecanismos de aprendizagem e memória dos agentes foram desabilitados, uma vez que a mudança na organização do sistema impacta fortemente no funcionamento desses componentes. Além destas configurações, ficou estabelecido o número limite de 100 regiões no sistema, o número mínimo de 30 soluções para uma região se particionar, e o tempo limite de 1000 unidades para que a simulação fosse interrompida.

O ambiente computacional utilizado foi o *cluster* do Laboratório de Sistemas Inteligentes (LSI) do CEFET, composto de seis máquinas com processador Intel i7, 32GB de RAM e 2TB de HD cada. O sistema operacional utilizado foi o CentOS 6, e o SLURM como escalonador de tarefas. A versão do JDK (*Java Development Kit*) utilizada foi a 12.0.1.

Foram escolhidos quatro tipos de mensagem para serem analisadas. A primeira escolhida foi **UpdateGlobalSummary** que é frequentemente enviada pelo líder da simulação a todas as regiões e agentes sempre que alguma mudança acontece no sistema. A segunda mensagem, do tipo **UpdateRegionSummary**, é enviada por uma região sempre que recebe uma nova solução, ou uma fissão ou fusão acontece. A terceira mensagem analisada, do tipo **RegionSplit**, é enviada pelas regiões ao líder da simulação sempre que uma região se particiona, e por fim, a quarta mensagem mensagem, do tipo **SolutionResponse**, é enviada aos agentes pelas regiões em resposta a uma solicitação de soluções. Neste subconjunto de mensagens estão presentes as principais e mais frequentes interações entre os agentes, regiões e o líder. Os dados extraídos foram tratados, removendo eventuais anomalias (latências negativas, por exemplo) e a média das latências para cada execução,

Figura 7 – *Boxplot* das médias das latências da mensagem **UpdateGlobalSummary** para 14 execuções da arquitetura D-Optimas com duração de 1000 unidades de tempo discreto em 3, 4, 5 e 6 nós do *cluster*



para cada tipo de mensagem, e foram computadas a média das latências para cada execução e para cada tipo de mensagem.

Antes de apresentar e discutir os resultados, é necessário fazer uma observação acerca do tratamento dos dados. Entre as execuções, cada mensagem pode ter sido trocada um número diferente de vezes e algumas delas podem não ter sido recebidas. Isso faz com que a média das latências da mensagem do tipo **UpdateGlobalSummary** em uma execução tenha sido calculada com um número diferente de observações em cada uma das 14 repetições para cada um dos 4 fatores. Essa particularidade dos dados, somada à estocasticidade inerente à arquitetura, pode levar os dados a não se comportarem bem, ainda que essa amostragem esteja congruente com a definição do teorema do limite central. Dito isso, as premissas da ANOVA são verificadas para cada tipo de mensagem, e quando atendidas, o teste paramétrico é escolhido. Todos os testes estatísticos foram realizados com um nível de confiança de 95%.

5.2 Resultados do experimento

A Figura 7 exibe o *boxplot* das médias das latências das mensagens do tipo **UpdateGlobalSummary**. Visualmente é possível perceber que os dados estão bem comportados e há uma tendência de queda da latência a medida que o número de nós cresce.

Para esse subconjunto dos dados, dois dos três pressupostos da ANOVA foram atendidos. O teste de Levene acusou um p-valor de 0.31538, falhando em rejeitar a hipótese nula

de que a variância dos resíduos é homogênea. Os resíduos são independentes mas não seguem uma distribuição normal, pelo teste de Shapiro-Wilk (p-valor de 0.000161). Como a ANOVA é robusta a pequenas variações na normalidade, este teste foi escolhido. O p-valor obtido foi de 0, indicando que há diferença em pelo menos uma das médias.

A Tabela 3 exibe os resultados do teste de Tukey-HSD, para um nível de significância de 5%. O teste conclui que não há diferença entre as instâncias S3 e S4, e S4 e S5, onde há variação de apenas um nó no *cluster*.

Tabela 3 – Comparações par-a-par (teste de Tukey-HSD) para as médias das latências de mensagens do tipo **UpdateGlobalSummary**

Instâncias		p-valor	Rejeita H_0 ?
S3	S4	0.0938	Não
S3	S5	0.0451	Sim
S3	S6	0.001	Sim
S4	S5	0.9	Não
S4	S6	0.001	Sim
S5	S6	0.001	Sim

A segunda mensagem analisada foi a **UpdateRegionSummary**. A Figura 8 exibe a distribuição das médias das latências para as quatro instâncias do experimento. Aparentemente há uma tendência de diminuição da variância a medida que a arquitetura escala. Os dados não atendem aos pressupostos de normalidade (com p-valor de 0.000121) e homocedasticidade (p-valor de 0.004774).

Por isso, foi escolhido um teste não paramétrico de Kruskal-Wallis, para verificar se há diferença entre as instâncias. O teste produziu um p-valor de 0.002919, que leva a rejeição da hipótese nula, indicando que há diferença em pelo menos uma das médias.

Para verificar em qual dos pares está a diferença, foi escolhido um teste *post-hoc* não paramétrico de Conover-Iman. Os p-valores obtidos pelo teste são apresentados na tabela Tabela 4. É possível observar que há diferença sempre que se adiciona pelo menos dois nós na simulação.

A terceira mensagem analisada foi a **RegionSplit**. O *boxplot* das médias das latências para as quatro instâncias do experimento é apresentado na Figura 9. Novamente é possível observar uma tendência de queda da latência a medida que novos nós são adicionados à simulação. Os pressupostos de normalidade e homocedasticidade não são atendidos, com p-valores de 0.000005 e 0.002790, respectivamente.

O teste de Kruskal-Wallis produziu um p-valor de 0.002748, o que indica uma diferença em pelo menos uma das médias. A Tabela 5 apresenta os resultados do teste de Conover-Iman. O teste novamente indica que não é possível observar diferença na latência entre

Figura 8 – *Boxplot* das médias das latências da mensagem **UpdateRegionSummary** para 14 execuções da arquitetura D-Optimas com duração de 1000 unidades de tempo discreto em 3, 4, 5 e 6 nós do *cluster*

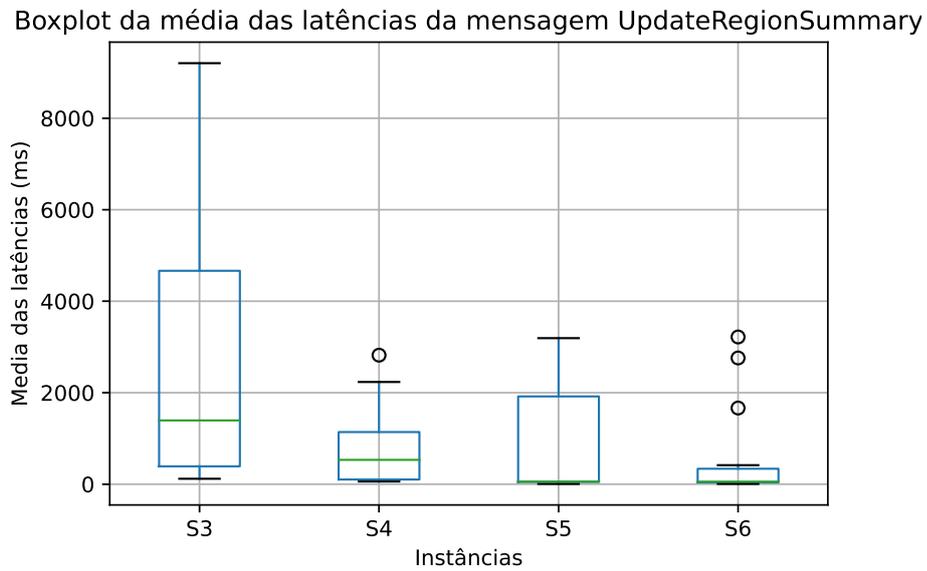


Figura 9 – *Boxplot* das médias das latências da mensagem **RegionSplit** para 14 execuções da arquitetura D-Optimas com duração de 1000 unidades de tempo discreto em 3, 4, 5 e 6 nós do *cluster*

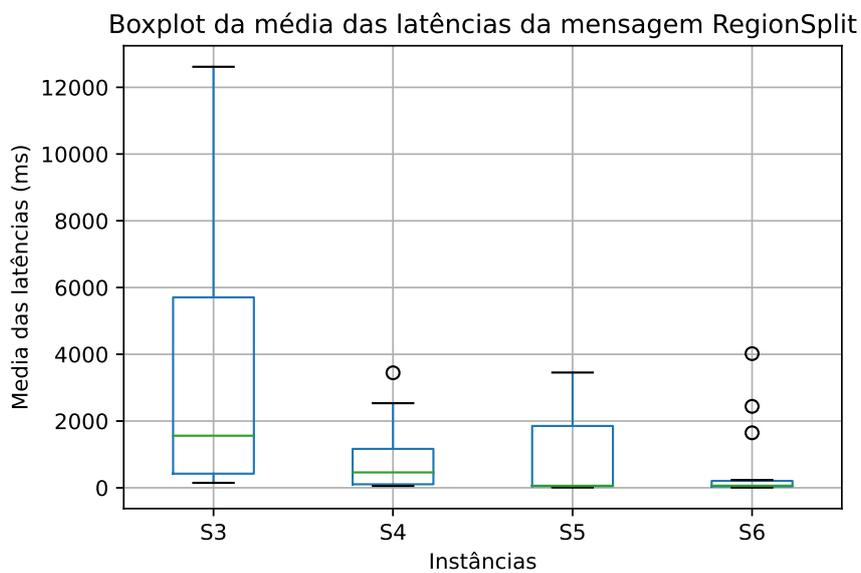


Tabela 4 – Comparações par-a-par (Teste de Conover) para a média das latências de mensagens do tipo **UpdateRegionSummary**

Instâncias		p-valor	Rejeita H_0 ?
S3	S4	0.06021	Não
S3	S5	0.00378	Sim
S3	S6	0.00017	Sim
S4	S5	0.27176	Não
S4	S6	0.03905	Sim
S5	S6	0.31895	Não

duas amostras com a variação de apenas um nó. Adicionando 2 nós, a diferença entre as médias da latência é significativa.

Tabela 5 – Comparações par-a-par (Teste de Conover) para a média das latências de mensagens do tipo **RegionSplit**

Instâncias		p-valor	Rejeita H_0 ?
S3	S4	0.06021	Não
S3	S5	0.00378	Sim
S3	S6	0.00017	Sim
S4	S5	0.27176	Não
S4	S6	0.03905	Sim
S5	S6	0.31895	Não

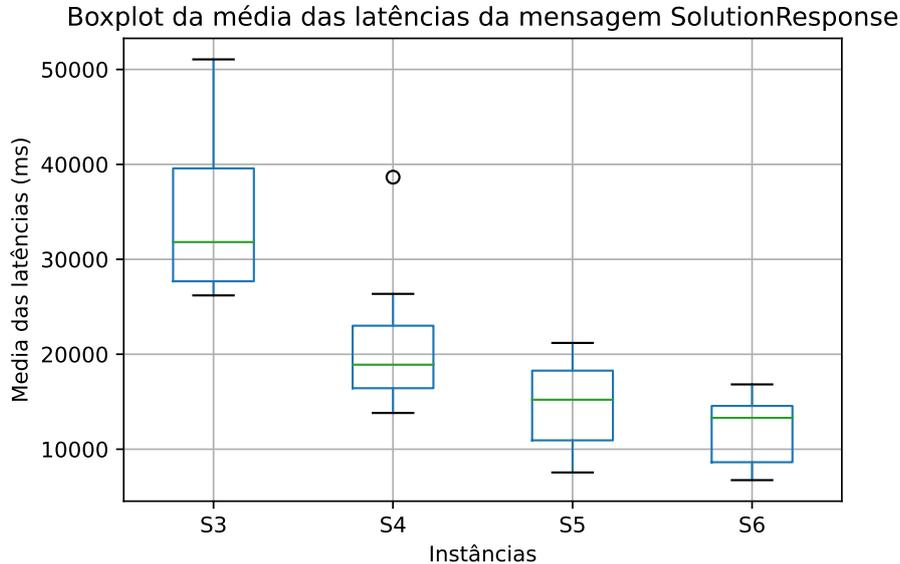
Finalmente, a distribuição das médias das latências para as mensagens do tipo **SolutionResponse** são exibidas na Figura 10. Pela análise gráfica é possível dizer que há diferença entre a instância S3 e S4, S5 e S6. As observações são independentes e as variâncias são homogêneas (com p-valor de 0.060870).

Há um pequeno desvio na normalidade, o que permite executar uma análise paramétrica. A ANOVA indica que há diferença entre as médias, com p-valor igual a 0. A Tabela 6 indica que há diferença entre S4 e S6, além da diferença entre S3 que já era percebida na análise gráfica.

Tabela 6 – Comparações par-a-par (Teste de Tukey-HSD) para a média das latências de mensagens do tipo **SolutionResponse**

Instâncias		p-valor	Rejeita H_0 ?
S3	S4	0.001	Sim
S3	S5	0.001	Sim
S3	S6	0.001	Sim
S4	S5	0.0862	Não
S4	S6	0.004	Sim
S5	S6	0.6325	Não

Figura 10 – *Boxplot* das médias das latências da mensagem **SolutionResponse** para 14 execuções da arquitetura D-Optimas com duração de 1000 unidades de tempo discreto em 3, 4, 5 e 6 nós do *cluster*



5.3 Síntese do resultado

De uma maneira geral, aumentar o tamanho do *cluster*, mantendo a quantidade de agentes e o limite de regiões, significa oferecer mais recursos computacionais à arquitetura. A presença de mais núcleos de processamento, memória, largura de banda, favorece o balanceamento de carga. Ao distribuir melhor a carga entre os nós de processamento, é natural que o escalonamento entre os atores, o tempo de espera por I/O e a paginação de memória diminuam, fazendo que cada execução do ator seja mais eficiente.

Dito isto, os resultados mostram que há de fato uma tendência de queda da latência média com o aumento do número de nós, em especial, sempre que mais de um nó é adicionado na simulação. Em outras palavras, os atores são mais rápidos em processar as mensagens que recebem. Essa tendência é um indício de que a arquitetura foi construída de maneira a aproveitar de maneira eficiente os recursos que estão disponíveis para ela.

Entre os quatro tipos de mensagem analisadas neste experimento é possível observar que há uma diferença na ordem de grandeza e na variância comparando as mensagens recebidas pelo líder e as recebidas por outras entidades da simulação. É possível que este fenômeno seja reflexo de uma concentração de tarefas no líder. Mesmo que as tarefas não tenham uma complexidade computacional alta (a maioria delas possui complexidade assintótica da ordem $\mathcal{O}(n)$, onde n é o número de agentes e regiões na simulação), pode haver uma demora em atender as mensagens que se acumulam rapidamente. Ainda assim, é possível observar que o líder se beneficia também do aumento de recursos computacionais,

diminuindo a média da latência.

É preciso ressaltar que os resultados obtidos neste experimento não indicam uma escalabilidade da arquitetura sob qualquer condição. Os resultados são preliminares e dão bons indícios da resiliência da arquitetura como um sistema distribuído, mas para obter resultados mais robustos é necessário avaliar outros fatores, como por exemplo, o número máximo de regiões permitidas, o número de agentes em execução e o crescimento do conjunto de soluções produzido pela arquitetura. Além disso, é necessário avaliar o funcionamento da arquitetura num *cluster* com dezenas ou centenas de nós.

Avaliar o desempenho da D-Optimas é importante para certificar a correta implementação como um sistema distribuído, bem como a eficiência do *software*, principalmente para oferecê-lo como uma ferramenta de trabalho para um usuário final, um tomador de decisão num processo de otimização. De todo modo, este é um sistema de tempo discreto, todas as meta-heurísticas são algoritmos discretos, que não dependem de resposta em tempo real. É claro que uma latência baixa é desejável, mas ela de fato não influencia na qualidade das soluções, somente no tempo de espera por soluções de boa qualidade.

Capítulo 6

Avaliação do efeito da diversidade na qualidade das soluções produzidas

Rir, antes da hora, engasga.

Riobaldo, em "Grande Sertão: Veredas"

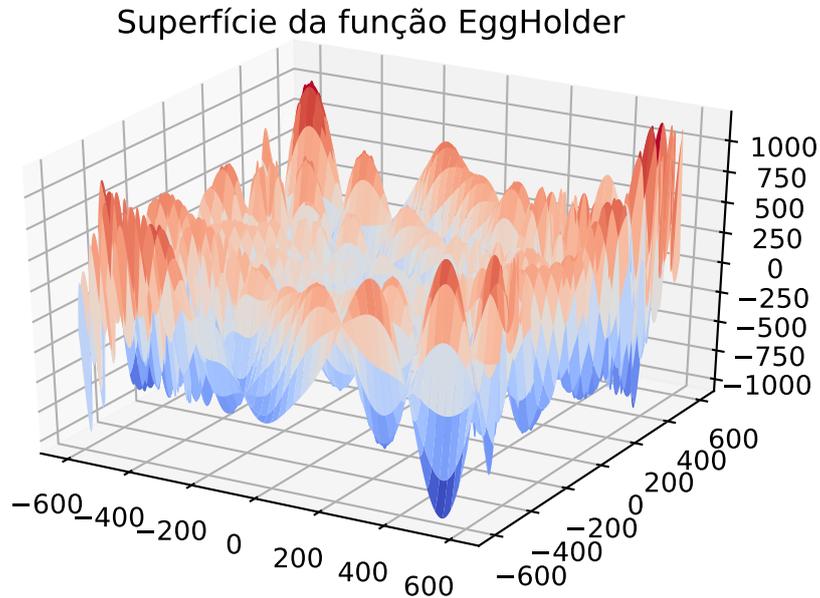
Uma vez avaliada a capacidade de escalar horizontalmente, o atual capítulo descreverá os experimentos realizados para avaliar o efeito da diversidade de agentes na qualidade das soluções produzidas pela arquitetura.

Esta bateria de experimentos foi executada em máquinas virtuais no *Google Cloud Platform*¹, em instâncias do tipo e2-standard-4 com 4 vCPUs e 8Gb de memória RAM. Nestes experimentos, a arquitetura executou como um único nó, conectada a dois nós do banco de dados *Cassandra*. Os experimentos foram provisionados com o auxílio da ferramenta *Ansible*, e executaram isoladamente em um ambiente *Docker*. Para todas as configurações dos experimentos foram coletadas 20 amostras. Essas amostras incluem o valor de todas as soluções encontradas pela arquitetura, bem como dados do comportamento das regiões (divisões e fusões) e dos agentes (uso de memória). O critério de parada do experimento foi atingir o limite de tempo discreto da arquitetura de 1000 unidades.

Ainda sobre a coleta de dados, aqui foram analisados somente os valores finais produzidos pela função objetivo. Diferente do experimento descrito no capítulo anterior, que computou a latência de todas as mensagens de um determinado tipo de mensagem, e analisou a média das latências de diferentes amostras. Naquele caso, quando a amostra não atendia a premissa de normalidade, principalmente no caso da ANOVA, essa condição foi relaxada sem prejuízo da utilização do método. Isso porque, de acordo com o teorema do limite central, a média de um conjunto de médias tende para uma distribuição normal. Para os

¹<https://cloud.google.com/compute/docs/machine-types>

Figura 11 – Superfície da função *EggHolder*. O ponto mínimo está logo no centro inferior do gráfico em coloração azul escuro. Os demais vales estão em azul e os picos em vermelho.



Fonte: O próprio autor

valores aqui apresentados, a premissa de normalidade não foi atendida em nenhuma das amostras. Deste modo, apenas os testes não-paramétricos serão utilizados neste capítulo.

Todos os experimentos foram executados sobre a função *EggHolder*, que está descrita na equação abaixo.

$$f(x, y) = -(y + 47) * \sin \sqrt{\left| y + \frac{x}{2.0} + 47 \right|} - x * \sin \sqrt{\left| x - (y + 47) \right|}$$

Este problema é interessante para a arquitetura, pois possui uma grande quantidade de mínimos locais. A Figura 11 exibe a superfície da função objetivo. O espaço de busca foi restringido no intervalo de $[-512, +512]$ em ambas as coordenadas. O mínimo global está localizado na posição $f(512, 404.2319) = -959.6407$.

Dito isto, o presente capítulo se divide em seis seções. A primeira seção trata dos experimentos preparatórios para calcular o tamanho da amostra dos experimentos seguintes. A Seção 6.2 trata dos experimentos de *baseline*, que servirão de base de comparação para os experimentos seguintes. Esta etapa do procedimento avaliou o resultado de cada uma das meta-heurísticas presentes na arquitetura D-Optimas na presença de um agente gerador de

soluções e uma única região. Esta configuração é a mais simples que pode ser executada, uma vez que o agente gerador é necessário para inicializar o agente explorador do espaço de busca. A Seção 6.3 trata da hibridização de um algoritmo populacional e uma estratégia de busca local. Para que os dois agentes que passam a compor a simulação possam colaborar na exploração do espaço de busca, na Seção 6.4 é avaliado o impacto do aumento do número mínimo de regiões na qualidade das soluções finais encontradas pela arquitetura.

Uma vez apreciado o comportamento de um agente populacional acompanhado de uma busca local, a Seção 6.5 avalia o efeito do aumento do número de agentes no resultado final produzido pela arquitetura D-Optimas.

Por fim, a Seção 6.6 avalia o efeito da composição de várias heurísticas populacionais, juntamente com a busca local, comparando os resultados obtidos nesta etapa com todas as configurações anteriores. A síntese do resultado e as considerações finais deste capítulo estão dispostas na Seção 6.7.

6.1 Experimentos preparatórios

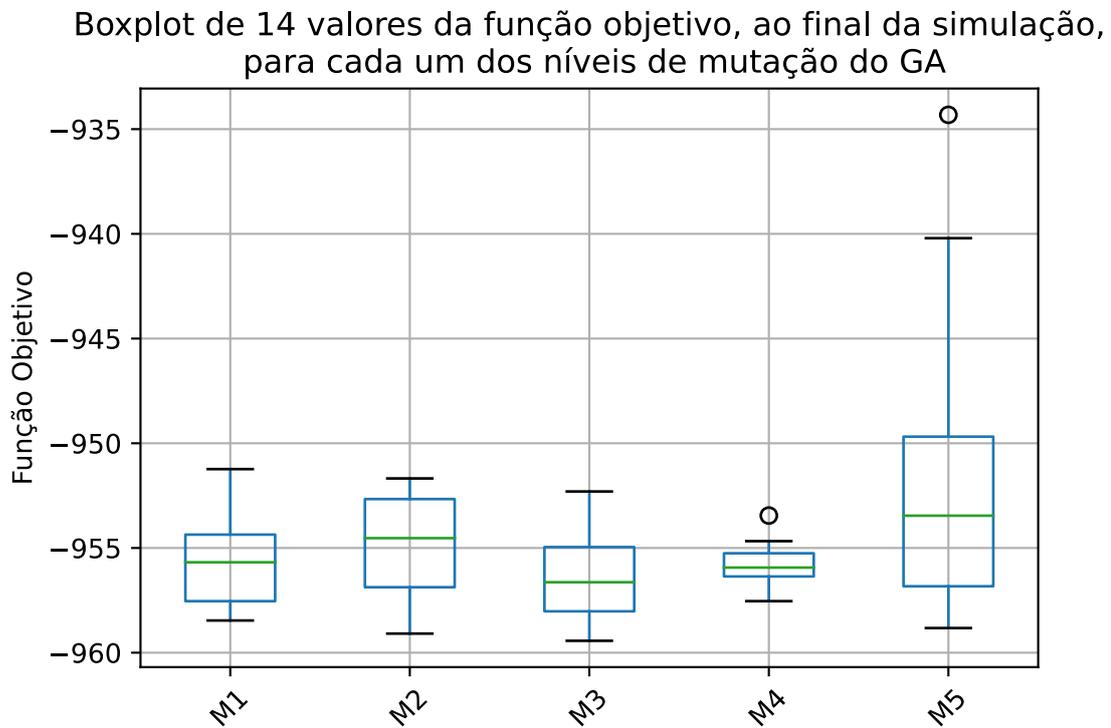
Foi realizado um experimento preparatório para avaliar o efeito da mudança de parâmetros na arquitetura, bem como a duração apropriada para o experimento. Esse experimento avaliou a arquitetura com um agente do tipo ILS, um agente do tipo GA e um agente gerador de soluções. O parâmetro variado foi a taxa de mutação do agente GA, que variou entre os níveis 0.05, 0.1, 0.15, 0.20 e 0.25. Este experimento foi repetido 14 vezes para cada um dos níveis. Os demais parâmetros das metaheurísticas foram configuradas conforme os dados apresentados na Tabela 7.

Os resultados obtidos no experimento são exibidos na Figura 12. Uma vez que a amostra não atende as premissas de normalidade e homocedasticidade (p-valores de 2.0×10^{-6} e 3.448×10^{-3} respectivamente), foi utilizado o teste não paramétrico de Kruskal-Wallis. O teste indica com p-valor de 0.208669 que não é possível identificar diferença significativa entre as amostras.

Deste modo, o experimento indica que para essa configuração da arquitetura, com um agente gerador de soluções, um agente de busca local e um agente de busca populacional, não faz diferença a escolha do parâmetro de mutação de um agente.

A análise do poder do teste indicou que o tamanho da amostra ideal para este experimento seria de 20 valores para atingir um poder do teste de 90%. Deste modo, esse foi o tamanho da amostra escolhido para os experimentos de *baseline* e avaliação da diversidade.

Figura 12 – Boxplot de 20 valores da função objetivo, ao final da simulação, para diferentes configurações da taxa de mutação do GA



Fonte: O próprio autor

6.2 Experimentos de *baseline*

O objetivo deste experimento é servir como base de comparação para os experimentos seguintes, uma vez que é difícil comparar a arquitetura D-Optimas com qualquer outro algoritmo *standalone*, por suas características de um sistema multi-agentes.

Deste modo, buscou-se executar a configuração mais simples possível para a arquitetura D-Optimas. Essa configuração inclui um agente gerador de soluções iniciais e um outro agente explorador do espaço de busca. Para o agente gerador de soluções iniciais foi escolhido o algoritmo GRASP. Ele foi executado juntamente com um agente do tipo ILS, GA, DE e PSO. A configuração dos agentes está disposta na Tabela 7.

Apresentada a configuração do experimento, a Figura 13 exhibe as 20 últimas soluções de cada uma das configurações. Não foi possível verificar as premissas de homocedasticidade e normalidade dos resíduos (os testes de Levene e Shapiro-Wilk forneceram os p-valores iguais a 0.000491 e 0.027282, respectivamente). Assim, o método de Kruskal-Wallis indicou diferença entre as amostras com p-valor de 0. Uma vez que há pelo menos uma amostra diferente, foi feito um teste de múltiplas comparações de Conover, e o resultado está apresentado na Tabela 8.

Tabela 7 – Parâmetros das metaheurísticas utilizadas para o experimento de *baseline*

Metaheurística	Parâmetro	Valor
GRASP	iterações	100
	iterações busca local	10
	alpha	0.5
ILS	iterações	500
	nível de distúrbio	7
DE	iterações	50
	tamanho da população	30
	taxa de mutação	0.2
	taxa de cruzamento	0.5
GA	iterações	500
	tamanho da população	30
	taxa de mutação	0.2
	taxa de cruzamento	0.5
PSO	iterações	500
	tamanho da população	30
	C1	0.15
	C2	0.15
	fator de inércia	1

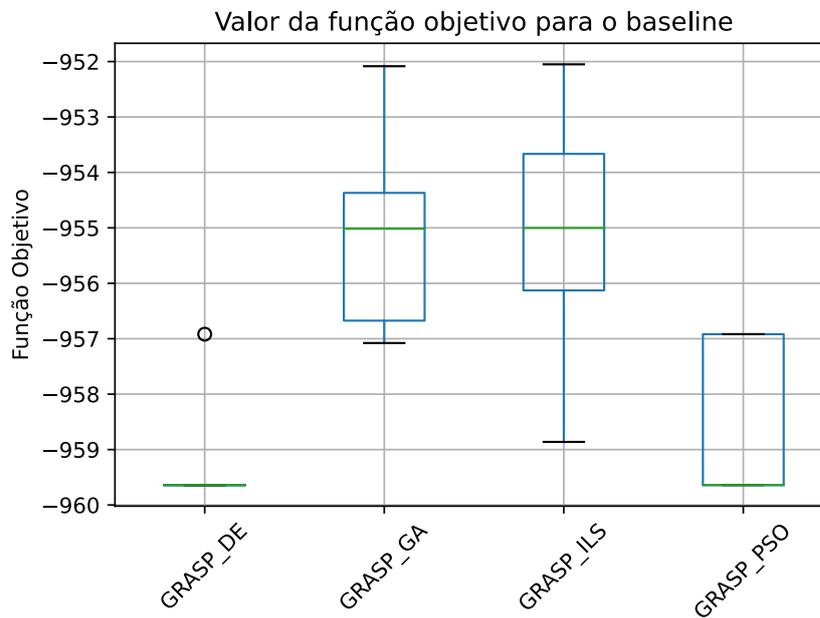
Fonte: O próprio autor

Tabela 8 – Comparações par-a-par (Teste de Conover) da média da função objetivo para as quatro configurações do experimento de *baseline*

Instância		p-valor	Rejeita H0?
GRASP_DE	GRASP_GA	3.706×10^{-16}	Sim
GRASP_DE	GRASP_ILS	1.451×10^{-16}	Sim
GRASP_DE	GRASP_PSO	0.005	Sim
GRASP_GA	GRASP_ILS	0.828	Não
GRASP_GA	GRASP_PSO	9.798×10^{-11}	Não
GRASP_ILS	GRASP_PSO	3.783×10^{-11}	Não

Observando os resultados do teste de Conover expostos na Tabela 8, constata-se que há diferença entre as instâncias GRASP_GA e GRASP_ILS (p-valor 0.828). Entretanto as configurações GRASP_DE e GRASP_PSO diferem das demais. Em especial, a configuração GRASP_DE obteve o melhor resultado para este problema, encontrando o ótimo-global ou se aproximando dele na maioria das execuções.

Figura 13 – Boxplot de 20 valores obtidos da função objetivo, ao final da simulação, para cada uma das metaheurísticas.



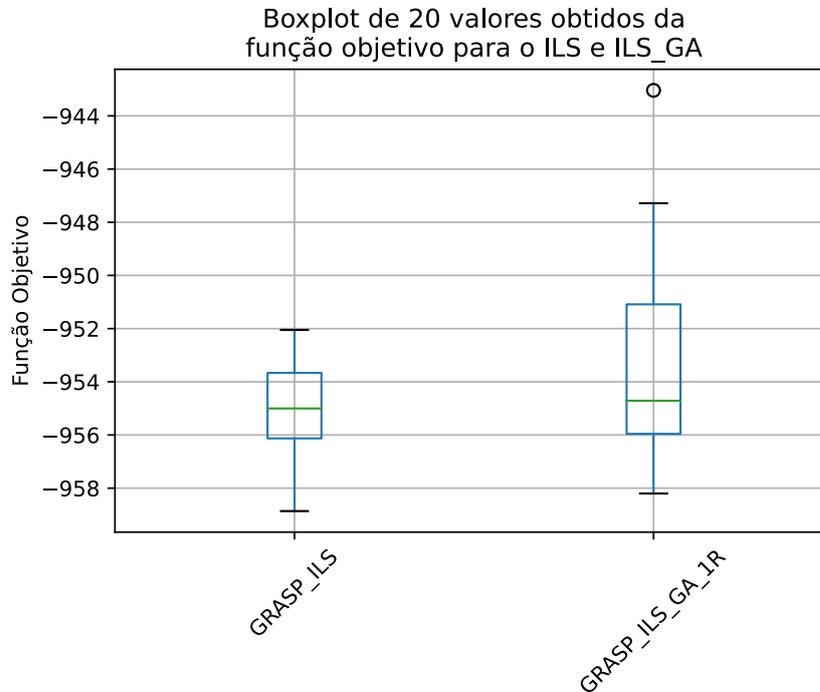
Fonte: O próprio autor

6.3 Avaliando o efeito da hibridização de um agente populacional e uma busca local

A seção anterior avaliou o resultado da configuração mais simples possível para a arquitetura D-Optimas, que compreende um agente gerador de soluções e um agente explorador do espaço de busca. Dado este primeiro passo, o presente experimento adiciona um pouco mais de complexidade e avalia a combinação de duas heurísticas fundamentalmente diferentes, a saber, uma heurística de busca local e uma heurística populacional. As configurações dos agentes utilizadas neste experimento foram as mesmas dispostas na Tabela 7. O tamanho da amostra também se manteve, bem como a duração do experimento. As combinações avaliadas neste experimento foram as seguintes: GRASP_ILS_DE, GRASP_ILS_GA e GRASP_ILS_PSO.

A Figura 14 compara o desempenho do baseline do algoritmo ILS (GRASP_ILS) com a sua hibridização com o GA (GRASP_ILS_GA). Como há sobreposição de caixas, não é possível dizer visualmente que há diferença entre as amostras. Os dados foram submetidos ao teste de normalidade de Shapiro-Wilk que forneceu o p-valor 0.000573, valor que está abaixo do nível de confiança e permite rejeitar a hipótese de que a amostra segue uma distribuição normal. Deste modo, os valores foram submetidos ao teste não-paramétrico de Kruskal-Wallis, que obteve o p-valor de 0.401720. Como este valor é maior do que o nível de significância de 5%, não é possível afirmar que há diferença entre utilizar somente

Figura 14 – Comparação de 20 valores obtidos da função objetivo do *baseline* ILS com a combinação do ILS e GA. Os dados não seguem uma distribuição normal, dado o p-valor de 0.000573. O teste de Kruskal-Wallis forneceu o p-valor de 0.401720, que falha em rejeitar a hipótese nula de que as amostras tem medianas iguais.



um ILS, ou utilizá-lo em conjunto com o GA.

A mesma constatação se repete quando comparado o *baseline* do algoritmo populacional (GRASP_GA) com a sua hibridização com a busca local (GRASP_ILS_GA). Este resultado está exibido na Figura 15. Novamente, não é possível dizer visualmente que há diferença entre as amostras. A amostra foi submetida ao teste de normalidade de Shapiro-Wilk, que forneceu o p-valor igual a 0.000032, indicando que os dados não seguem a distribuição normal. Assim, aplicando o teste de Kruskal-Wallis, obtive o p-valor de 0.233966 que é maior que o nível de significância de 5%. Este resultado portanto não permite afirmar que há diferença estatística relevante entre utilizar somente o GA ou combiná-lo com a metaheurística ILS.

Ao comparar os resultados do algoritmo PSO (GRASP_PSO) contra a sua hibridização com a busca local (GRASP_ILS_PSO), também não é possível afirmar que há diferença significativa entre as amostras. Este resultado está exibido na Figura 16. Não sendo possível aplicar um teste paramétrico neste resultado (o teste de Shapiro-Wilk forneceu o p-valor de 0.000000), foi aplicado o teste de Kruskal-Wallis. O p-valor obtido de 0.349729 falha em rejeitar a hipótese nula de que as medianas são diferentes.

Finalmente, resta comparar os resultados do algoritmo DE no experimento de *baseline*

Figura 15 – Comparação de 20 valores de função objetivo do baseline GA com a combinação do ILS e GA. O teste de normalidade rejeita a hipótese nula (p-valor 0.000032), e o teste dos postos falha em rejeitar a hipótese nula de que as amostras tem medianas iguais (p-valor de 0.233966)

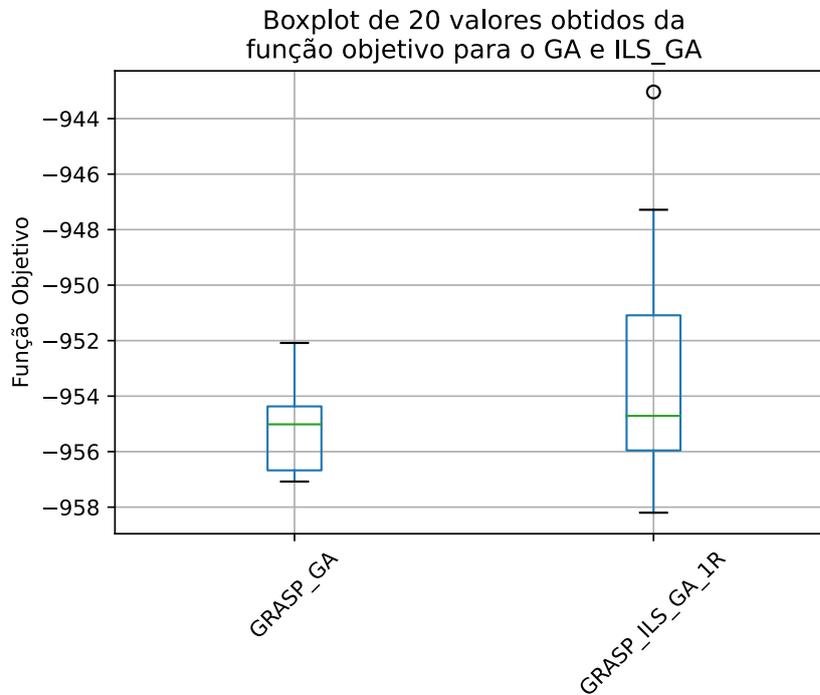


Figura 16 – Comparação de 20 valores obtidos da função objetivo do *baseline* PSO com a combinação do ILS e PSO. Os dados não seguem uma distribuição normal (p-valor igual a 0.000000), e o teste não-paramétrico falha em rejeitar a hipótese nula (p-valor igual a 0.349729).

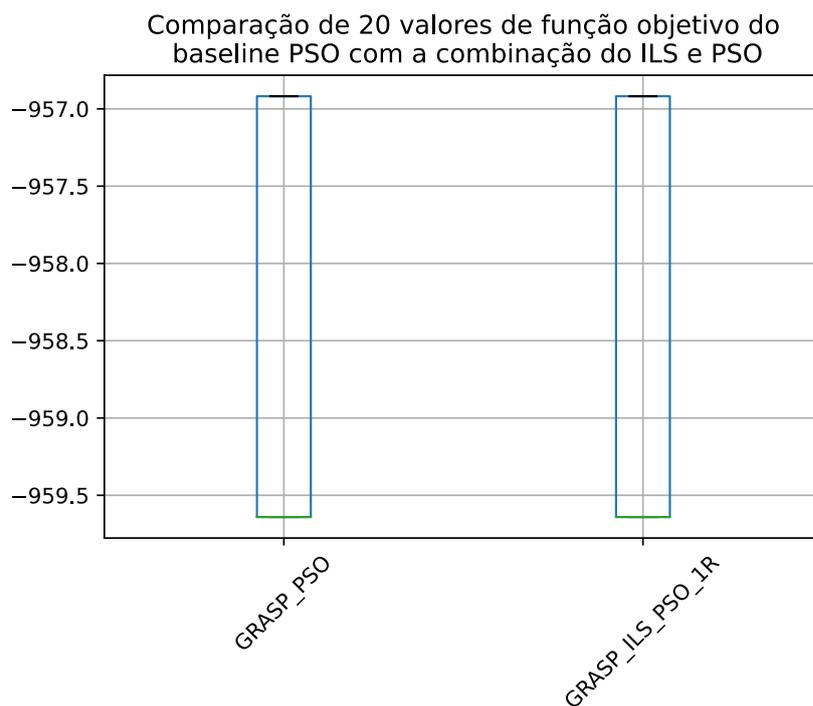
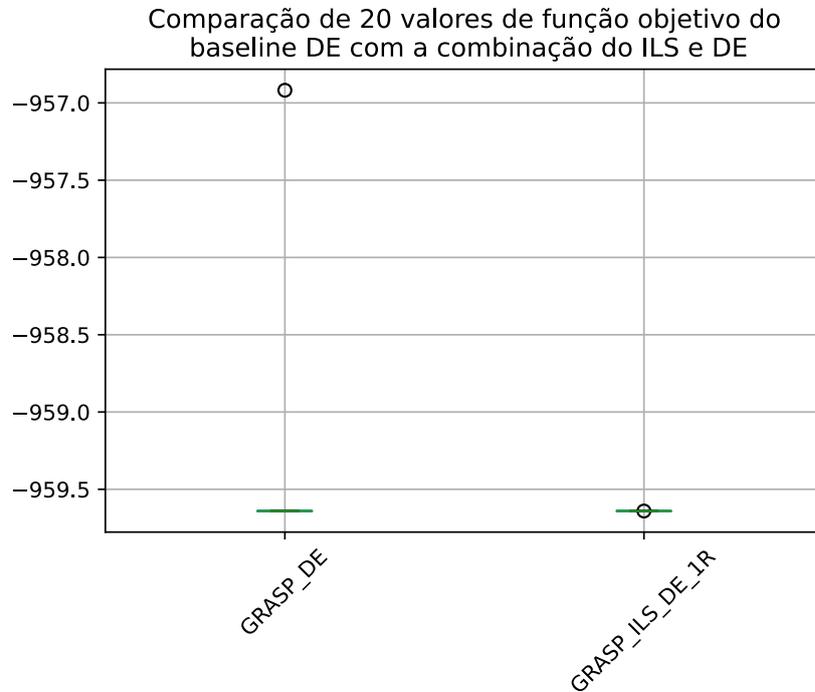


Figura 17 – Comparação de 20 valores obtidos da função objetivo do *baseline* DE com a combinação do ILS e DE. O teste de normalidade rejeita a hipótese nula com o p-valor de 0.000000. O teste de Kurskal-Wallis fornece o p-valor de 0.918293, que falha em rejeitar a hipótese nula, acusando uma diferença entre as amostras.



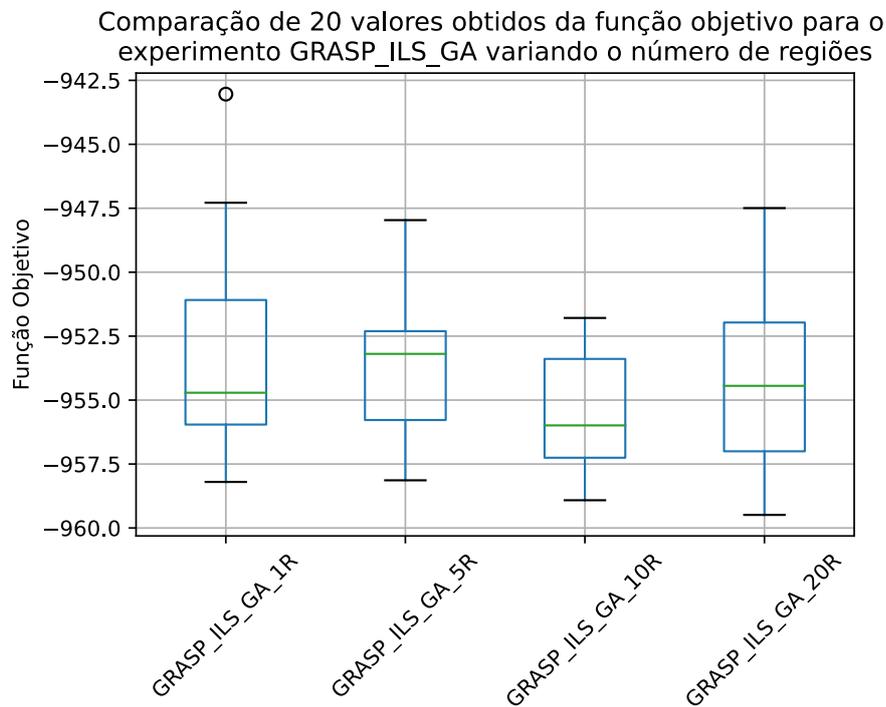
(GRASP_DE) contra os dados obtidos no experimento GRASP_ILS_DE. Estes dados encontram-se exibidos na Figura 17. Obteve-se o p-valor de 0.0000 ao testar a normalidade da amostra utilizando o teste de Shapiro-Wilk,. Deste modo, foi aplicado o teste de Kruskal-Wallis, que forneceu o p-valor de 0.018293 que é menor que o nível de significância de 5%. Neste caso há diferença estatística relevante entre o *baseline* que obteve um resultado médio de -959.50454 ± 0.608754 , contra a instância GRASP_ILS_DE cujo resultado médio foi de $-959.640664 \pm 0.0000002$.

6.4 Avaliando o efeito do aumento do número de regiões na presença de um agente populacional e uma busca local

Até o momento, foram avaliadas as configurações mais básicas da arquitetura, com um agente gerador de soluções iniciais, e um agente explorador do espaço de busca, bem como a hibridização de um algoritmo populacional e um algoritmo de busca local. Foi possível observar que para uma das combinações a hibridização fez sentido, e produziu melhoria, mas esse resultado não foi observado em todos os casos.

Sendo assim, após explorar a possibilidade de colaboração entre os agentes, introduzir

Figura 18 – Comparação de 20 valores de função objetivo do experimento com os agentes GRASP, ILS e GA para 1, 5, 10 e 20 regiões



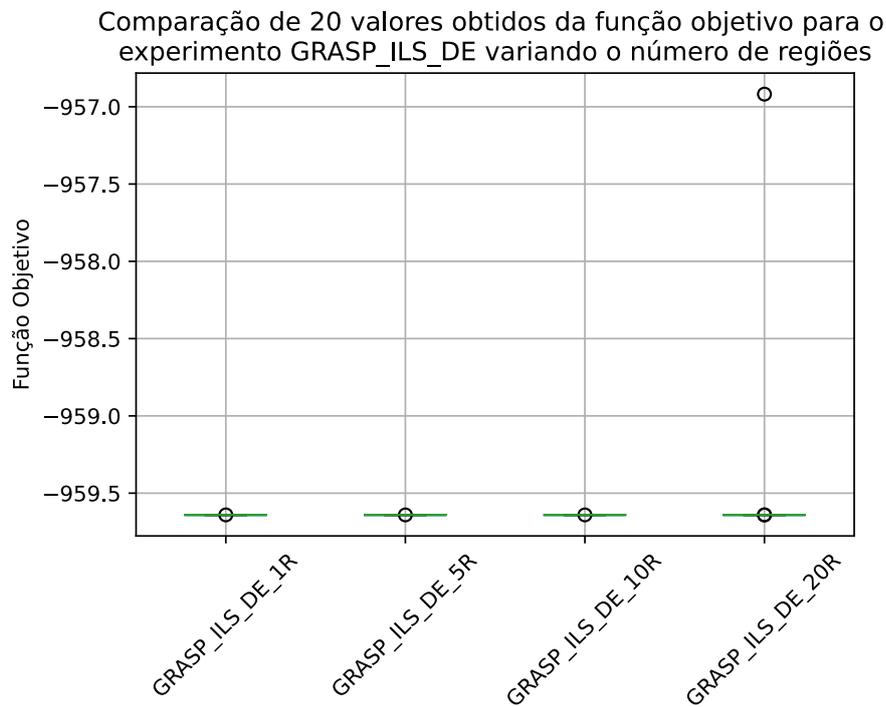
mais regiões no sistema abre a possibilidade de segmentação do espaço de busca. Isso possibilita aos agentes formar uma memória das últimas regiões exploradas e da qualidade das soluções produzidas por cada uma delas.

Deste modo, neste novo experimento, variou-se o fator número de regiões em 4 níveis: o primeiro, com uma única região, o segundo com 5 regiões, que também corresponde ao limite da memória do agente, e os próximos níveis com 10 e 20 regiões. Este experimento foi repetido para as três hibridizações estudadas até agora. Os resultados são exibidos nas Figuras 18, 19 e 20.

Há que se fazer uma ressalva a este experimento, no que diz respeito à duração da instância correspondente ao fator de 20 regiões. Para este caso, foi possível observar que a duração de 1000 unidades de tempo discreto não eram suficientes para explorar além do limite de 10 regiões. Deste modo, para o caso de 20 regiões nos experimentos GRASP_ILS_DE, GRASP_ILS_GA e GRASP_ILS_PSO, o experimento foi executado com a duração de 2000 unidades de tempo.

Não sendo possível verificar a premissa da ANOVA de normalidade, foi aplicado o teste não-paramétrico de Kruskal-Wallis. O teste não acusou nenhuma diferença em nenhuma das três instâncias, obtendo os p-valores de 0.599747, 0.175647 e 0.111431 para os experimentos GRASP_ILS_DE, GRASP_ILS_GA e GRASP_ILS_PSO, respectivamente. É importante ressaltar que, mesmo aumentando o tempo de duração do experimento para

Figura 19 – Comparação de 20 valores de função objetivo do experimento GRASP, ILS e DE para 1, 5, 10 e 20 regiões



Fonte: O próprio autor

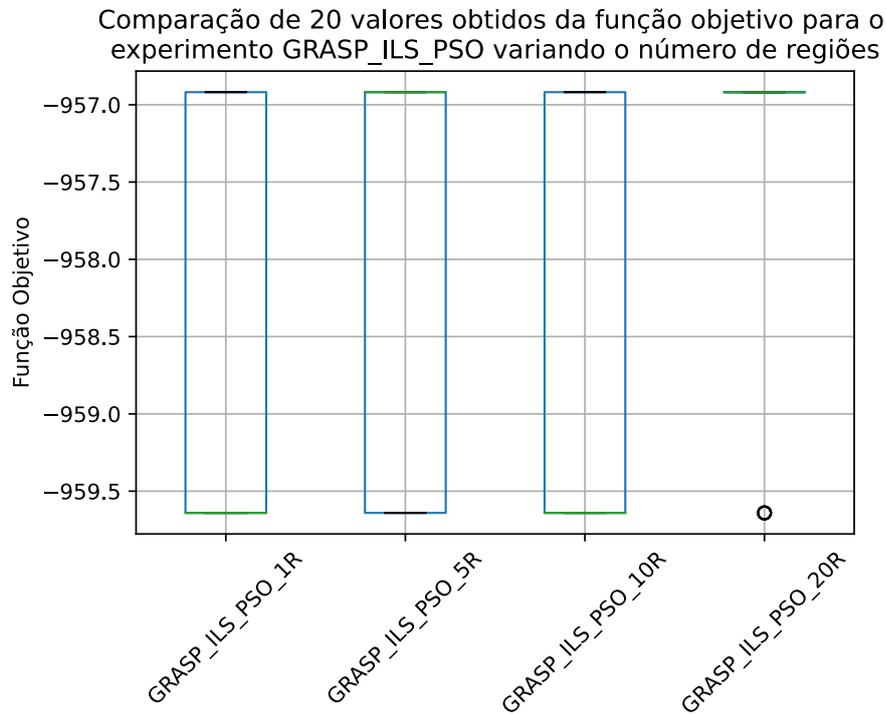
20 regiões, o mero aumento do número de regiões não produz qualquer efeito no resultado final global.

Diante disto, o fator seguinte que deve ser avaliado é a diversidade de agentes no sistema. Entretanto, antes de avaliar a hibridização de diferentes agentes populacionais, há que se observar que, aumentar o número de agentes na simulação implica, conseqüentemente, no aumento do número de avaliações da função objetivo. Considerando que avaliar mais vezes a função objetivo pode melhorar o resultado, e é um fator que deve ser analisado. Deste modo, um novo experimento foi realizado, no qual se pretende avaliar o o efeito causado pelo aumento do número de agentes do mesmo tipo, com os mesmos parâmetros, bem como com parâmetros diferentes.

6.5 Efeito do aumento do número dos agentes e variações em suas configurações

Esta seção apresenta os resultados para duas variações do conceito de diversidade sobre o experimento GRASP_ILS_GA. A primeira amostra corresponde ao simples aumento da quantidade de agentes, mas mantendo as suas configurações iguais. Portanto, este experimento executou com 2 agentes do tipo ILS e 4 agentes do tipo GA, todos com

Figura 20 – Comparação de 20 valores de função objetivo do experimento GRASP, ILS e PSO para 1, 5, 10 e 20 regiões



Fonte: O próprio autor

a mesma configuração, descrita na Tabela 7. Uma segunda bateria de experimentos foi executada, com os mesmos 7 agentes do primeiro experimento com modificações dos parâmetros das metaheurísticas. Os parâmetros que se diferenciaram estão descritos na Tabela 9. Para as duas configurações foram executadas 4 instâncias, variando o número de regiões em 1, 5, 10 e 20. A instância com 20 regiões também teve o seu tempo de duração aumentado, pelo mesmo motivo descrito na seção anterior.

Tabela 9 – Parâmetros das metaheurísticas utilizadas para o experimento que avaliou a diversidade de configurações de agentes

Metaheurística	Parâmetro	Agente	Valor
ILS	Distúrbio	1	4
		2	7
GA	Mutaç�o	1	0.10
		2	0.15
		3	0.20
		4	0.25

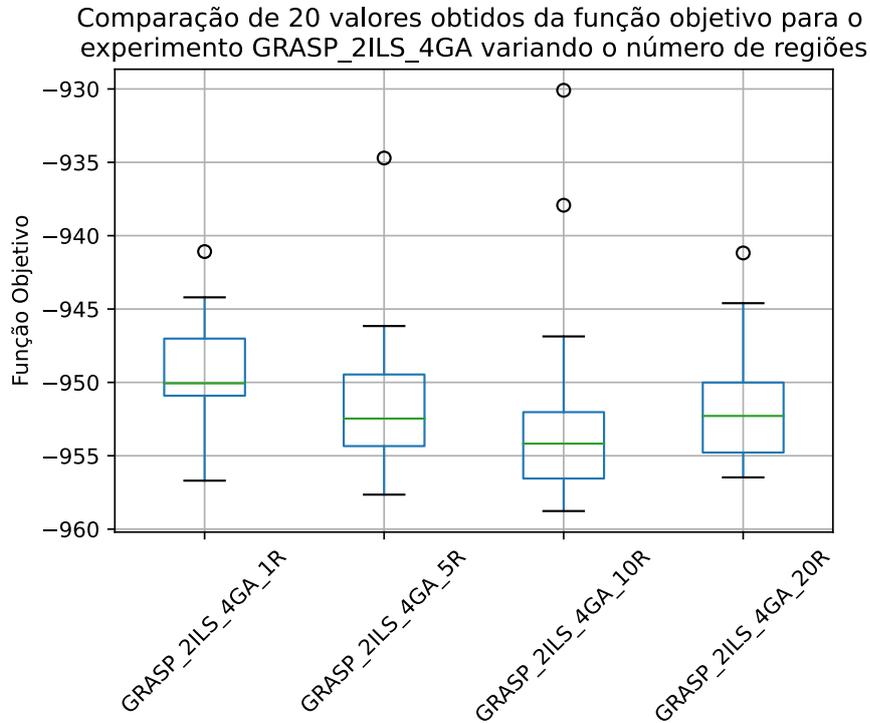
Fonte: O próprio autor

A Figura 21 exibe o resultado para a configuração na qual não há diversidade de configurações. Não tendo sido possível verificar a premissa de normalidade da ANOVA, os

Tabela 10 – Comparações par-a-par (Teste de Conover) da média da função objetivo para o experimento GRASP_2ILS_4GA

Instância		p-valor	Rejeita H0?
GRASP_2ILS_4GA_1R	GRASP_2ILS_4GA_5R	0.125922	Não
GRASP_2ILS_4GA_1R	GRASP_2ILS_4GA_10R	0.004907	Sim
GRASP_2ILS_4GA_1R	GRASP_2ILS_4GA_20R	0.163585	Não
GRASP_2ILS_4GA_5R	GRASP_2ILS_4GA_1R	0.125921	Não
GRASP_2ILS_4GA_5R	GRASP_2ILS_4GA_10R	0.180877	Não
GRASP_2ILS_4GA_5R	GRASP_2ILS_4GA_20R	0.888501	Não
GRASP_2ILS_4GA_10R	GRASP_2ILS_4GA_20R	0.140067	Não

Figura 21 – Comparação de 20 valores obtidos da função objetivo do experimento com os agentes GRASP, 2 agentes ILS e 4 agentes GA, configurados com os mesmos parâmetros, limitados a 1, 5, 10 e 20 regiões

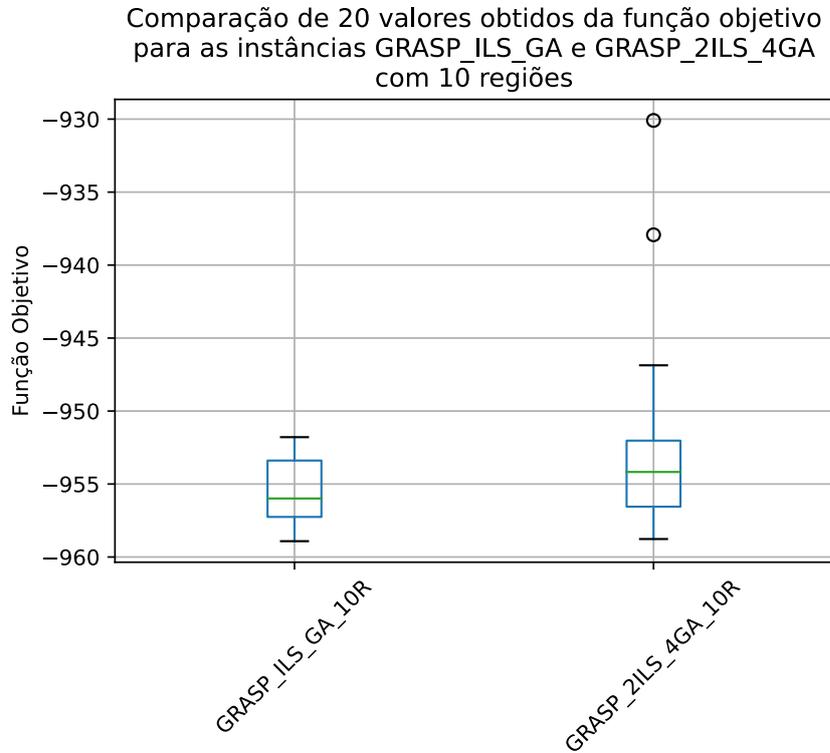


Fonte: O próprio autor

resultados foram submetidos ao teste de Kruskal-Wallis que produziu o p-valor de 0.048590, levando à rejeição da hipótese nula. Há uma diferença visual entre a instância limitada a 10 regiões e as demais instâncias, o que também pode ser verificado pelo teste de Conover que produziu o p-valor de 0.004907, de acordo com a Tabela 10.

Entretanto, ao comparar o resultado obtido para 10 regiões, mostrado na Figura 22, com o mesmo experimento sem a variedade de agentes o teste de Kruskal-Willis falha em rejeitar a hipótese nula, com o p-valor de 0.08, de modo que não há qualquer melhoria

Figura 22 – Comparação de 20 valores obtidos da função objetivo do experimento com os três agentes GRASP, ILS e GA para a configuração com vários agentes com as mesmas configurações para o caso com 10 regiões



Fonte: O próprio autor

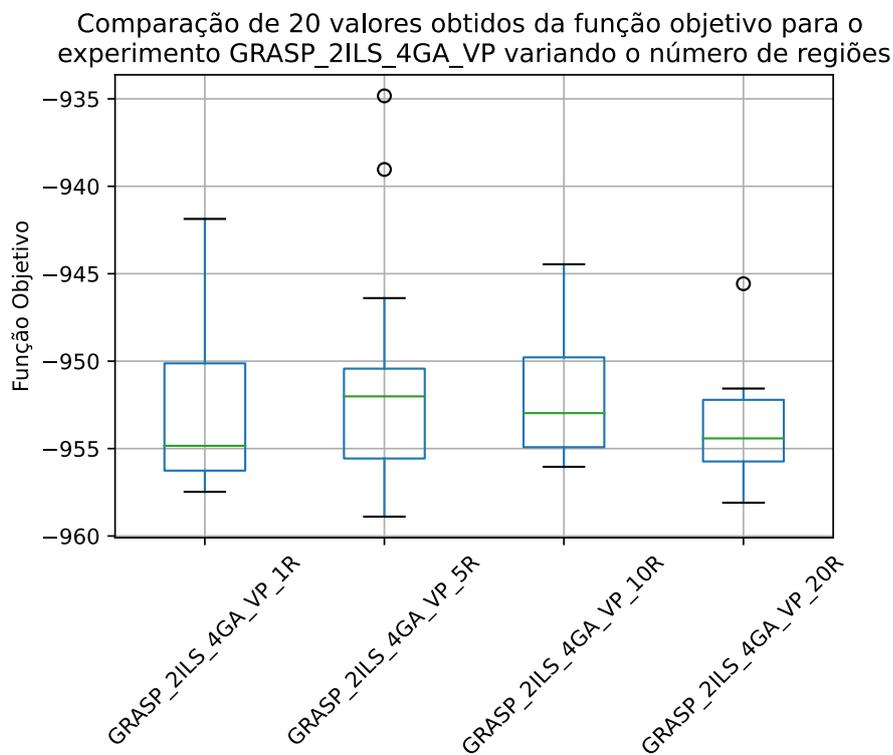
entre o *baseline* e o melhor resultado obtido com a variedade de agentes. Contudo, há um indício de que na presença de um número variado de agentes, a arquitetura necessita também de uma variedade de regiões para obter um bom resultado. Há também o indício de que existe um valor ótimo para o número de regiões uma vez que, mesmo com uma duração maior, a instância GRASP_2ILS_2GA_20R não obteve o mesmo resultado que a instância GRASP_2ILS_2GA_10R.

Dando continuidade aos testes, a Figura 23 exhibe os resultados do experimento no qual foram considerados a diversidade de configuração entre os agentes, bem como a variação do número máximo de regiões. Não é possível para este caso distinguir visualmente se as amostras tem origem em diferentes populações. O teste de normalidade rejeita a hipótese nula com o p-valor de 0.000014. Assim, os dados foram submetidos ao teste de Kruskal-Wallis, que para esta amostra produziu o p-valor de 0.302437, e portanto falha ao rejeitar a hipótese nula de que existe alguma diferença entre as amostras.

Dado que não há diferença entre as amostras com diferentes limites de regiões, escolheu-se a amostra GRASP_2ILS_4GA_VP_10R, cuja média é de -951.858922 , para comparar com a instância GRASP_ILS_GA_10R onde não há variedade de agentes, que obteve um

resultado médio de -955.520160 . O mesmo teste não-parâmetro foi utilizado, e produziu o p-valor de 0.002449 , que levou a rejeitar a hipótese nula, indicando que há diferença entre as amostras. Sendo assim, é possível perceber que a configuração com variedade de configurações produz um resultado pior do que o grupo de controle, uma vez que o valor médio obtido pela arquitetura com esta configuração foi maior. Este resultado não corrobora para a verificação da hipótese de que a diversidade é um fator relevante para a obtenção de bons resultados pela arquitetura.

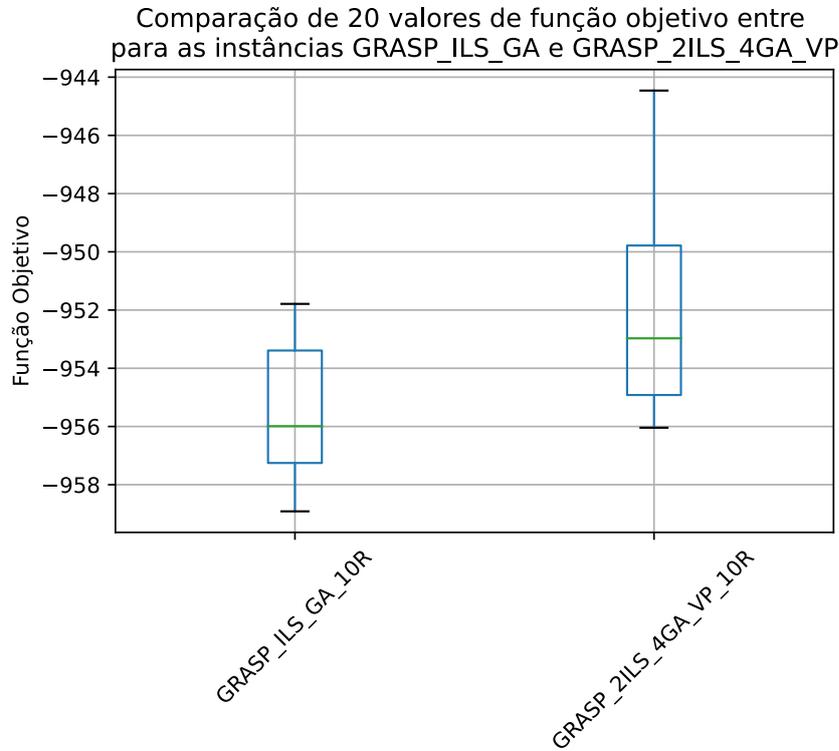
Figura 23 – Comparação de 20 valores de função objetivo do experimento com os agentes GRASP, 2 agentes ILS e 4 agentes GA, variando o parâmetro de distúrbio e mutação dos agentes, limitados a 1, 5, 10 e 20 regiões



Fonte: O próprio autor

Apesar do resultado apresentado nessa seção não ser encorajador da hipótese de que diversidade é um fator que contribui para uma melhor exploração do espaço de busca, vale ressaltar que este fator foi explorado de maneira superficial neste experimento. Também é importante frisar que uma alteração em um parâmetro de um algoritmo populacional e de busca local ainda é um tímido passo à diversidade de fato. Fazendo um paralelo com sistemas biológicos, o primeiro experimento desta seção se compararia a um conjunto de clones trabalhando em uma mesma tarefa. Os clones não adicionam nenhum outro fator a não ser a multiplicação da capacidade de trabalho. O segundo experimento pode ser comparado a uma população de indivíduos da mesma espécie, com características ligeiramente diferentes entre si, mas que em essência exploram o espaço de busca da mesma

Figura 24 – Comparação de 20 valores obtidos da função objetivo do experimento com os três agentes GRASP, ILS e GA para a configuração com vários agentes com diferentes configurações de distúrbio e mutação para o caso com 10 regiões



Fonte: O próprio autor

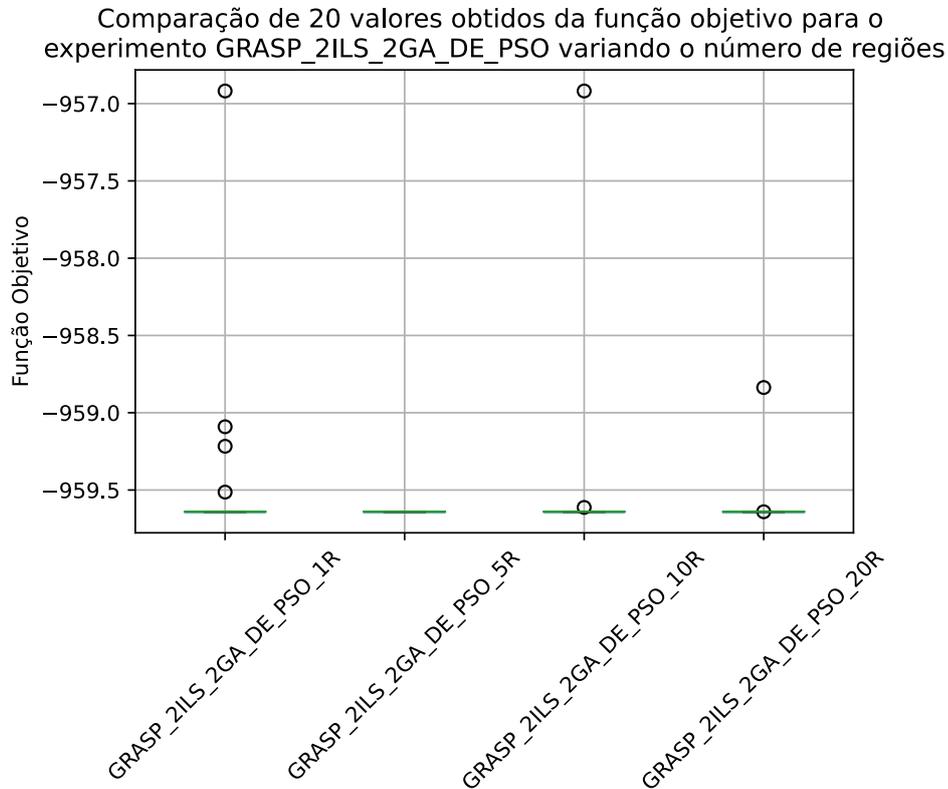
maneira.

6.6 Efeito da combinação de diferentes agentes populacionais

Os último experimento avaliou a combinação de vários agentes do mesmo tipo, com os mesmos parâmetros, bem como com variações nos parâmetros dos agentes. Executar uma simulação com um único tipo de agente não representa uma diversidade de estratégias de busca, apesar de ser uma estratégia de paralelismo que pode ser explorada em um *cluster*, e diminuir o tempo de busca. Entretanto, foi possível perceber que ter vários agentes do mesmo tipo com configurações diferentes pode inserir ruído no sistema e levar a um resultado pior, comparado ao *baseline*.

Diante disto, o presente experimento avalia a qualidade das soluções produzidas pela combinação de diferentes agentes populacionais. Este experimento foi executado com as mesmas configurações da Tabela 7, contando com um agente gerador de soluções iniciais, dois agentes de busca local, dois agentes executando a metaheurística GA, um agente DE e um agente PSO. As configurações das metaheurísticas utilizadas neste experimento

Figura 25 – Comparação de 20 valores obtidos da função objetivo do experimento com os agentes GRASP, 2 agentes ILS e 2 agentes GA, 1 agente DE e 1 agente PSO, variando o parâmetro de distúrbio e mutação dos agentes, limitados a 1, 5, 10 e 20 regiões



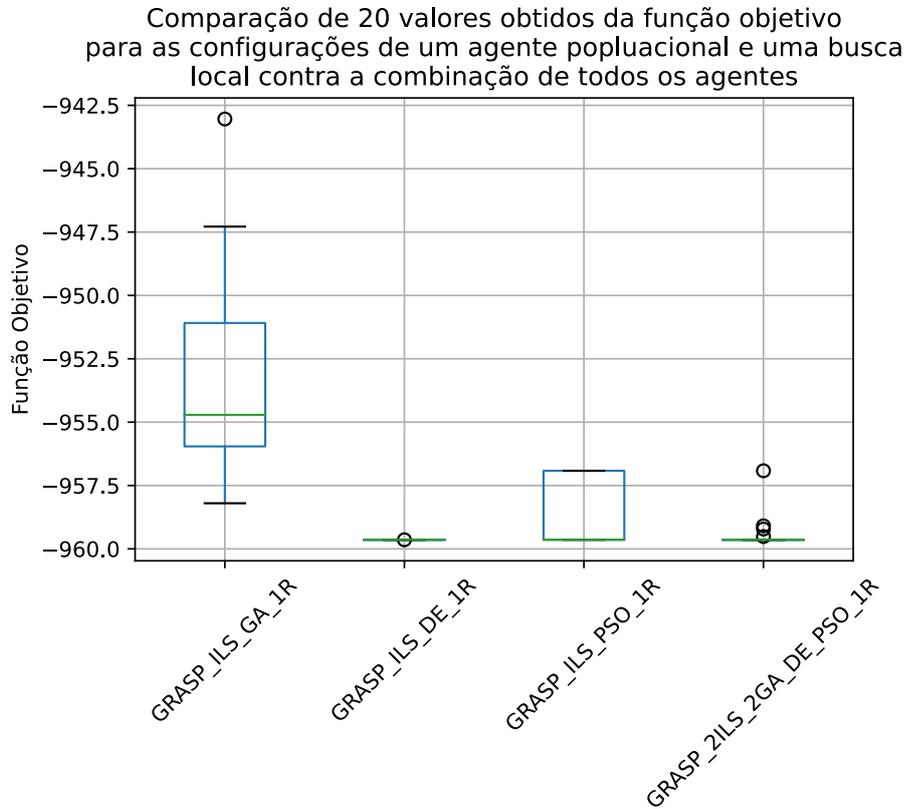
Fonte: O próprio autor

foram mesmas exibidas na Tabela 7.

Os resultados obtidos para 1, 5, 10 e 20 regiões são exibidos na Figura 25. Foi aplicado o teste de normalidade Shapiro-Wilk que produziu o p-valor de 0.000000, que leva a rejeitar a hipótese nula de que os dados seguem uma distribuição normal. Desta forma, a amostra foi submetida ao teste de Kruskal-Wallis, que produziu o p-valor de 0.087670 para esta amostra, que indica que não há diferença entre as quatro amostras. Isso nos leva a rejeitar a hipótese nula de que aumentar o número de regiões produz algum efeito no resultado final da arquitetura na presença de diferentes tipos de agentes.

Uma vez que o número de regiões não influencia no resultado final, as amostras correspondente a 1 região dos experimentos GRASP_ILS_GA, GRASP_ILS_DE e GRASP_ILS_PSO foram comparadas com o resultado apresentado nesta seção. A comparação está disposta na Figura 26. É possível perceber uma diferença entre a primeira e a segunda e a quarta amostra pelo princípio da sobreposição de caixas. Esse resultado é confirmado pelo teste de Kruskal-Wallis que produz o p-valor de 0.000000. O resultado aqui que mais interessa é

Figura 26 – Comparação de 20 valores de função objetivo do experimento com os agentes GRASP, 2 agentes ILS e 4 agentes GA, variando o parâmetro de distúrbio e mutação dos agentes, limitados a 1, 5, 10 e 20 regiões



Fonte: O próprio autor

Tabela 11 – Comparações par-a-par (Teste de Conover) da média dos valores obtidos da função objetivo para os experimento GRASP_ILS_GA, GRASP_ILS_DE, GRASP_ILS_PSO e GRASP_2ILS_2GA_DE_PSO

Instância		p-valor	Rejeita H0?
GRASP_ILS_GA	GRASP_ILS_DE	1.512×10^{-18}	Sim
GRASP_ILS_GA	GRASP_ILS_PSO	1.488×10^{-09}	Sim
GRASP_ILS_GA	GRASP_2ILS_2GA_DE_PSO	2.597×10^{-11}	Sim
GRASP_ILS_DE	GRASP_ILS_PSO	9.287×10^{-06}	Sim
GRASP_ILS_DE	GRASP_2ILS_2GA_DE_PSO	0.0002	Sim
GRASP_ILS_PSO	GRASP_2ILS_2GA_DE_PSO	0.3569	Não

a comparação entre o presente experimento e o GRASP_ILS_DE, pois foi a configuração que obteve o melhor resultado em relação ao *baseline*. O teste de Conover rejeita a hipótese nula de que os resultados são iguais com p-valor de 2.024374×10^{-17} (vide Tabela 11). O presente experimento (GRASP_ILS_2GA_DE_PSO_10) obteve um resultado médio de 959.503136 ± 0.608456 enquanto a amostra de controle (GRASP_ILS_DE_10R) obteve o resultado médio de -959.640665 ± 0.000002 .

O resultado portanto reforça a desconfiança de que a diversidade de configurações poderia produzir uma melhora expressiva no resultado para este problema, ou mesmo se equiparar ao melhor resultado encontrado pela configuração de *baseline*.

6.7 Síntese do resultado e considerações finais

O presente capítulo apresentou o procedimento experimental desenvolvido para testar a principal hipótese deste trabalho: a arquitetura é capaz de produzir soluções de melhor qualidade para diferentes problemas através da diversidade de estratégias de busca, combinada com um sistema de memória e um mecanismo de colaboração entre os agentes.

Para isso, foi escolhido um problema clássico da literatura que possui vários mínimos locais. Sobre este problema foram executadas as configurações mais básicas da arquitetura, de modo a compor um conjunto *baseline* para a comparação dos resultados das configurações seguintes. Esta configuração básica foi composta de um agente gerador de soluções iniciais, e de um agente explorador do espaço de busca, seja ele de busca local ou populacional. Foi possível observar que o algoritmo DE, ainda que não calibrado para alcançar o mínimo global em uma única interação com o sistema, conseguiu um resultado significativamente melhor do que os outros algoritmos.

Construída a amostra de partida para as comparações, o experimento seguinte avaliou o resultado de três algoritmos em conjunto: um algoritmo gerador de soluções iniciais, uma busca local e um algoritmo populacional. Neste experimento foi possível observar a interação entre os agentes, denominada aqui por hibridização dinâmica. Contudo, essa hibridização nem sempre se mostrou benéfica para o resultado final da arquitetura.

Ainda com o objetivo de estudar as interações entre os agentes e a arquitetura, o experimento seguinte avaliou o efeito do aumento do número de regiões no sistema. A expectativa é que os agentes conseguissem explorar as regiões ao enviar mensagens de *broadcast* pedindo por novas soluções, e pudessem aprender com o auxílio da memória *Q-learning* quais regiões seriam as mais promissoras. Essa expectativa não se concretizou neste experimento, uma vez que não foi possível perceber diferença estatística relevante no resultado em nenhuma das três configurações estudadas.

Desta forma, o experimento seguinte adicionou mais um nível de complexidade, adicionando mais agentes à simulação. Entretanto, manteve-se aqui o tipo dos agentes, bem como suas configurações. Neste ponto foi possível perceber uma diferença estatística relevante entre os diferentes limites de regiões. Ainda assim, este experimento obteve resultados estatisticamente iguais aos do *baseline*. Isso pode indicar que em um ambiente diverso, a arquitetura lida melhor com o espaço de busca fragmentado em regiões.

Apesar da variedade de agentes com a mesma configuração não produzir nenhum efeito em relação ao *baseline*, quando realizado o mesmo experimento variando as configurações do agente, observa-se o efeito contrário ao esperado. Os resultados foram piores que o *baseline*, e as regiões, de maneira geral, não produzem diferença estatística. Esse resultado pode indicar que agentes pouco adaptados podem acabar por atrapalhar agentes que estão melhor calibrados.

Finalmente, o último experimento avaliou a hibridização de diferentes tipos de agentes populacionais. Este último experimento também não obteve performance superior quando comparado ao melhor resultado do conjunto de *baseline*, indicando que para este problema ainda seria melhor utilizar somente o DE e uma busca local.

Colocados estes resultados em perspectiva, é necessário levantar algumas ressalvas. A primeira delas é que este experimento foi executado em um único problema de otimização de duas dimensões. A arquitetura sequer foi testada sob uma família de problemas, o que impede a generalização deste resultado ou qualquer conclusão forte sobre ele. A segunda ressalva é que, dos algoritmos apresentados neste experimento, DE sozinho se destaca pelos resultados superiores aos demais algoritmos. Ainda assim, foi possível perceber uma melhoria no seu desempenho ao hibridizá-lo com um agente de busca local. Essa melhoria foi pequena e não foi possível observá-la nos outros algoritmos. Os resultados não atendem a premissa de normalidade, fato que impede a condução de uma análise do poder do teste para este resultado em particular. As regiões se mostraram efetivas em somente um dos resultados, o que indica que há um problema em sua dinâmica, de modo que ela não contribui para o processo de escolha e otimização dos agentes. A última ressalva diz respeito à escala do experimento, que ficou limitada ao ambiente disponível na Plataforma *Google Cloud*, não sendo possível executar este mesmo experimento para uma quantidade maior de agentes/regiões.

Dadas as condições em que este experimento foi executado e os resultados obtidos, a principal contribuição deste trabalho é prover um procedimento experimental que pode ser aplicado a outros problemas, ou à famílias de problemas. Este trabalho também revelou indícios de que a dificuldade dos problemas de otimização, bem como a adaptação dos agentes aos mesmos, são fatores que precisam ser controlados. Como por exemplo, ficou claro pelos experimentos que o algoritmo DE é bem adaptado para este tipo de problema contínuo, de modo que convergência da arquitetura se dá de forma muito rápida, e qualquer outro agente acaba gerando ruído no resultado final. Olhando para o experimento em que há variedade de configuração de agentes, parece ser necessário existir um mecanismo de adaptação entre os agentes, de modo que os agentes que geram soluções piores, e pioram o resultado global, sofram uma pressão seletiva. Isso seria útil, pois evitaria uma calibragem inicial dos parâmetros dos agentes.

Por fim, de modo geral, as regiões não demonstraram ser um fator relevante na maioria dos resultados apresentados. As regiões são o meio pelo qual que os agentes trocam informações e compreendem o espaço de busca. Esperava-se que a dinâmica do espaço de busca agrupasse as soluções semelhantes, e o agente aprendesse a explorar as regiões mais promissoras. Isso se daria por meio do mecanismo de memória utilizado, baseado no algoritmo *Q-learning*. Entretanto, o número de regiões no sistema não alcança um ponto de equilíbrio (vide Seção A.1) e a única exceção a esta observação foi o experimento descrito na Seção 6.5, onde aumentar o número de regiões parece fazer os agentes aprenderem a produzir soluções de melhor qualidade.

Capítulo 7

Conclusão

Vivendo, se aprende; mas o que se aprende, mais, é só a fazer outras maiores perguntas.

Riobaldo, em "Grande Sertão: Veredas"

Tendo sido apresentados os resultados dos experimentos acerca da diversidade no capítulo anterior, o presente capítulo retoma os objetivos iniciais deste trabalho, bem como a metodologia proposta. Inicialmente, o Capítulo 1 apresentou os fenômenos naturais que servem como inspiração tanto para a construção de sistemas multi-agentes como para metaheurísticas, introduzindo os conceitos de diversidade e colaboração. Após isso, são apresentados os elementos da literatura e problemas da vida prática que motivaram o desenvolvimento de novas metaheurísticas, bem como as limitações que essas estratégias tem em fornecer boas soluções em diferentes famílias de problemas. Este argumento é construído em torno do "*No Free Lunch Theorem*" para otimização (WOLPERT; MACREADY, 1997). Deste modo, é possível motivar e justificar a construção de um MAS, capaz de hibridizar de maneira dinâmica diferentes metaheurísticas, de modo que elas consigam colaborar e aprender características de um determinado problema. Dado um sistema como este, é razoável esperar que uma configuração com uma variedade de estratégias de busca produza bons resultados para um número maior de problemas, em comparação com configurações pouco diversas.

Na sequência, o Capítulo 2 visita a literatura de sistemas multi-agentes e *frameworks* voltados para otimização, apresentando em especial aquelas que possuem alguma característica distribuída. A maioria dos trabalhos encontrados na literatura são baseados em JADE (*Java Agent DEvelopment Framework*), que dá suporte a programação distribuída. Entretanto, nenhum desses trabalhos apresentam resultados da solução de problemas de larga escala em um *cluster*. O histórico da arquitetura D-Optimas é também apresentado neste capítulo, desde a sua primeira implementação, porposta por Junior (2010), até a sua última versão proposta por Pacheco (2017), tendo sido completamente remodelada sobre o *toolkit akka*.

Para permitir a execução da arquitetura em um número indeterminado de nós, algumas melhorias foram propostas no Capítulo 4. O principal objetivo dessas melhorias eram prover um mecanismo de balanceamento de carga, transparência de localidade para as entidades da simulação e tolerância a falhas. Essas melhorias envolveram uma mudança na ontologia dos agentes, regiões e atores supervisores da simulação, bem como uma mudança nos protocolos de comunicação. Além disso, um novo mecanismo de memória, novas meta-heurísticas foram adicionadas e os algoritmos de particionamento e fusão das regiões foi aprimorado, de modo que ele não mais dependa de nenhum conhecimento prévio dos problemas.

Para avaliar os efeitos da nova organização dos atores na simulação o Capítulo 5 descreve o primeiro experimento que comparou a média da latência de quatro tipos de mensagens na arquitetura, variando o número de nós no *cluster* de 3 à 6. A análise estatística permitiu dizer com 95% de confiança que houve diferença nas latências sempre que dois ou mais nós eram adicionados à simulação.

Finalmente, o Capítulo 6 apresentou os experimentos realizados para testar a principal hipótese deste trabalho, apresentada no Capítulo 3. Este experimento simulou a arquitetura com uma diversidade de estratégias de busca, para verificar se este fator contribui para a produção de soluções de melhor qualidade, em comparação a execuções com um único agente explorador do espaço de busca. Foi necessário, para isso, prover um conjunto inicial de *baseline*, com as configurações mais simples da arquitetura. Esta configuração básica foi sendo estendida a cada seção do capítulo, adicionando agentes de busca local, segmentando o espaço de busca, adicionando agentes do mesmo tipo com diferentes configurações e, finalmente, agentes de diferentes tipos.

7.1 Principais contribuições

Retomando o objetivo principal apresentado no início deste trabalho, a principal meta aqui posta era consolidar a arquitetura D-Optimas do ponto de vista de um sistema distribuído. Esperava-se aprimorar o sistema, dando a ele algum grau de tolerância a falhas, por meio de um mecanismo de balanceamento de carga e transparência de localidade. Este tipo de modificação daria à arquitetura a possibilidade de ser executada em um *cluster* com vários nós, escalando horizontalmente.

Aberta a possibilidade de escalar o sistema horizontalmente, o próximo passo seria executar simulações com problemas mais complexos, com uma variedade de agentes e estratégias de busca. Deste modo seria possível observar a adaptação da arquitetura ao problema, com os agentes colaborando e, dessa forma, verificar, neste comportamento, o surgimento de uma hibridização dinâmica das meta-heurísticas.

Para cumprir este objetivo geral, alguns objetivos específicos são listados no Capítulo 1. O primeiro deles foi compreender a arquitetura D-Optimas, seus fundamentos teóricos e seus mecanismos de generalização de problemas. Uma revisão da literatura em sistemas multi-agentes, principalmente do histórico da arquitetura D-Optimas, desde a sua precursora Bimasco, foi apresentado no Capítulo 2. O objetivo seguinte, que seria viabilizar a sua execução em um *cluster* com um número arbitrário de nós, foi cumprido uma vez que uma nova versão da arquitetura foi proposta neste trabalho, baseada na biblioteca *akka-cluster*. Foi proposta uma nova dinâmica para o comportamento das regiões e dos agentes no Capítulo 4, que não dependam de um conhecimento específico do problema de otimização. Na sequência, também foram adicionadas novas meta-heurísticas populacionais, a saber, o algoritmo de evolução diferencial e o algoritmo de otimização por enxame de partículas. Estes dois pontos também estavam presentes na lista inicial de objetivos.

Por fim, para estudar o comportamento da arquitetura, dois experimentos foram descritos no Capítulo 5 e Capítulo 6. O primeiro experimento realizado avaliou a escalabilidade da arquitetura em um **cluster** em relação ao aumento do número de nós na rede. O sistema se mostrou escalável para o problema escolhido, que foi a função *EggHolder*, problema clássico da literatura. A métrica escolhida para avaliar a escalabilidade foi a latência média das mensagens trocadas pela arquitetura com maior frequência. Os resultados deram bons indícios da resiliência da arquitetura como um sistema distribuído, mas para obter resultados mais robustos é necessário avaliar outros fatores, como por exemplo, a complexidade do problema de otimização, o número máximo de regiões permitidas, o número de agentes em execução e o crescimento do conjunto de soluções produzido pela arquitetura. É necessário também avaliar o funcionamento da arquitetura em um número maior de nós.

O segundo experimento proposto avaliou o efeito da diversidade na qualidade das soluções produzidas pela arquitetura. Este experimento foi dividido em 5 etapas. A primeira delas serviu para obter um conjunto de *baseline* de comparação, com a configuração mais simples possível da arquitetura. A segunda etapa avaliou a hibridização de um agente populacional e uma busca local em relação ao *baseline*. Nesta etapa foi possível observar uma melhora dos resultados quando hibridizada a busca local ILS com o algoritmo populacional DE. As demais combinações não demonstraram melhora estatística significativa. A terceira etapa avaliou o aumento do número de regiões para a configuração com um agente de busca local e um agente populacional. Neste caso, o aumento do número de regiões não fez diferença entre o experimento com uma única região para nenhuma das configurações propostas. A etapa seguinte avaliou a variação do número de agentes e suas respectivas configurações. Quando variado só o número de agentes, *i.e.* aumentando a quantidade de agentes populacionais e busca local mas mantendo as configurações, os resultados mostraram que a arquitetura tem um desempenho melhor para a configuração com 10

regiões, mas este resultado não é extensível para um número maior de regiões. Também não é extensível para agentes iguais com diferentes configurações, experimento que para algumas configurações do número de regiões se mostrou inclusive pior do que o *baseline*. A última etapa do procedimento experimental foi avaliar o desempenho da arquitetura com diferentes tipos de agentes em uma mesma simulação. Este resultado também não mostrou melhora em relação ao *baseline*.

Os resultados alcançados no experimento a cerca da diversidade não permitem afirmar que a arquitetura exibe um resultado colaborativo de melhor qualidade na presença de diversidade de agentes. Entretanto, estes resultados também foram preliminares no sentido de que avaliaram a arquitetura em um único problema de otimização. Além disto, os resultados revelam que há dois problemas na versão atual da arquitetura. O primeiro é em relação à organização das regiões. Esperava-se que o número de regiões no sistema se estabiliza a medida que novas soluções fossem produzidas, e que as regiões se organizassem de maneira a facilitar o aprendizado dos agentes a cerca das soluções mais interessantes. O observado é que as regiões raramente fazem fusão, e aumentam de maneira indiscriminada em direção ao limite de regiões do sistema. Aumentar o número de regiões também não surte efeito positivo para o agente, o que pode indicar um problema também na estratégia de seleção de ação implementada sobre a memória *Q-learning*.

De toda forma, a principal contribuição deste trabalho, para além da nova versão da arquitetura em um versão mais recente da biblioteca *akka*, foi um procedimento experimental para avaliar tanto a escalabilidade quanto o efeito da diversidade na qualidade das soluções. Os trabalhos futuros serão discutidos com mais detalhes na próxima seção, mas é já está claro que estes experimentos aqui apresentados devem ser repetidos num conjunto maior de problemas, e avaliando mais fatores. Neste sentido, este trabalho proveu um roteiro experimental, com os devidos métodos estatísticos, que pode ser repetido simplesmente trocando o problema de otimização. A versão da arquitetura aqui apresentada também é mais robusta, no que diz respeito aos princípios de sistemas distribuídos, comparada com a versão anterior. A versão atual não tem um ponto único de falha em um ator coordenador, está baseada em um sistema de banco de dados distribuído, e utiliza um mecanismo de balanceamento de carga que redistribui as regiões pelos nós a medida que estas crescem ou diminuem.

7.2 Trabalhos futuros

O presente trabalho evoluiu a arquitetura D-Optimas comparada a sua última versão, adicionando mais algoritmos de otimização, atualizando a implementação para a biblioteca *akka-cluster* e simplificando a sua execução em um *cluster*. Este trabalho também proveu todo o ferramental necessário para executá-la em diferentes infraestruturas, como o *HPC*

do CEFET-MG e a plataforma *Google Cloud*. Como os resultados não indicaram que as modificações no sistema de memória e dinâmica das regiões melhora de alguma forma o desempenho da arquitetura, estudar outras alternativas para a dinâmica das regiões, em especial a operação de fusão, parece um passo natural para um próximo trabalho. As regiões são a maneira que os agente segmentam e se comunicam dentro do espaço de busca, deste modo, um bom ajuste deste mecanismo é importante para o funcionamento da arquitetura.

Apesar das interações dos agentes com o mundo não ser determinística, num sentido de que não há uma ordem pré-estabelecida de qual agente vai trabalhar em qual conjunto de soluções, a sua interação com as regiões é limitada de modo que um agente só pode pedir soluções a uma região por vez. Adicionar mais comportamentos aos agentes traria a possibilidade de interações mais complexas com o ambiente, *e.g.* o agente poderia solicitar soluções de diversas regiões ao mesmo tempo baseado em sua memória passada, diversificando a população em que ele trabalharia.

Por fim, um passo importante neste projeto seria avaliar o desempenho da arquitetura em um número maior de problemas, principalmente em problemas de diferentes categorias, *e.g.* problemas de otimização combinatória, programação inteira, com várias restrições e multi-objetivos. Para isso, seria também interessante adicionar diferentes tipos de modificadores e estratégias dos algoritmos de busca local e populacional a fim de verificar a hipótese da diversidade em um conjunto maior de indivíduos.

Referências

- BARBUCHA, D. et al. Jabat middleware as a tool for solving optimization problems. In: **Transactions on computational collective intelligence II**. [S.l.]: Springer, 2010. p. 181–195. Citado na página 11.
- BINITHA, S.; SATHYA, S. S. et al. A survey of bio inspired optimization algorithms. **International journal of soft computing and engineering**, Citeseer, v. 2, n. 2, p. 137–151, 2012. Citado na página 3.
- BIRBIL, Ş. İ.; FANG, S.-C. An electromagnetism-like mechanism for global optimization. **Journal of global optimization**, Springer, v. 25, n. 3, p. 263–282, 2003. Citado na página 12.
- BONYADI, M. R.; MICHALEWICZ, Z. **Particle swarm optimization for single objective continuous space problems: a review**. [S.l.]: MIT Press, 2017. Citado na página 42.
- BURKE, E. et al. Hyper-heuristics: An emerging direction in modern search technology. In: **Handbook of metaheuristics**. [S.l.]: Springer, 2003. p. 457–474. Citado na página 4.
- CAMPOS, L. M. de A. **Modelagem do processo cognitivo-emocional de um organismo artificial numa perspectiva dinâmico-interacionista**. 2006. 145 f. Dissertação (Mestrado) — CEFET-MG, Belo Horizonte, 2006. Citado na página 14.
- CETNAROWICZ, K.; KISIEL-DOROHINICKI, M.; NAWARECKI, E. The application of evolution process in multi-agent world to the prediction system. In: **Proceedings of the Second International Conference on Multi-Agent Systems, ICMAS**. [S.l.: s.n.], 1996. v. 96, p. 26–32. Citado na página 11.
- DOKEROGLU, T. et al. A survey on new generation metaheuristic algorithms. **Computers & Industrial Engineering**, Elsevier, v. 137, p. 106040, 2019. Citado na página 4.
- DORRI, A.; KANHERE, S. S.; JURDAK, R. Multi-agent systems: A survey. **Ieee Access**, IEEE, v. 6, p. 28573–28593, 2018. Citado na página 3.
- EIBEN, A. E.; SMITH, J. E. **Introduction to evolutionary computing**. [S.l.]: Springer, 2015. Citado na página 7.
- FERNANDES, F. C. et al. A multiagent architecture for solving combinatorial optimization problems through metaheuristics. In: IEEE. **Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on**. [S.l.], 2009. p. 3071–3076. Citado 2 vezes nas páginas 5 e 13.
- GASPAR-CUNHA, A.; TAKAHASHI, R.; ANTUNES, C. H. **Manual de computação evolutiva e metaheurística**. [S.l.]: Imprensa da Universidade de Coimbra/Coimbra University Press, 2012. Citado 2 vezes nas páginas 42 e 44.

GONG, Y.-J. et al. Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. **Applied Soft Computing**, Elsevier, v. 34, p. 286–300, 2015. Citado na página 5.

HARARI, Y. N. **Sapiens: Uma Breve História da Humanidade**. [S.l.]: L&PM, 2015. Citado na página 2.

HEWITT, C. What is computation? actor model versus turing’s model. In: **A computable universe: understanding and exploring nature as computation**. [S.l.]: World Scientific, 2013. p. 159–185. Citado na página 5.

HEWITT, C.; BISHOP, P.; STEIGER, R. A universal modular actor formalism for artificial intelligence. In: **Proceedings of the 3rd International Joint Conference on Artificial Intelligence**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1973. (IJCAI’73), p. 235–245. Citado na página 19.

HOLLAND, J. H. **Complexity: A very short introduction**. [S.l.]: OUP Oxford, 2014. Citado na página 3.

JUAN, A. A. et al. The sr-gcws hybrid algorithm for solving the capacitated vehicle routing problem. **Applied Soft Computing**, Elsevier, v. 10, n. 1, p. 215–224, 2010. Citado na página 12.

JUNIOR, E. S. **Sistema Multiagente Bioinspirado para Otimização Combinatória**. Dissertação (Mestrado) — Centro Federal de Educação Tecnológica de Minas Gerais, Belo Horizonte, 8 2010. Citado 7 vezes nas páginas 5, 13, 14, 15, 16, 17 e 76.

JDRZEJOWICZ, P.; WIERZBOWSKA, I. Jade-based a-team environment. In: SPRINGER. **International Conference on Computational Science**. [S.l.], 2006. p. 719–726. Citado na página 11.

KARP, R. M. Reducibility among combinatorial problems. In: **Complexity of computer computations**. [S.l.]: Springer, 1972. p. 85–103. Citado na página 6.

Kennedy, J.; Eberhart, R. Particle swarm optimization. In: **Proceedings of ICNN’95 - International Conference on Neural Networks**. [S.l.: s.n.], 1995. v. 4, p. 1942–1948 vol.4. Citado na página 42.

KERÇELLI, L. et al. Mango: A multiagent environment for global optimization. In: **Proceedings of the 7th Conference on Autonomous and Multi-agent Systems**. [S.l.: s.n.], 2008. Citado na página 12.

KISIEL-DOROHINICKI, M. Agent-based models and platforms for parallel evolutionary algorithms. In: SPRINGER. **International Conference on Computational Science**. [S.l.], 2004. p. 646–653. Citado na página 11.

LI, Y. **A New Exact Algorithm for Traveling Salesman Problem with Time Complexity Interval ($O(n^4)$, $O(n^3 * 2^n)$)**. 2015. *Citadonapágina6*.

MALEK, R. Collaboration of metaheuristic algorithms through a multi-agent system. In: SPRINGER. **International Conference on Industrial Applications of Holonic and Multi-Agent Systems**. [S.l.], 2009. p. 72–81. Citado na página 11.

- MALEK, R. An agent-based hyper-heuristic approach to combinatorial optimization problems. In: IEEE. **2010 IEEE International Conference on Intelligent Computing and Intelligent Systems**. [S.l.], 2010. v. 3, p. 428–434. Citado na página 11.
- MAPA, S. **Modelagem de Organismos Artificiais Cognitivo-Emocionais Dotados de Memória Experimental de Longo Prazo**. 2009. 169 f. Dissertação (Mestrado) — CEFET-MG, Belo Horizonte, 2009. Citado na página 14.
- MARCOLINO, L. S.; JIANG, A. X.; TAMBE, M. Multi-agent team formation: Diversity beats strength? In: **Twenty-Third International Joint Conference on Artificial Intelligence**. [S.l.: s.n.], 2013. Citado na página 8.
- MARTIN, S. et al. A multi-agent based cooperative approach to scheduling and routing. **European Journal of Operational Research**, Elsevier, v. 254, n. 1, p. 169–178, 2016. Citado na página 12.
- MATURANA, H.; VARELA, F. A árvore do conhecimento. **São Paulo: Palas Athena**, v. 2, 2001. Citado 2 vezes nas páginas 1 e 2.
- MILANO, M.; ROLI, A. Magma: a multiagent architecture for metaheuristics. **IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)**, IEEE, v. 34, n. 2, p. 925–941, 2004. Citado na página 5.
- NORVIG, R. . **ARTIFICIAL INTELLIGENCE: A MODERN APPROACH, 3/e**. [S.l.]: Pearson Education, 2018. ISBN 0136042597. Citado na página 39.
- OLIVEIRA, R. B. de. **Desenvolvimento de uma arquitetura multiagentes baseada em meta-heurísticas com um abordagem adaptative learning search**. Dissertação (Mestrado) — Centro Federal de Educação Tecnológica de Minas Gerais, Belo Horizonte, 10 2008. Citado na página 13.
- PACHECO, R. A. **D-OPTIMAS: Um Sistema de Otimização Multiagentes Distribuído Baseado no Modelo de Atores**. Dissertação (Mestrado) — Centro Federal de Educação Tecnológica de Minas Gerais, Belo Horizonte, 1 2017. Citado 14 vezes nas páginas 5, 7, 13, 19, 20, 22, 23, 24, 26, 31, 32, 37, 46 e 76.
- PÉREZ, A. A. J. et al. A simulation-based approach for solving the flowshop problem. In: **Proceedings of the 2010 winter simulation conference**. [S.l.: s.n.], 2010. p. 3384–3395. Citado na página 12.
- PIĘTAK, K.; KISIEL-DOROHINICKI, M. Agent-based framework facilitating component-based implementation of distributed computational intelligence systems. In: **Transactions on Computational Collective Intelligence X**. [S.l.]: Springer, 2013. p. 31–44. Citado na página 11.
- PIĘTAK, K. et al. Functional integrity of multi-agent computational system supported by component-based implementation. In: SPRINGER. **International Conference on Industrial Applications of Holonic and Multi-Agent Systems**. [S.l.], 2009. p. 82–91. Citado na página 11.
- SANTOS, B. A. **Aspectos Conceituais e Arquiteturais para a Criação de Linhagens de Agentes de Software Cognitivos e Situados**. Junho 2003. 130 f. Dissertação (Mestrado) — CEFET-MG, Belo Horizonte, 2003. Citado na página 14.

SANTOS, M. R. A. dos. **Efeitos da Cooperação em um Sistema Multiagentes Autônomos Para a Solução de Problemas de Otimização**. Dissertação (Mestrado) — Centro Federal de Educação Tecnológica de Minas Gerais, Belo Horizonte, 3 2015. Citado 9 vezes nas páginas 5, 7, 13, 18, 22, 23, 24, 26 e 28.

SHAPIRO, M. et al. Conflict-free replicated data types. In: SPRINGER. **Symposium on Self-Stabilizing Systems**. [S.l.], 2011. p. 386–400. Citado na página 34.

Shi, Y.; Eberhart, R. A modified particle swarm optimizer. In: **1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)**. [S.l.: s.n.], 1998. p. 69–73. Citado na página 42.

SILVA, M. A. et al. Amam: arquitetura multiagente para a solução, via metaheurísticas, de problemas de otimização combinatória. In: **8th Symposium on Intelligent Automation**. [S.l.: s.n.], 2007. Citado na página 13.

SILVA, M. A. L. **AMAM: Framework Multiagente Para Otimização Usando Metaheurísticas**. Tese (Doutorado) — Centro Federal de Educação Tecnológica de Minas Gerais, Belo Horizonte, 2019. Citado na página 20.

SILVA, M. A. L. et al. A multi-agent metaheuristic optimization framework with cooperation. In: IEEE. **2015 Brazilian Conference on Intelligent Systems (BRACIS)**. [S.l.], 2015. p. 104–109. Citado na página 13.

SILVA, M. A. L. et al. Hybrid metaheuristics and multi-agent systems for solving optimization problems: A review of frameworks and a comparative analysis. **Applied Soft Computing**, Elsevier, v. 71, p. 433–459, 2018. Citado 4 vezes nas páginas 5, 10, 14 e 20.

SILVA, V. A. **Modelagem e Desenvolvimento de um Mecanismo de Condicionamento para a Arquitetura Artífice**. 2008. 120 f. Dissertação (Mestrado) — CEFET-MG, 2008. Citado na página 14.

SOLOMON, M. M. Algorithms for the vehicle routing and scheduling problems with time window constraints. **Operations research**, *Inform*s, v. 35, n. 2, p. 254–265, 1987. Citado na página 17.

SOUZA, D. de. **Generalização de Agentes de Software Metaheurísticos na Arquitetura BIMASCO**. Dissertação (Mestrado) — Centro Federal de Educação Tecnológica de Minas Gerais, Belo Horizonte, 3 2014. Citado 8 vezes nas páginas 5, 13, 17, 18, 31, 41, 42 e 43.

STORN, R.; PRICE, K. Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. **Journal of global optimization**, Springer, v. 11, n. 4, p. 341–359, 1997. Citado na página 42.

STÜTZLE, T.; LÓPEZ-IBÁÑEZ, M. Automated design of metaheuristic algorithms. In: **Handbook of Metaheuristics**. [S.l.]: Springer, 2018. p. 541–579. Citado na página 4.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction**. [S.l.]: MIT press, 2018. Citado na página 39.

TANENBAUM, A. S.; STEEN, M. V. **Distributed systems: principles and paradigms**. [S.l.]: Prentice-Hall, 2007. Citado 3 vezes nas páginas 31, 32 e 36.

WHITLEY, D. A genetic algorithm tutorial. **Statistics and computing**, Springer, v. 4, n. 2, p. 65–85, 1994. Citado na página 7.

WOLPERT, D. H.; MACREADY, W. G. No free lunch theorems for optimization. **IEEE transactions on evolutionary computation**, IEEE, v. 1, n. 1, p. 67–82, 1997. Citado 3 vezes nas páginas 4, 7 e 76.

XU, M. et al. An improved dijkstra’s shortest path algorithm for sparse network. **Applied Mathematics and Computation**, v. 185, n. 1, p. 247 – 254, 2007. ISSN 0096-3003. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0096300306008563>>. Citado na página 6.

ZHENG, X.-l.; WANG, L. A multi-agent optimization algorithm for resource constrained project scheduling problem. **Expert Systems with Applications**, Elsevier, v. 42, n. 15, p. 6039–6049, 2015. Citado na página 5.

Apêndices

APÊNDICE A – Avaliação do efeito da diversidade

Este capítulo contém gráficos auxiliares utilizados no desenvolvimento do experimento de avaliação do efeito da diversidade. A Seção A.1 exibe os gráficos de número médio de regiões ao longo do tempo para cada um dos experimentos avaliados.

A.1 Número médio de regiões

Figura 27 – Número médio de regiões em função do tempo para o experimento GRASP_ILS_GA_5R

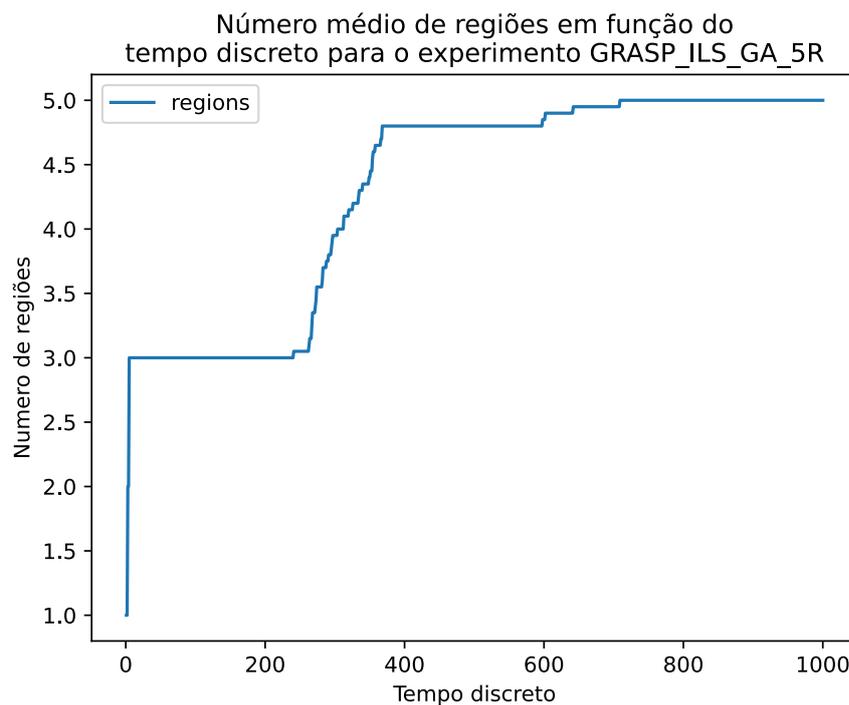


Figura 28 – Número médio de regiões em função do tempo para o experimento GRASP_ILS_GA_10R

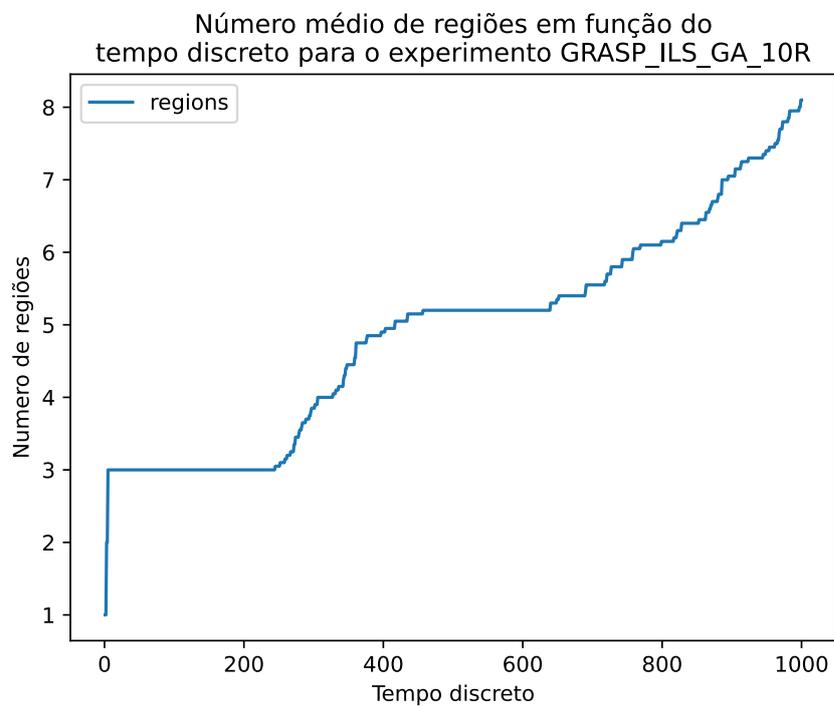


Figura 29 – Número médio de regiões em função do tempo para o experimento GRASP_ILS_GA_20R

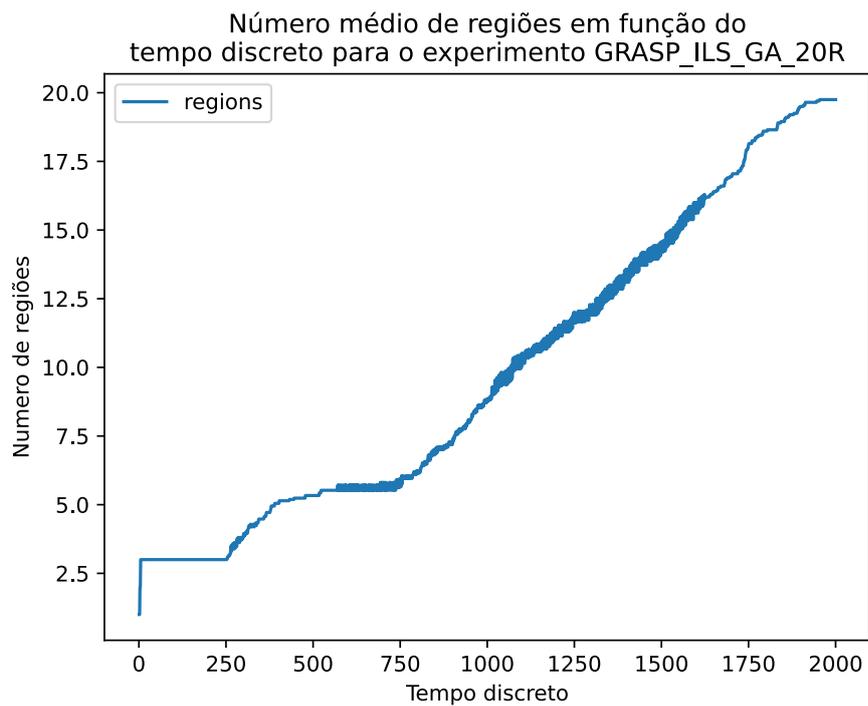


Figura 30 – Número médio de regiões em função do tempo para o experimento GRASP_ILS_PSO_5R

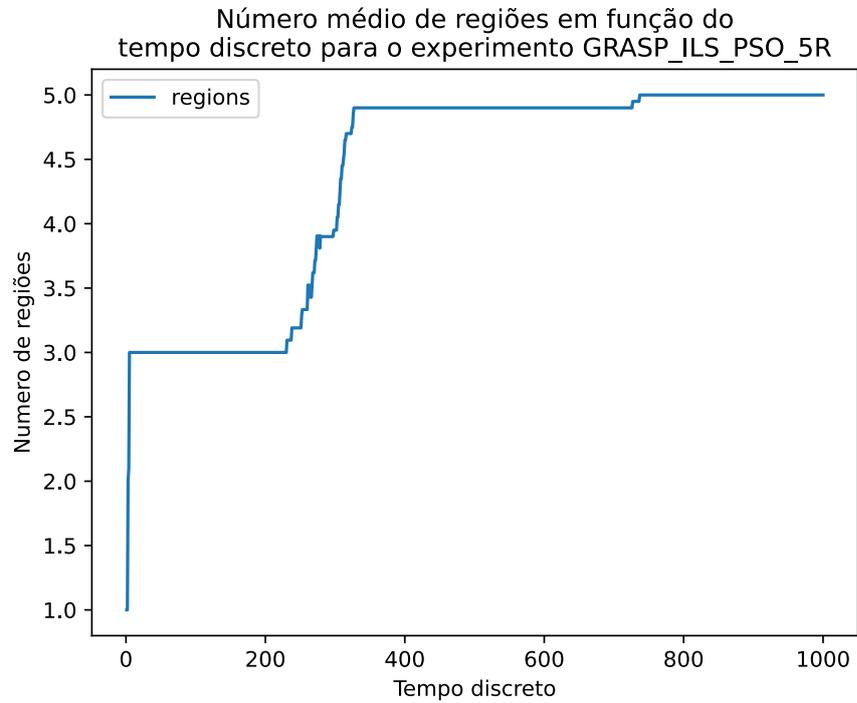


Figura 31 – Número médio de regiões em função do tempo para o experimento GRASP_ILS_PSO_10R

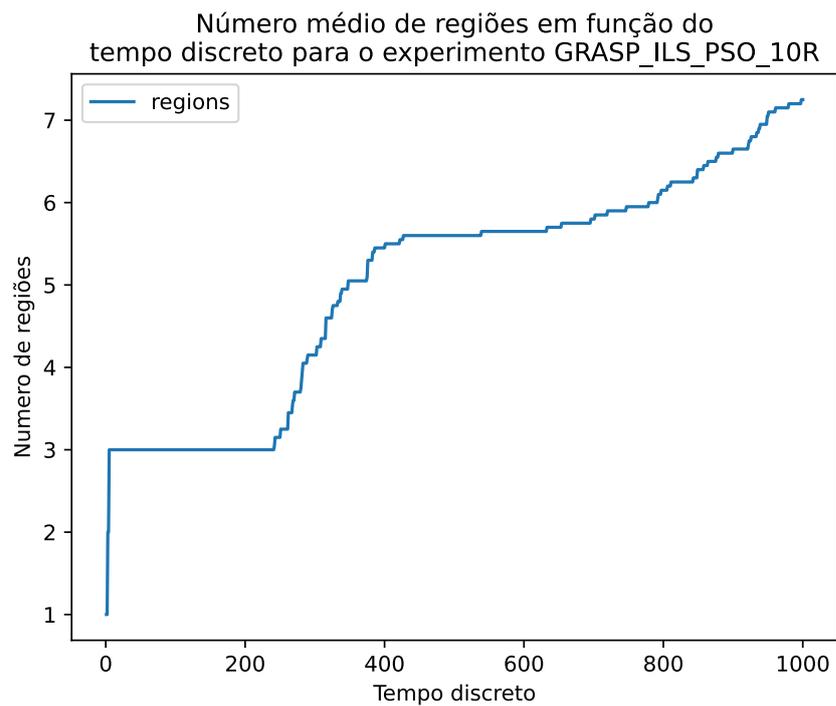


Figura 32 – Número médio de regiões em função do tempo para o experimento GRASP_ILS_PSO_20R

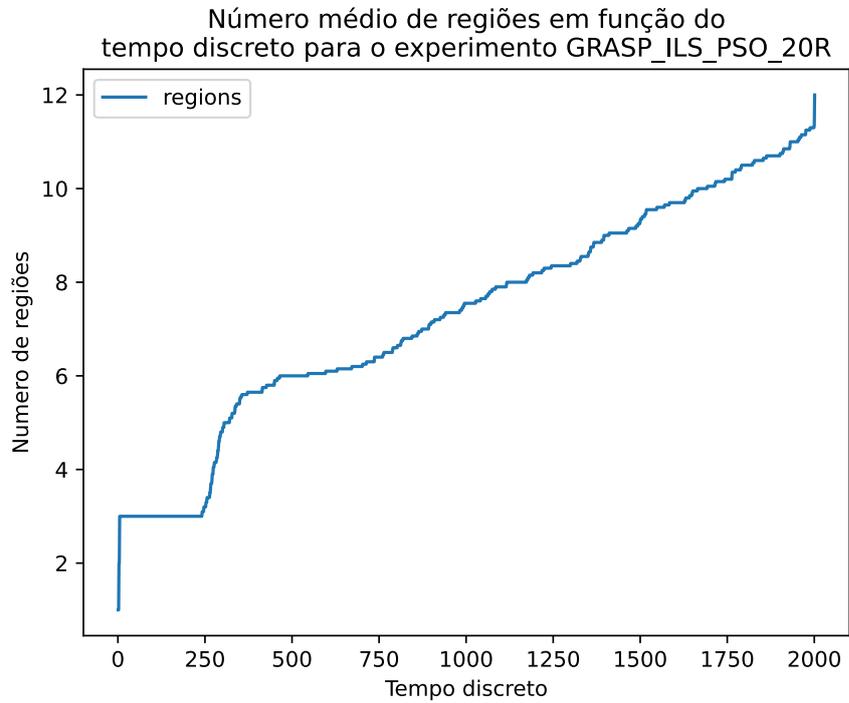


Figura 33 – Número médio de regiões em função do tempo para o experimento GRASP_ILS_DE_5R

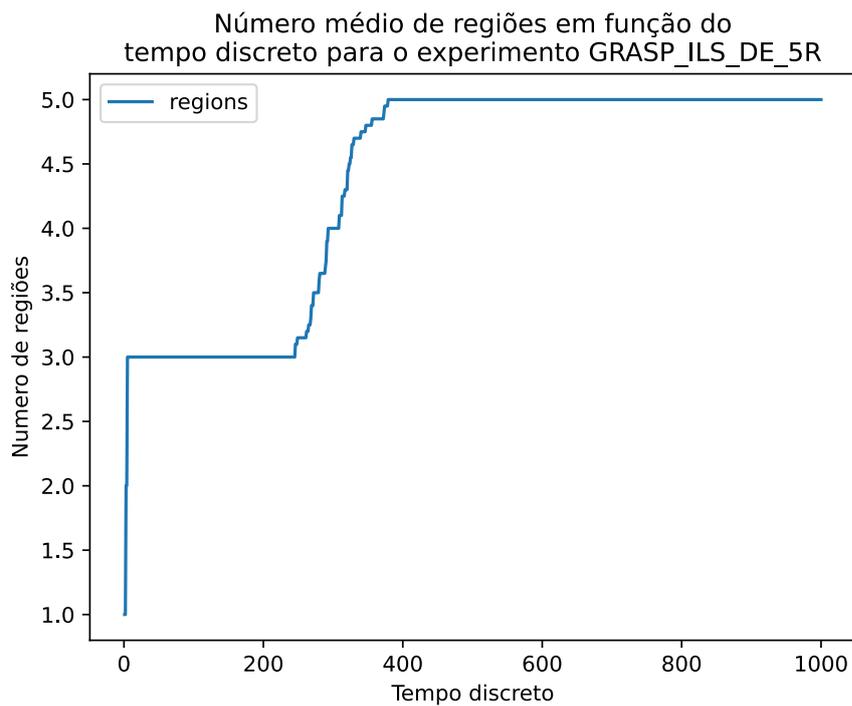


Figura 34 – Número médio de regiões em função do tempo para o experimento GRASP_ILS_DE_10R

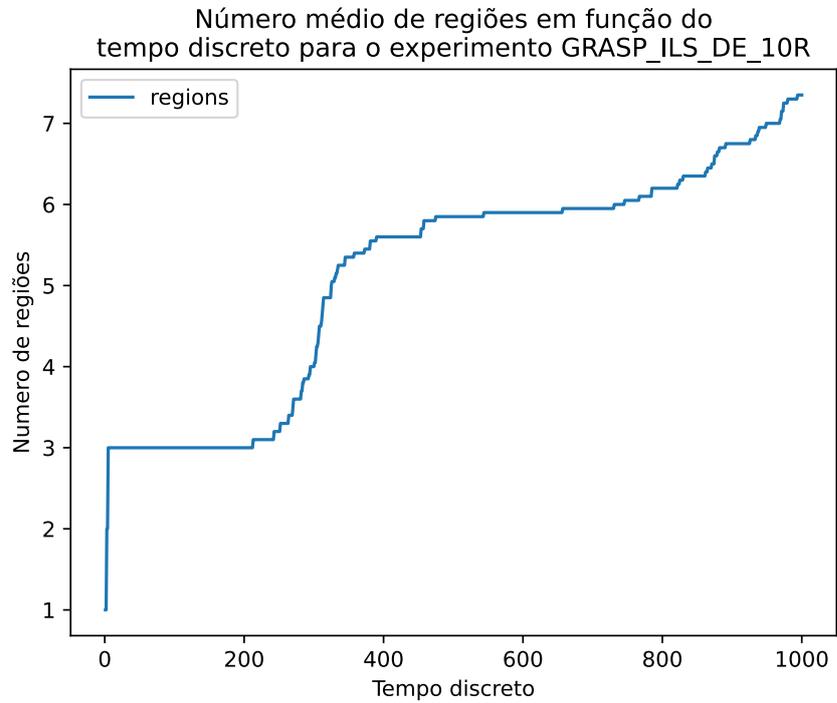


Figura 35 – Número médio de regiões em função do tempo para o experimento GRASP_ILS_DE_20R

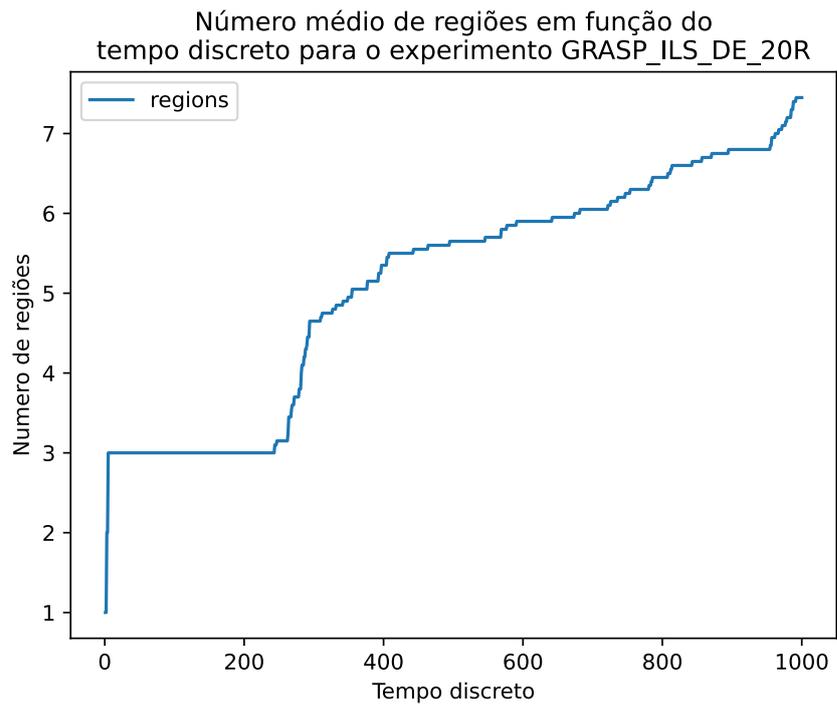


Figura 36 – Número médio de regiões em função do tempo para o experimento GRASP_2ILS_4GA_5R

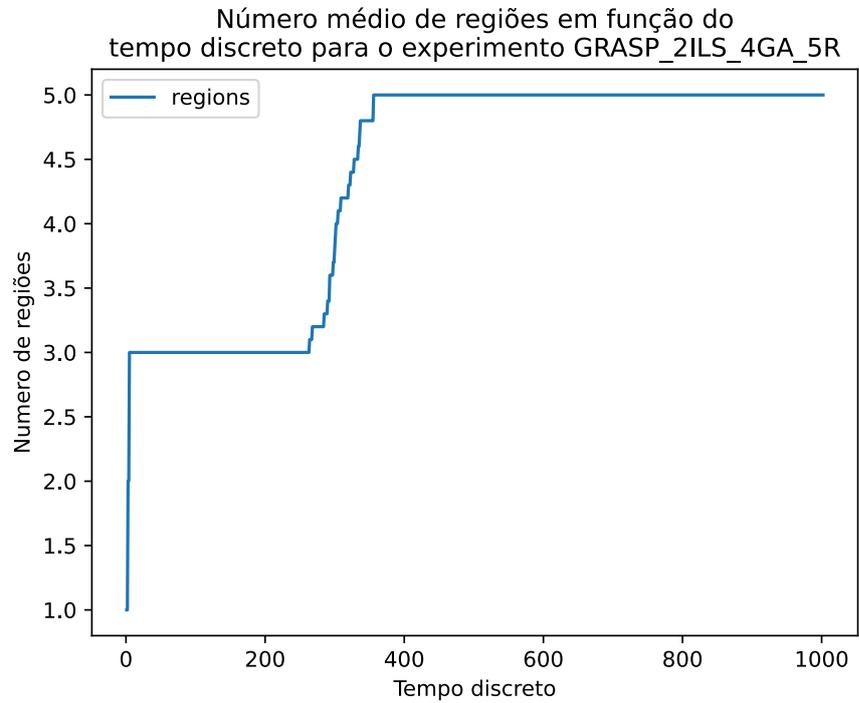


Figura 37 – Número médio de regiões em função do tempo para o experimento GRASP_2ILS_4GA_10R

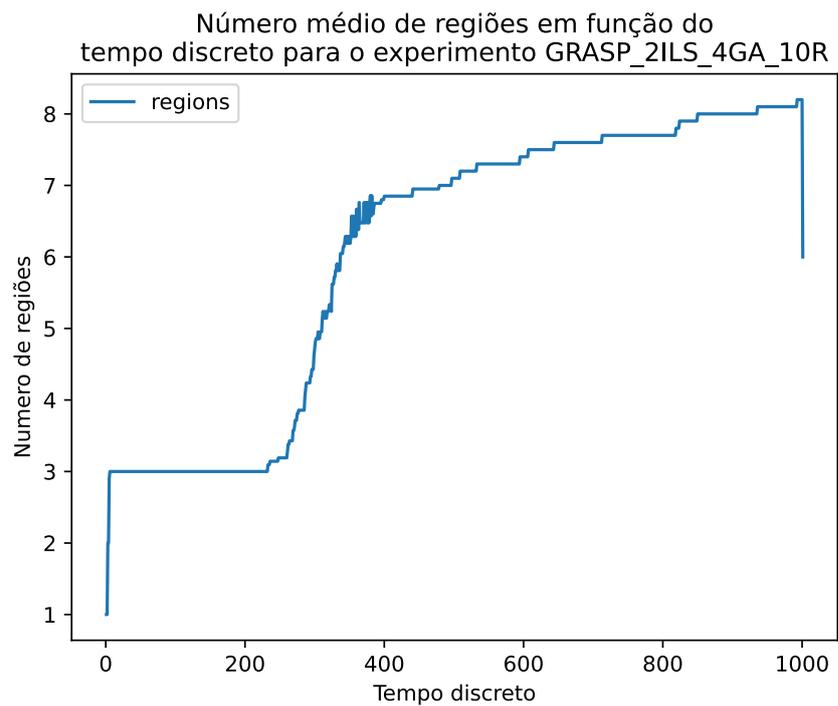


Figura 38 – Número médio de regiões em função do tempo para o experimento GRASP_2ILS_4GA_20R

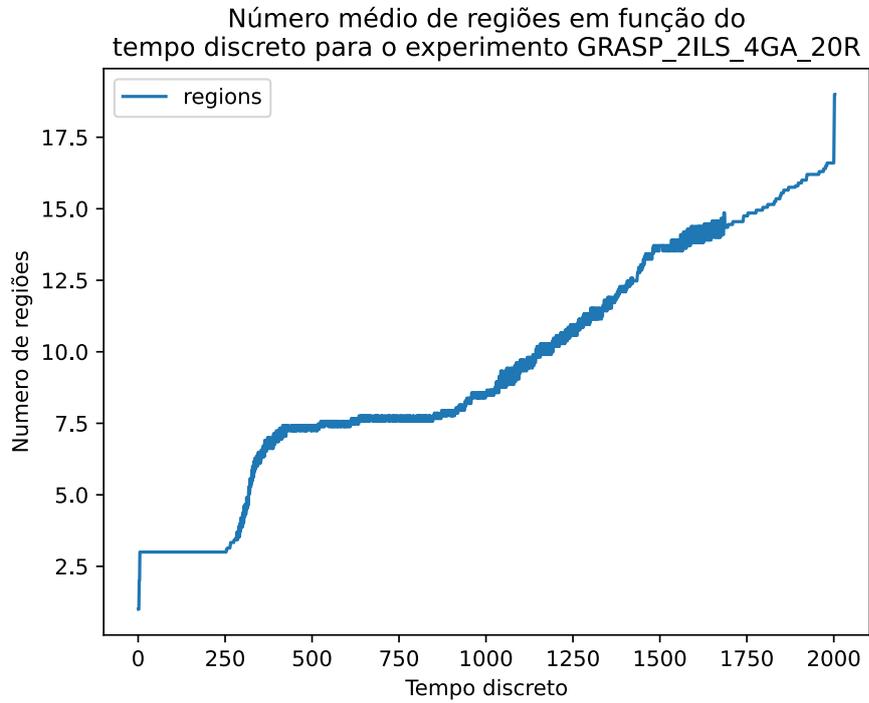


Figura 39 – Número médio de regiões em função do tempo para o experimento GRASP_2ILS_4GA_VP_5R

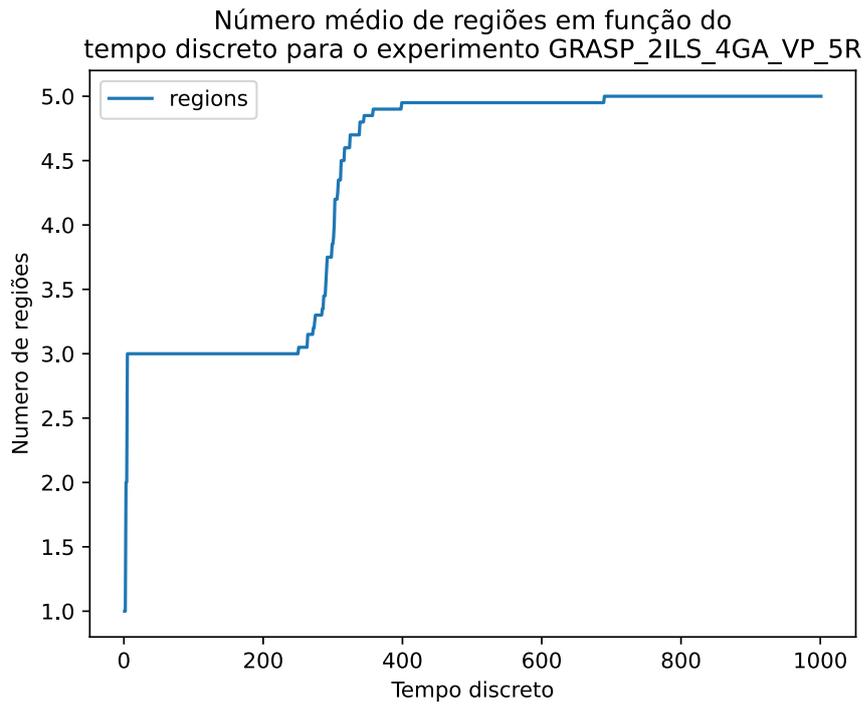


Figura 40 – Número médio de regiões em função do tempo para o experimento GRASP_2IL§₄GA_VP_10R

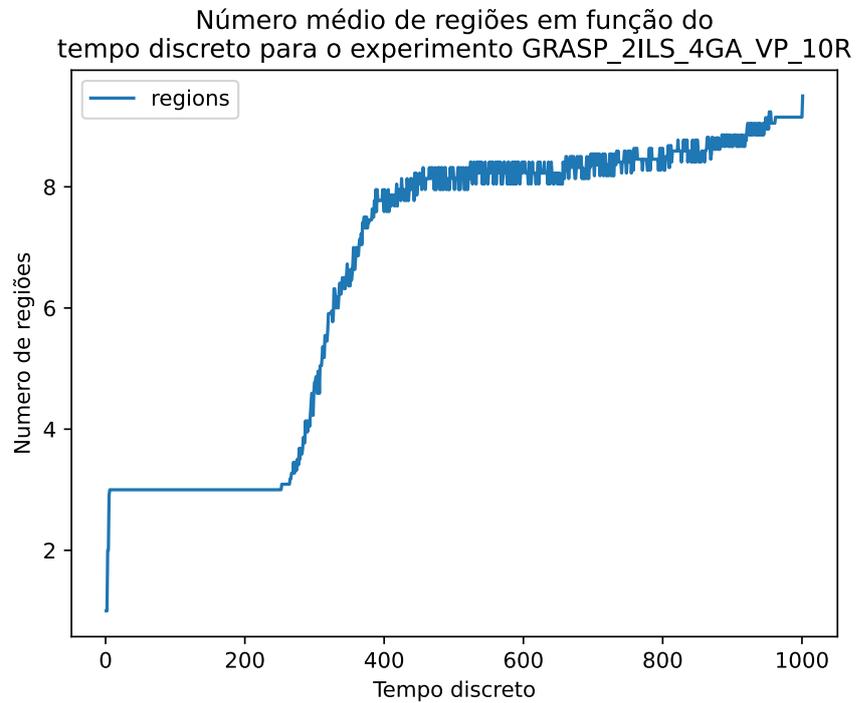


Figura 41 – Número médio de regiões em função do tempo para o experimento GRASP_2IL§₄GA_VP_20R

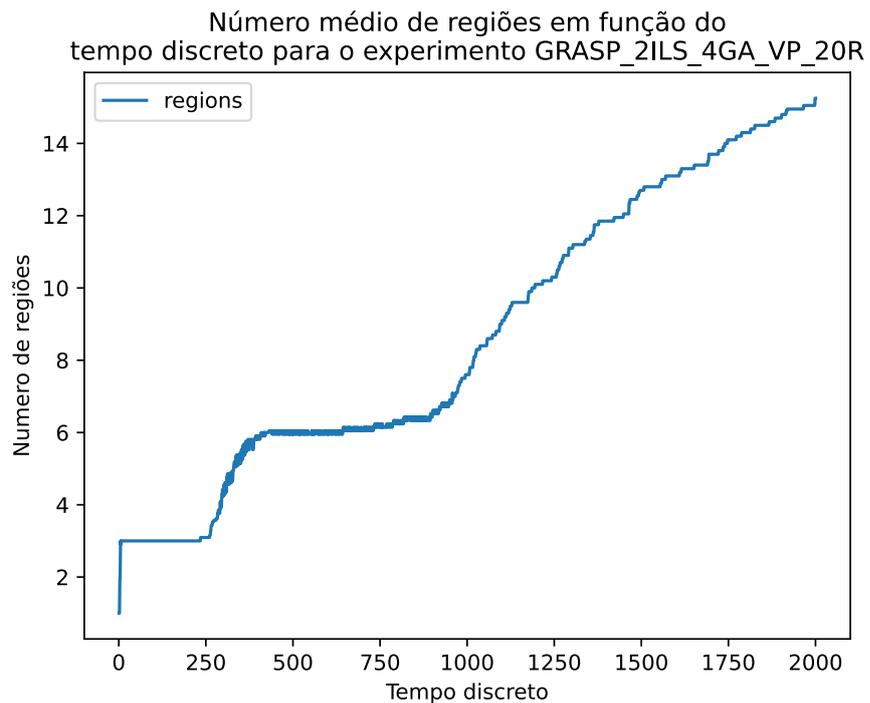


Figura 42 – Número médio de regiões em função do tempo para o experimento GRASP_2ILS_2GA_DE_PSO_5R

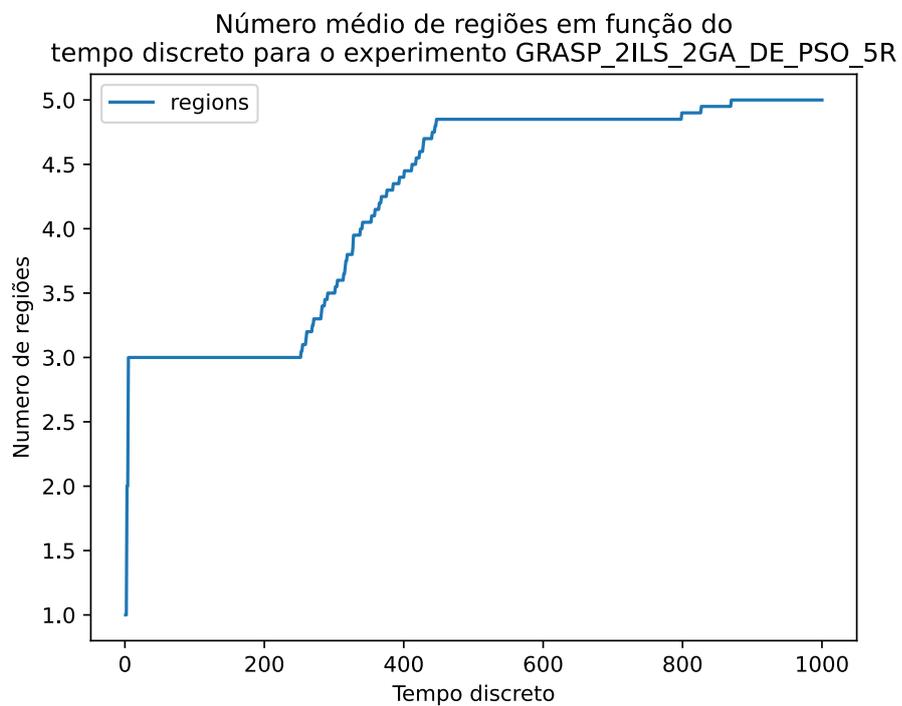


Figura 43 – Número médio de regiões em função do tempo para o experimento GRASP_2ILS_2GA_DE_PSO_10R

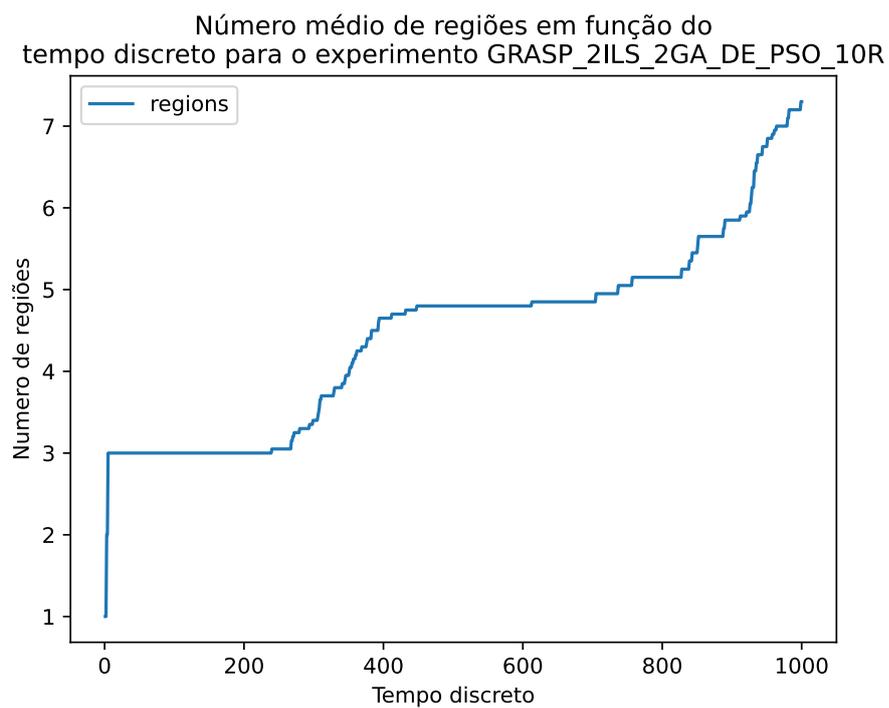


Figura 44 – Número médio de regiões em função do tempo para o experimento GRASP_2ILS_2GA_DE_PSO_20R

