Algoritmos Evolutivos Eficientes para um Problema de Roteamento de Veículo

Fábio Linhares Dalboni

Dissertação apresentada ao Curso de Pós-Graduação em Computação Aplicada e Automação da Universidade Federal Fluminense como parte dos requisitos necessários à obtenção do título de Mestre em Computação Aplicada e Automação.

Orientadores: Luiz Satoru Ochi
Lúcia Maria de Assumpção Drummond

AGOSTO DE 2003

Algoritmos Evolutivos Eficientes para um Problema de Roteamento de Veículo

Fábio Linhares Dalboni

Dissertação apresentada ao Curso de Pós-Graduação em Computação Aplicada e Automação da Universidade Federal Fluminense, no dia 06/08/2003, como parte dos requisitos necessários à obtenção do título de Mestre em Computação Aplicada e Automação, tendo sido aprovada pela Banca Examinadora, da qual participaram os seguintes professores:

_____

Luiz Satoru Ochi (Orientador), IC/UFF

_____

Lúcia Maria de Assumpção Drummond (Orientadora), IC/UFF

_____

Valmir Carneiro Barbosa, COPPE SISTEMAS/UFRJ

_____

Paulo Morelato França, DENSIS/FEE/UNICAMP

"O homem que decide parar até que as coisas melhorem, verificará mais tarde que aquele que não parou e colaborou com o tempo, estará tão adiante que jamais será alcançado."

Rui Barbosa

Aos meus familiares

e grandes amigos...

Agradecimentos

São muitos os agradecimentos a fazer. Neste período de quase dois anos e meio (03/01 - 08/03), em que me ingressei no mestrado, passei por muitas dificuldades que graças a Deus consegui superar com relativa tranqüilidade, mas com muito esforço.

Inicio meus agradecimentos primeiramente a Deus, por ter me ajudado a ingressar em um Mestrado de conceituado renome e por me ajudar nos momentos em que precisei tomar decisões difíceis na vida como, por exemplo, optar por continuar o mestrado ou manter o meu emprego. Deus, sempre esteve presente me fornecendo afortunadas "portas", que em vez de obstáculos, constituíam diferentes passagens. Não seria fácil passar por elas, mas uma a uma foram vencidas as dificuldades. A vida me enriqueceu com seus segredos e os erros foram transformados em acertos.

Na UFF, são muitas as pessoas a quem gostaria de agradecer: Ao meu "Pai do Rio", Mestre Inhaúma Neves Ferraz, cuja ajuda e cujos conselhos me levaram a alcançar este estágio e a sentir a alegria que estão acumulados em mim neste momento; aos meus Orientadores Dr. Luiz Satoru Ochi e Drª. Lúcia Maria Assumpção Drummond, com quem aprendi muito em relação a área de otimização e paralelismo, e que têm acompanhado minha evolução desde a graduação. À Dulcinea, secretária do departamento de Ciência da Computação, que através de conversas, antes das aulas, conseguia me deixar menos nervoso; ao funcionário Carlinhos, pelo pronto atendimento em qualquer situação.

Durante o mestrado, conquistei amizades importantes, como o Luís Alberto Alves Buenes e a Cristiane Norbiato Targa, que muito contribuíram para o desenvolvimento da minha dissertação. Luís foi responsável por decifrar junto comigo os "bugs" incríveis que apareciam durante a programação. Cris, foi responsável por me emprestar o algoritmo Apriori que ela utilizou na defesa de sua dissertação recentemente.

À minha namorada, Carla Moreira Martins de Barros, que teve uma paciência do tamanho do mundo, para agüentar minhas reclamações, meus desabafos e por ajudar na leitura de alguns artigos.

À minha família: a família Linhares Dalboni. Meu pai Francisco de Assis Dalboni Cunha, um homem vencedor, de quem tenho muito orgulho em falar e que soube fazer de seus sonhos as suas realizações com muita luta. À minha mãe, Valderice Linhares Dalboni, que me ajudou nas revisões desta dissertação, que me apoiou e que ao lado de meu pai conseguiu construir uma família unida pela lealdade, respeito e amor. Às minhas irmãs, Rejane Linhares Dalboni e Karen Linhares Dalboni, que estão lutando para conquistar os seus lugares no mercado de trabalho.

Aos meus Avós, Raimundo Cirilo Quaresma e Dirce Linhares Quaresma, pela acolhida nos finais de semana e pelo apoio.

Ao meu tio Dr. Ettore Dalboni da Cunha, Juiz de Direito, que sempre me incentivou na carreira e a quem estou devendo uma cópia desta dissertação.

À Tamara Kinupp responsável por me ajudar na diagramação visual dos slides empregados no dia da defesa e ao Glaucio Alves de Souza pelo apoio na reta final.

À Mestra Rita de Cássia Almeida Pontes, minha "Mãe do Rio", ex-chefe, que forneceu uma ajuda incrível e dicas fantásticas com relação ao Mestrado.

Aos meus amigos Cezar Alexandre Rodrigues Baldi, Robson Vinício Santos Guimarães e Frederico Chicralla Nunes, que durante esse tempo, entenderam a minha ausência nas altas badaladas nas noites de Niterói e que ficaram torcendo para que voltássemos a sair e a nos divertirmos.

A minha amiga estrangeira, Maria Carolina Regino Carneiro, quem me ajudou a traduzir o *abstract* desta dissertação.

Aos meus tios Valfredo Linhares Quaresma (*in memoriam*) e Raimundo Linhares Quaresma (*in memoriam*), que tenho certeza de que estão ao meu lado, me conduzindo a novas conquistas. Que Deus os abençoe!

Esta dissertação, eu divido com todos vocês: ela é conquista de todos nós.

Resumo

Este trabalho apresenta propostas que visam melhorar o desempenho de algoritmos evolutivos (AEs). Os AEs e em particular, os algoritmos genéticos (AGs), apesar de muito conhecidos, não têm alcançado resultados competitivos com suas versões básicas (AGB) na solução de diferentes problemas de otimização.

Diferentes variações dos AGBs têm sido propostas na literatura, com o intuito de tentar reduzir algumas das limitações presentes nestes algoritmos, tais como: dificuldade em efetuar uma busca local de forma eficiente; e a exigência de um tempo computacional maior do que o exigido por outras metaheurísticas.

Neste trabalho, propomos alternativas para melhorar o desempenho destes AGBs, que aborda: a substituição dos operadores básicos de reprodução por heurísticas no algoritmo chamado de AG; a inclusão no AG de módulos de busca local que chamaremos AG+BL; e finalmente a inclusão neste último algoritmo de módulos de Mineração de Dados que chamaremos de AG+BL+MD.

Numa segunda etapa, passaremos a analisar o problema da redução dos tempos computacionais exigidos por um AG.

Procuramos adequar as versões aqui propostas a um Problema de Roteamento de Veículos (PRV), onde buscamos encontrar um percurso otimizado para uma unidade móvel de pistoneio (UMP), de modo a maximizar a extração do petróleo dos poços terrestres *não surgentes*, respeitando as restrições do problema.

Resultados computacionais mostram o bom desempenho dos algoritmos propostos neste trabalho, e em particular, a da versão com mineração de dados.

Abstract

This work proposes solutions for enhancing the performance of evolutionary algorithms (EAs). The EAs and, in particular, the genetic algorithms (GAs), although very well known, are not reaching comparably better results against their basic versions (BGA) when solving different hard optimization problems.

Different variations of BGAs are being researched and presented, with the goal of reducing some of their limitations, such as: the difficulty on executing an efficient local search and the demand for more computing time than other metaheuristics.

We propose alternatives to enhance the performance of these BGAs, which can be described as: replacement of the basic reproduction operators by heuristics called GA, inclusion of local search procedures, called GA+LS, and also, the inclusion of procedures of Data Mining, called GA+LS+DM.

In a second phase of this research, we will analyze the problem of reducing the computation time of a GA.

In this work, four versions of evolutionary algorithms are proposed to solve the Vehicle Routing Problem, where we try to find an optimized route for a Oil Collector Vehicle, to help maximize oil extraction from surgent oil fields, thus, considering the constraints of the problem.

The results of this computational investigation showed that the proposed algorithms of this research outperformed the other basic versions of GAs, specially the algorithm with procedures of Data Mining.

Sumário

CAPÍTULO 1 - INTRODUÇÃO 18

CAPÍTULO 2 - O PROBLEMA DE ROTEAMENTO DE UMA UNIDADE MÓVEL DE PISTONEIO (RUMP) 20

2.1 MODELO MATEMÁTICO 22
2.2 FORMULAÇÃO MATEMÁTICA 23

CAPÍTULO 3 - ALGORITMO GENÉTICO 28

3.1 INTRODUÇÃO 28
3.2 COMPONENTES DE UM AGB 33
3.2.1 Representação Genética de Soluções Viáveis 33
3.2.2 População Inicial 34
3.2.3 Função de Avaliação 34
3.2.4 Reprodução 34
3.2.5 Parâmetros do AG 36
3.3 ALGORITMOS GENÉTICOS PARALELOS 37

CAPÍTULO 4 - MINERAÇÃO DE DADOS 40

4.1 REGRAS DE ASSOCIAÇÃO 40

CAPÍTULO 5 - ALGORITMOS PROPOSTOS 46

5.1 ALGORITMOS SEQÜENCIAIS 47
5.1.1 ALGORITMO 1 –AGB 47
5.1.2 ALGORITMO 2 –AG 48
5.1.3 ALGORITMO 3 – AG + BL 50
5.1.4 ALGORITMO 4 – AG + BL + MD 50
5.2 ALGORITMOS PARALELOS 52
5.2.1 Algoritmo 1 - AGP1 (AGB Paralelo sem Comunicação) 52
5.2.2 Algoritmo 2 – AGP2 (AGB Paralelo com Comunicação) 53
5.2.3 Algoritmo 3 - AGP3 (AG Paralelo sem Comunicação) 53
5.2.4 Algoritmo 4 - AGP4 (AG Paralelo com Comunicação) 53
5.2.5 Algoritmo 5 - AGP5 (AG + BL Paralelo sem Comunicação) 54
5.2.6 Algoritmo 6 – AGP6 (AG + BL Paralelo com Comunicação) 54
5.2.7 Algoritmo 7– AGP7 (AG+BL+MD Paralelo sem Comunicação) 54
5.2.8 Algoritmo 8– AGP8 (AG+BL+MD Paralelo com Comunicação) 54
5.3 EXEMPLO DE EXECUÇÃO 55

CAPÍTULO 6 - RESULTADOS COMPUTACIONAIS 58

6.1 RESULTADOS DE ALGORITMOS SEQÜENCIAIS 60
6.2 RESULTADOS DOS ALGORITMOS PARALELOS 69
6.3 COMPARAÇÃO ENTRE RESULTADOS DOS ALGORITMOS SEQÜENCIAIS E PARALELOS 87

CAPÍTULO 7 - CONCLUSÕES 105

CAPÍTULO 8 - REFERÊNCIAS 107

CAPÍTULO 9 – APÊNDICE “A” 112
9.1 GRÁFICOS SPEED UP E EFICIÊNCIA DOS ALGORITMOS AGP1 E AGP2. 112
9.2 GRÁFICOS SPEED UP E EFICIÊNCIA DOS ALGORITMOS AGP3 E AGP4. 120
9.3 GRÁFICOS SPEED UP E EFICIÊNCIA DOS ALGORITMOS AGP5 E AGP6. 128
9.4 GRÁFICOS SPEED UP E EFICIÊNCIA DOS ALGORITMOS AGP7 E AGP8. 135

Lista de Figuras

Figura 2.1: Uma UMP acoplada a um caminhão tanque. 21
Figura 2.2: Uma possível trajetória de uma UMP. 25
Figura 2.3: Exemplo de um problema RUMP. 26
Figura 4.1: Ilustra a codificação do algoritmo Apriori. 45
Figura 5.1: Grafo ilustrativo de um problema RUMP. 56
Figura 6.1: Soluções para instâncias de 50 vértices. 61
Figura 6.2: Soluções para instâncias de70 vértices. 61
Figura 6.3: Soluções para instâncias de 100 vértices. 62
Figura 6.4: Soluções para instâncias de 120 vértices. 62
Figura 6.5: Soluções para instâncias de 300 vértices. 63
Figura 6.6: Soluções para instâncias de 500 vértices. 63
Figura 6.7: Soluções para instâncias de 1000 vértices. 64
Figura 6.8: Tempo de execução para 50 vértices. 66
Figura 6.9: Tempo de execução para 70 vértices. 66
Figura 6.10: Tempo de execução para 100 vértices. 67
Figura 6.11: Tempo de execução para 120 vértices. 67
Figura 6.12: Tempo de execução para 300 vértices. 68
Figura 6.13: Tempo de execução para 500 vértices. 68
Figura 6.14: Tempo de execução para 1000 vértices. 69
Figura 6.15: Speed up médio de AGB. 72
Figura 6.16: Eficiência Média de AGB. 72
Figura 6.17: Speed up Médio do AG. 73
Figura 6.18: Eficiência Média de AG. 73
Figura 6.19: Speed up Médio de AG+BL. 74
Figura 6.20: Eficiência Média de AG+BL. 74
Figura 6.21: Speed up Médio de AG+BL+MD. 75
Figura 6.22: Eficiência Média de AG+BL+MD. 75
Figura 6.23: Comparativo de speed up médio (exceto AGP7 e AGP8). 77
Figura 6.24: Comparativo de Eficiência média (exceto AGP7 e AGP8). 77
Figura 6.25: Comparativo de speed up (exceto AGP7 e AGP8). 78
Figura 6.26: Comparativo de Eficiência média (exceto AGP7 e AGP8). 78
Figura 6.27: Comparativo de speed up médio (exceto AGP7 e AGP8). 79
Figura 6.28: Comparativo de Eficiência (exceto AGP7 e AGP8). 79
Figura 6.29: Comparativo de speed up (exceto AGP7 e AGP8). 80
Figura 6.30: Comparativo de Eficiência (exceto AGP7 e AGP8). 80
Figura 6.31: Percentual de tempo das Fases 1 e 2 em diferentes instâncias do RUMP. . 81
Figura 6.32: Tempo na Fase 1 para diferentes instâncias. 81
Figura 6.33: Percentual de tempo das Fases 1 e 2 em diferentes instâncias do RUMP. .. 82
Figura 6.34: Tempo na Fase 1 para diferentes instâncias. 82
Figura 6.35: Percentual de tempo das Fases 1 e 2 em diferentes instâncias do RUMP. .. 82
Figura 6.36: Tempo na Fase 1 para diferentes instâncias. 82
Figura 6.37: Tempo computacional entre versões AG+BL+MD (mestre-escravo). 85

Figura 6.38: Tempo computacional entre versões AG+BL+MD (mestre-escravo).....85
Figura 6.39: Tempo computacional entre versões AG+BL+MD (mestre-escravo).....86
Figura 6.40: Tempo computacional entre versões AG+BL+MD (mestre-escravo).....86
Figura 6.41: Comparativo de tempo entre seqüencial, AGP1 e AGP2 para 50 vértices....87
Figura 6.42: Comparativo de tempo entre seqüencial, AGP1 e AGP2 para 70 vértices....88
Figura 6.43: Comparativo de tempo entre seqüencial, AGP1 e AGP2 para 100 vértices..88
Figura 6.44: Comparativo de tempo entre seqüencial, AGP1 e AGP2 para 120 vértices..89
Figura 6.45: Comparativo de tempo entre seqüencial, AGP3 e AGP4 para 50 vértices....89
Figura 6.46: Comparativo de tempo entre seqüencial, AGP3 e AGP4 para 70 vértices....90
Figura 6.47: Comparativo de tempo entre seqüencial, AGP3 e AGP4 para 100 vértices..90
Figura 6.48: Comparativo de tempo entre seqüencial, AGP3 e AGP4 para 120 vértices..91
Figura 6.49: Comparativo de tempo entre seqüencial, AGP5 e AGP6 para 50 vértices....91
Figura 6.50: Comparativo de tempo entre seqüencial, AGP5 e AGP6 para 70 vértices....92
Figura 6.51: Comparativo de tempo entre seqüencial, AGP5 e AGP6 para 100 vértices..92
Figura 6.52: Comparativo de tempo entre seqüencial, AGP5 e AGP6 para 120 vértices..93
Figura 6.53: Comparativo de tempo entre seqüencial, AGP7 e AGP8 para 50 vértices....93
Figura 6.54: Comparativo de tempo entre seqüencial, AGP7 e AGP8 para 70 vértices....94
Figura 6.55: Comparativo de tempo entre seqüencial, AGP7 e AGP8 para 100 vértices..94
Figura 6.56: Comparativo de tempo entre seqüencial, AGP7 e AGP8 para 120 vértices..95
Figura 6.57: Comparativo de Volume entre AGB, AGP1 e AGP2.....96
Figura 6.58: Comparativo de Volume entre AGB, AGP1 e AGP2.....96
Figura 6.59: Comparativo de Volume entre AGB, AGP1 e AGP2.....97
Figura 6.60: Comparativo de Volume entre AGB, AGP1 e AGP2.....97
Figura 6.61: Comparativo de Volume entre AG, AGP3 e AGP4. 98
Figura 6.62: Comparativo de Volume entre AG, AGP3 e AGP4. 98
Figura 6.63: Comparativo de Volume entre AG, AGP3 e AGP4. 99
Figura 6.64: Comparativo de Volume entre AG, AGP3 e AGP4. 99
Figura 6.65: Comparativo de Volume entre AG+BL, AGP5 e AGP6. 100
Figura 6.66: Comparativo de Volume entre AG+BL, AGP5 e AGP6. 100
Figura 6.67: Comparativo de Volume entre AG+BL, AGP5 e AGP6. 101
Figura 6.68: Comparativo de Volume entre AG+BL, AGP5 e AGP6. 101
Figura 6.69: Comparativo de Volume entre AG+BL+MD, AGP7 e AGP8.....102
Figura 6.70: Comparativo de Volume entre AG+BL+MD, AGP7 e AGP8.....102
Figura 6.71: Comparativo de Volume entre AG+BL+MD, AGP7 e AGP8.....103
Figura 6.72: Comparativo de Volume entre AG+BL+MD, AGP7 e AGP8.....103
Figura 6.73: Comparação entre todas as versões de AGs.....104
Figura 9.1: Comparativo de Speed up entre versões Paralelas do AGB 112
Figura 9.2: Comparativo de Eficiência entre versões Paralelas do AGB 112
Figura 9.3: Comparativo de Speed up entre versões Paralelas do AGB 113
Figura 9.4: Comparativo de Eficiência entre versões Paralelas do AGB 113
Figura 9.5: Comparativo de Speed up entre versões Paralelas do AGB 113
Figura 9.6: Comparativo de Eficiência entre versões Paralelas do AGB 113
Figura 9.7: Comparativo de Speed up entre versões Paralelas do AGB 113
Figura 9.8: Comparativo de Eficiência entre versões Paralelas do AGB 113
Figura 9.9: Comparativo de Speed up entre versões Paralelas do AGB 114
Figura 9.10: Comparativo de Eficiência entre versões Paralelas do AGB 114
Figura 9.11: Comparativo de Speed up entre versões Paralelas do AGB 114

Figura 9.12: Comparativo de Eficiência entre versões Paralelas do AGB 114
Figura 9.13: Comparativo de Speed up entre versões Paralelas do AGB 114
Figura 9.14: Comparativo de Eficiência entre versões Paralelas do AGB 114
Figura 9.15: Comparativo de Speed up entre versões Paralelas do AGB 115
Figura 9.16: Comparativo de Eficiência entre versões Paralelas do AGB 115
Figura 9.17: Comparativo de Speed up entre versões Paralelas do AGB 115
Figura 9.18: Comparativo de Eficiência entre versões Paralelas do AGB 115
Figura 9.19: Comparativo de Speed up entre versões Paralelas do AGB 115
Figura 9.20: Comparativo de Eficiência entre versões Paralelas do AGB 115
Figura 9.21: Comparativo de Speed up entre versões Paralelas do AGB 116
Figura 9.22: Comparativo de Eficiência entre versões Paralelas do AGB 116
Figura 9.23: Comparativo de Speed up entre versões Paralelas do AGB 116
Figura 9.24: Comparativo de Eficiência entre versões Paralelas do AGB 116
Figura 9.25: Comparativo de Speed up entre versões Paralelas do AGB 116
Figura 9.26: Comparativo de Eficiência entre versões Paralelas do AGB 116
Figura 9.27: Comparativo de Speed up entre versões Paralelas do AGB 117
Figura 9.28: Comparativo de Eficiência entre versões Paralelas do AGB 117
Figura 9.29: Comparativo de Speed up entre versões Paralelas do AGB 117
Figura 9.30: Comparativo de Eficiência entre versões Paralelas do AGB 117
Figura 9.31: Comparativo de Speed up entre versões Paralelas do AGB 117
Figura 9.32: Comparativo de Eficiência entre versões Paralelas do AGB 117
Figura 9.33: Comparativo de Speed up entre versões Paralelas do AGB 118
Figura 9.34: Comparativo de Eficiência entre versões Paralelas do AGB 118
Figura 9.35: Comparativo de Speed up entre versões Paralelas do AGB 118
Figura 9.36: Comparativo de Eficiência entre versões Paralelas do AGB 118
Figura 9.37: Comparativo de Speed up entre versões Paralelas do AGB 118
Figura 9.38: Comparativo de Eficiência entre versões Paralelas do AGB 118
Figura 9.39: Comparativo de Speed up entre versões Paralelas do AGB 119
Figura 9.40: Comparativo de Eficiência entre versões Paralelas do AGB 119
Figura 9.41: Comparativo de Speed up entre versões Paralelas do AG 120
Figura 9.42: Comparativo de Eficiência entre versões Paralelas do AG 120
Figura 9.43: Comparativo de Speed up entre versões Paralelas do AG 120
Figura 9.44: Comparativo de Eficiência entre versões Paralelas do AG 120
Figura 9.45: Comparativo de Speed up entre versões Paralelas do AG 121
Figura 9.46: Comparativo de Eficiência entre versões Paralelas do AG 121
Figura 9.47: Comparativo de Speed up entre versões Paralelas do AG 121
Figura 9.48: Comparativo de Eficiência entre versões Paralelas do AG 121
Figura 9.49: Comparativo de Speed up entre versões Paralelas do AG 121
Figura 9.50: Comparativo de Eficiência entre versões Paralelas do AG 121
Figura 9.51: Comparativo de Speed up entre versões Paralelas do AG 123
Figura 9.52: Comparativo de Eficiência entre versões Paralelas do AG 123
Figura 9.53: Comparativo de Speed up entre versões Paralelas do AG 123
Figura 9.54: Comparativo de Eficiência entre versões Paralelas do AG 123
Figura 9.55: Comparativo de Speed up entre versões Paralelas do AG 123
Figura 9.56: Comparativo de Eficiência entre versões Paralelas do AG 123
Figura 9.57: Comparativo de Speed up entre versões Paralelas do AG 124
Figura 9.58: Comparativo de Eficiência entre versões Paralelas do AG 124

Figura 9.59: Comparativo de Speed up entre versões Paralelas do AG 124
Figura 9.60: Comparativo de Eficiência entre versões Paralelas do AG..... 124
Figura 9.61: Comparativo de Speed up entre versões Paralelas do AG 124
Figura 9.62: Comparativo de Eficiência entre versões Paralelas do AG..... 124
Figura 9.63: Comparativo de Speed up entre versões Paralelas do AG 125
Figura 9.64: Comparativo de Eficiência entre versões Paralelas do AG..... 125
Figura 9.65: Comparativo de Speed up entre versões Paralelas do AG 125
Figura 9.66: Comparativo de Eficiência entre versões Paralelas do AG..... 125
Figura 9.67: Comparativo de Speed up entre versões Paralelas do AG 125
Figura 9.68: Comparativo de Eficiência entre versões Paralelas do AG..... 125
Figura 9.69: Comparativo de Speed up entre versões Paralelas do AG 126
Figura 9.70: Comparativo de Eficiência entre versões Paralelas do AG..... 126
Figura 9.71: Comparativo de Speed up entre versões Paralelas do AG 126
Figura 9.72: Comparativo de Eficiência entre versões Paralelas do AG..... 126
Figura 9.73: Comparativo de Speed up entre versões Paralelas do AG 126
Figura 9.74: Comparativo de Eficiência entre versões Paralelas do AG..... 126
Figura 9.75: Comparativo de Speed up entre versões Paralelas do AG 127
Figura 9.76: Comparativo de Eficiência entre versões Paralelas do AG..... 127
Figura 9.77: Comparativo de Speed up entre versões Paralelas do AG 127
Figura 9.78: Comparativo de Eficiência entre versões Paralelas do AG..... 127
Figura 9.79: Comparativo de Speed up entre versões Paralelas do AG 127
Figura 9.80: Comparativo de Eficiência entre versões Paralelas do AG..... 127
Figura 9.81: Comparativo de Speed up entre versões Paralelas do AG + BL 128
Figura 9.82: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 128
Figura 9.83: Comparativo de Speed up entre versões Paralelas do AG + BL 128
Figura 9.84: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 128
Figura 9.85: Comparativo de Speed up entre versões Paralelas do AG + BL 129
Figura 9.86: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 129
Figura 9.87: Comparativo de Speed up entre versões Paralelas do AG + BL 129
Figura 9.88: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 129
Figura 9.89: Comparativo de Speed up entre versões Paralelas do AG + BL 129
Figura 9.90: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 129
Figura 9.91: Comparativo de Speed up entre versões Paralelas do AG + BL 130
Figura 9.92: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 130
Figura 9.93: Comparativo de Speed up entre versões Paralelas do AG + BL 130
Figura 9.94: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 130
Figura 9.95: Comparativo de Speed up entre versões Paralelas do AG + BL 130
Figura 9.96: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 130
Figura 9.97: Comparativo de Speed up entre versões Paralelas do AG + BL 131
Figura 9.98: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 131
Figura 9.99: Comparativo de Speed up entre versões Paralelas do AG + BL 131
Figura 9.100: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 131
Figura 9.101: Comparativo de Speed up entre versões Paralelas do AG + BL 131
Figura 9.102: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 131
Figura 9.103: Comparativo de Speed up entre versões Paralelas do AG + BL 132
Figura 9.104: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 132
Figura 9.105: Comparativo de Speed up entre versões Paralelas do AG + BL 132

Figura 9.106: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 132
Figura 9.107: Comparativo de Speed up entre versões Paralelas do AG + BL..... 132
Figura 9.108: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 132
Figura 9.109: Comparativo de Speed up entre versões Paralelas do AG + BL..... 133
Figura 9.110: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 133
Figura 9.111: Comparativo de Speed up entre versões Paralelas do AG + BL..... 133
Figura 9.112: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 133
Figura 9.113: Comparativo de Speed up entre versões Paralelas do AG + BL..... 133
Figura 9.114: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 133
Figura 9.115: Comparativo de Speed up entre versões Paralelas do AG + BL..... 134
Figura 9.116: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 134
Figura 9.117 : Comparativo de Speed up entre versões Paralelas do AGB + BL 134
Figura 9.118: Comparativo de Eficiência entre versões Paralelas do AGB + BL..... 134
Figura 9.119: Comparativo de Speed up entre versões Paralelas do AG + BL..... 134
Figura 9.120: Comparativo de Eficiência entre versões Paralelas do AG + BL..... 134
Figura 9.121: Comparativo de Speed up entre versões Paralelas do AG+BL+MD..... 135
Figura 9.122: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 135
Figura 9.123: Comparativo de Speed up entre versões Paralelas do AG+BL+MD..... 135
Figura 9.124: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 135
Figura 9.125: Comparativo de Speed up entre versões Paralelas do AG+BL+MD..... 136
Figura 9.126: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 136
Figura 9.127: Comparativo de Speed up entre versões Paralelas do AG+BL+MD..... 136
Figura 9.128: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 136
Figura 9.129: Comparativo de Speed up entre versões Paralelas do AG+BL+MD..... 136
Figura 9.130: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 136
Figura 9.131: Comparativo de Speed up entre versões Paralelas do AG+BL+MD..... 137
Figura 9.132: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 137
Figura 9.133: Comparativo de Speed up entre versões Paralelas do AG+BL+MD..... 137
Figura 9.134: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 137
Figura 9.135: Comparativo de Speed up entre versões Paralelas do AG+BL+MD..... 137
Figura 9.136: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 137
Figura 9.137: Comparativo de Speed up entre versões Paralelas do AG+BL+MD..... 138
Figura 9.138: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 138
Figura 9.139: Comparativo de Speed up entre versões Paralelas do AG+BL+MD..... 138
Figura 9.140: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 138
Figura 9.141: Comparativo de Speed up entre versões Paralelas do AG+BL+MD..... 138
Figura 9.142: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 138
Figura 9.143: Comparativo de Speed up entre versões Paralelas do AG+BL+MD..... 139
Figura 9.144: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 139
Figura 9.145: Comparativo de Speed up entre versões Paralelas do AG+BL+MD..... 139
Figura 9.146: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 139
Figura 9.147: Comparativo de Speed up entre versões Paralelas do AG+BL+MD..... 139
Figura 9.148: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 139
Figura 9.149: Comparativo de Speed up entre versões Paralelas do AG+BL+MD..... 140
Figura 9.150: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 140
Figura 9.151: Comparativo de Speed up entre versões Paralelas do AG+BL+MD..... 140
Figura 9.152: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 140

Figura 9.153: Comparativo de Speed up entre versões Paralelas do AG+BL+MD.....140
Figura 9.154: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 140
Figura 9.155: Comparativo de Speed up entre versões Paralelas do AG+BL+MD.....141
Figura 9.156: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 141
Figura 9.157: Comparativo de Speed up entre versões Paralelas do AG+BL+MD.....141
Figura 9.158: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 141
Figura 9.159: Comparativo de Speed up entre versões Paralelas do AG+BL+MD.....141
Figura 9.160: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD 141

Lista de Tabelas

Tabela 4.1: Exemplo de uma base transacional de uma padaria. 42
Tabela 4.2: Regras extraídas da Base de Dados da Tabela 4.1 43
Tabela 5.1: Matriz de distância associado ao grafo da figura 5.1 56
Tabela 6.1: Resultados comparativos entre as médias das soluções obtidas por cada algoritmo seqüencial. 65
Tabela 6.2: Descrição dos algoritmos Paralelos 70
Tabela 6.3: Resultados percentuais das Fases 1 e 2 em relação ao tempo computacional total. 83

Capítulo 1 - Introdução

A solução de problemas de elevado nível de complexidade computacional (Problemas NP-Completo e NP-Difícil) tem levado pesquisadores a proporem diferentes métodos exatos e aproximados em otimização combinatória.

Apesar do grande desenvolvimento verificado no tocante a métodos exatos, estes ainda possuem uso limitado a pequenas e médias instâncias na grande maioria dos problemas aplicativos.

Por outro lado, métodos aproximados ou heurísticos tradicionais, apesar de sua reconhecida flexibilidade em se adaptar às características muitas vezes dinâmicas de diferentes aplicações, sempre conviveram com limitações históricas, tais como: a parada prematura em ótimos locais muitas vezes distantes de um ótimo global em problemas de otimização.

A partir dos anos 80, começaram a surgir na literatura os chamados algoritmos metaheurísticos ou heurísticas inteligentes, que incorporam métodos genéricos que procuram evitar uma parada precoce [43].

Dentre as metaheurísticas, podemos citar como as mais conhecidas: Redes Neurais (RNs), Algoritmos Genéticos (AGs), Busca Tabu (BT), *Greedy Randomized Adaptive Search Procedure* (GRASP), *Simulated Annealing* (SA) e *Variable Neighborhood Search* (VNS).

Em otimização combinatória, os AGs na sua versão clássica (como proposto originalmente) não têm se mostrado eficientes quando comparados com outras metaheurísticas.

Na tentativa de tornar os AGs ou os algoritmos evolutivos mais eficientes, muitos autores têm propostas variantes, normalmente incorporando módulos de busca local [22] e/ou propondo versões paralelas, aproveitando o paralelismo implícito dos AGs [15, 17, 29, 36, 39, 45, 52, 59, 61, 62, 70].

Neste trabalho, colocamos como objetivo explorar essas duas frentes. A primeira propondo e analisando diferentes versões de um AG, desde sua versão clássica (básica) passando por variantes que incorporam módulos de busca local (AG+BL) e outras incorporando busca local e módulos de mineração de dados (AG+BL+MD).

Numa etapa seguinte, a partir dos algoritmos seqüenciais, são propostos modelos paralelos de AGs.

Para avaliar o desempenho dos algoritmos propostos, estes são utilizados para resolver um problema de roteamento de unidades móveis de pistoneio que chamaremos RUMP.

Contudo, salientamos que as idéias básicas propostas podem ser adaptadas facilmente para a solução de diferentes problemas da área de otimização combinatória.

O restante do trabalho está organizado da seguinte maneira:

O Capítulo 2 descreve o problema de Roteamento de uma Unidade Móvel de Pistoneio, que é uma generalização do Problema do Caixeiro Viajante (PCV) e a seguir, listamos alguns trabalhos já existentes para o problema.

No Capítulo 3, são tratados os algoritmos Genéticos (AG). São apresentados componentes básicos de um AG, tais como: operadores genéticos e criação da população inicial. Também são apresentadas estratégias de paralelismo.

No capítulo seguinte, abordamos brevemente o conceito básico de mineração de dados, descrevendo alguns algoritmos existentes e o algoritmo utilizado em nossa estratégia.

Nossa proposta é apresentada no Capítulo 5. Inicialmente, apresentamos nosso modelo Básico de AG, passando por versões aprimoradas de AG em que incorporamos apenas um módulo de busca local, e versões que incorporam módulos de busca local e mineração de dados. Numa segunda etapa, as versões paralelas dos algoritmos genéticos paralelos propostos (AGP) são descritas.

Os resultados computacionais são apresentados no Capítulo 6.

Finalmente, no Capítulo 7, são apresentadas as conclusões.

Capítulo 2 - O Problema de Roteamento de uma Unidade Móvel de Pistoneio (RUMP)

A extração de petróleo pode ser feita basicamente de duas maneiras: em poços marítimos ou em poços terrestres. Poços terrestres de petróleo denominados "não surgentes" são os que não têm capacidade de elevar o fluido até a superfície. Inicialmente, justifica-se o custo do uso de uma sonda para extração de óleo, uma vez que possuem uma grande quantidade de óleo a ser explorado. Por se tratar de um produto não renovável, o volume de fluido acumulado no poço vai diminuindo até que não seja mais lucrativo manter um equipamento alocado para este poço. Por esta razão, o razoável é transferir o aparelho para outro poço de maior produção, já que o número de sondas disponíveis é normalmente inferior ao número de poços existentes nos campos petrolíferos, como ocorre na bacia Potiguar e a bacia do Recôncavo [46]. Após uma extração, técnicas de estimulação são empregadas para que o volume de petróleo do poço retorne a uma quantidade que justifique nova extração, conseqüentemente a espera de alguns dias deve ser respeitada, isto é, cada poço apresenta uma taxa de reenchimento de fluido, que vai sendo reduzida à medida que se aproxima do chamado nível de estabilização, no qual observa-se uma variação pequena de fluido acumulado [46].

Com cerca de 4.000 poços de gás e petróleo terrestres, a bacia Potiguar está situada entre o Rio Grande do Norte e o Ceará (RN-CE). A importância desta área corresponde a cerca de 10% da produção nacional. Os campos petrolíferos são compostos na sua grande maioria (em torno de 98%) por poços que não têm capacidade própria de elevação dos fluidos (óleo e água) até a superfície, necessitando de um método auxiliar de elevação artificial de fluidos, por isso são chamados de *não surgentes*. Para extrair o composto, necessitamos de métodos de elevação artificial de fluidos, tais como: bombeio mecânico (cavalo de pau), o bombeio por cavidades progressivas (BCP) ou o bombeio por Unidade Móvel de Pistoneio (UMP) (veja figura 2.1), que são usados em 81%, 9% e 4% do total de poços utilizados na região [46], respectivamente.

Figura 2.1: Uma UMP acoplada a um caminhão tanque.

Às vezes, em poços não surgentes, que utilizam um método de elevação artificial de fluido com equipamento permanente, como Bombeio Mecânico ou Bombeio por Cavidades Progressivas, não se torna rentável manter estes equipamentos devido à vazão insuficiente de composto. A alternativa é a utilização de um meio de extração intermitente chamado Unidade Móvel de Pistoneio (UMP). A UMP é um equipamento móvel, acoplado a um caminhão ou um trator, composto por acessórios de elevação de fluidos (guincho hidráulico, rolo de cabo de aço, sistema de lança e um tanque) [46].

Cada campo de exploração de petróleo possui uma ETO (Estação de Tratamento de Óleo) que é uma unidade responsável pela separação do óleo da água. Os poços estão ligados por estradas, isto é, não existem poços isolados. Cada UMP parte de sua ETO, percorre um número de poços, realiza a extração deste petróleo e retorna à ETO.

O objetivo do Problema de Roteamento de uma UMP chamado RUMP é encontrar uma seqüência de poços (rota) a serem visitados, de maneira a se extrair o máximo de composto possível, respeitando os limites impostos pelo problema, como a capacidade da UMP e o limite de tempo total de uma rota, que deve ser o período da jornada diária do trabalhador (8h).

A capacidade da UMP pode ser considerada ilimitada, pois na prática podemos acoplar um ou mais caminhões tanques a uma UMP na medida em que se extrai o composto (Figura 2.1). Neste trabalho, estamos desconsiderando o tempo de montagem e desmontagem do equipamento para retirada do petróleo, assim como o tempo de permanência de uma UMP em cada poço visitado. A distância e o tempo de percurso entre um poço e outro são considerados os mesmos.

O problema do Roteamento de Unidade Móvel de Pistoneio (RUMP) pode ser visto como um problema de Roteamento de Veículos (PRV), onde uma única UMP deve atender a demanda de vazão, dos diversos poços. A UMP deve partir de uma única Estação de Tratamento de Óleo (ETO) e retornar a ela, após extrair o petróleo de um certo número de poços percorridos, respeitando-se o limite da jornada diária do trabalhador (8 horas).

O problema RUMP pode ser definido da seguinte forma: Dado um grafo não direcionado G=(V, D), V ={v_0, v_1, ... , v_n} representa um conjunto de poços e D = { (v_i,v_j): i≠j/v_i,v_j ∈ V} representa um conjunto de estradas. Em cada vértice v_i, existe uma vazão associada (vazão(i)), que representa o ganho de óleo extraído naquele poço. Cada aresta contém um peso associado, que representa a distância e o tempo de percurso entre os dois poços. Deseja-se maximizar a quantidade de petróleo extraído por uma UMP, satisfazendo as restrições envolvidas.

2.1 Modelo Matemático

Admitamos que **n** poços são indexados por i=1, ... , n. Assumimos uma rede de estradas de um dado campo petrolífero como sendo um grafo não direcionado G= (V, D), V= {0, ..., n}, onde cada aresta (i, j) do conjunto D corresponde a distância e o tempo de percurso entre os vértices i e j. Definimos este custo como sendo o tempo de percurso c_{ij} entre os poços i e j quaisquer, obtido a partir de distâncias mínimas $d_{ij} \in D_m$, onde D_m é a matriz de distâncias mínimas associada ao G. A estação de tratamento de óleo (ETO) de onde a UMP sai e retorna, após uma jornada diária de L horas é considerada como sendo o vértice 0. Desta forma, as constantes e variáveis empregadas neste modelo são definidas como:

Constantes

n: número total de poços do campo petrolífero.

p_i: volume de fluido que pode ser extraído do poço i.

bsw_i: taxa de água por volume recuperado para o poço i.

c_{ij}: menor tempo de percurso entre os poços i e j.

L: duração da jornada diária da UMP.

Variáveis

P: Produção diária total de óleo a ser obtida (volume em m^3).

r_{ij}: { 1, se os poços i e j são consecutivos no roteamento para i, j = 0,..., n.
0, caso contrário.

2.2 Formulação Matemática

Maximizar:

$$P = \sum_{i=1}^{n} (1 - bsw_i)\, p_i \qquad (1a)$$

Sujeito à:

$$\sum_{i=0}^{n} \sum_{j=0}^{m} r_{ij}\, c_{ij} \leq L, \qquad (2a)$$

$$\sum_{j=0}^{n} r_{ji} = 1, \quad \text{para i=1, ..., n,} \qquad (3a)$$

$$\sum_{j=0}^{n} r_{ij} = 1, \quad \text{para i=1, ..., n,} \qquad (4a)$$

$$\sum_{j=1}^{n} r_{j0} = 1, \tag{5a}$$

$$\sum_{j=1}^{n} r_{0j} = 1, \tag{6a}$$

$r_{ij} \in \{0,1\}$, para todo i, j $\in$ V. (7a)

A restrição (2a) impõe que o somatório no trajeto ETO $\Rightarrow$ poços $\Rightarrow$ ETO, não exceda o limite de jornada diária de uma UMP. As restrições de (3a) a (6a), garantem a viabilidade da rota definida pelas variáveis r_{ij}. As equações (3a) a (4a) garantem que há apenas uma aresta entrando e uma aresta saindo do poço i. De maneira análoga, as equações (5a) e (6a) garantem que há apenas uma aresta entrando e uma aresta saindo da ETO.

A figura 2.2 nos mostra uma possível trajetória a ser percorrida pela UMP, indicada pelas setas que representam o seu deslocamento, dado um campo petrolífero fictício formado por 99 poços [46].

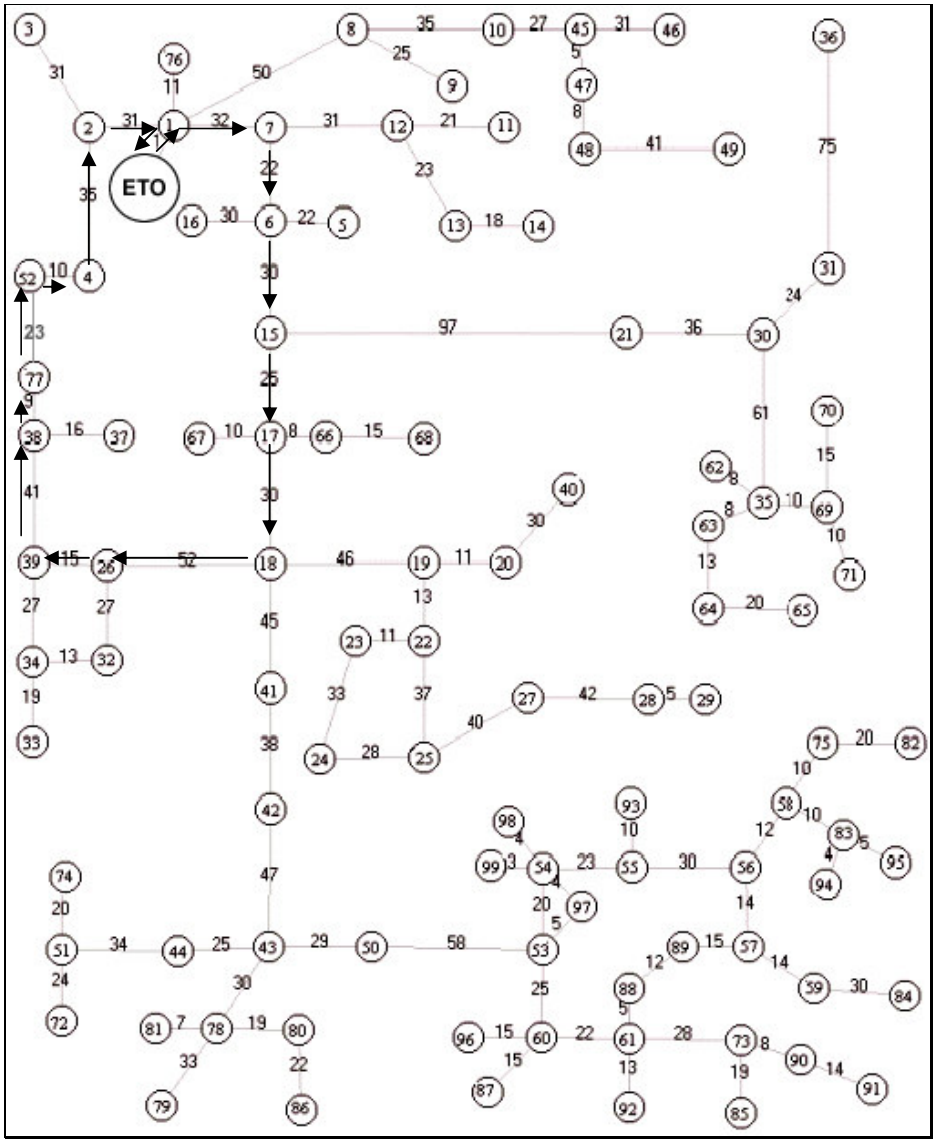

Figura 2.2: Uma possível trajetória de uma UMP.

O grafo da figura 2.3 representa um exemplo ilustrativo de um campo petrolífero para o problema RUMP. Os vértices (poços) são representados por círculos e os valores em seu interior são as identificações dos poços neste campo. O rótulo P_i associado a cada poço constitui o volume que será acumulado ao se visitar o nó i. Os valores associados a cada aresta indicam a distância e o tempo de percurso entre dois poços que são consideradas idênticas, neste trabalho, para efeito de simplificação. Cada campo de exploração de petróleo possui uma ETO (Estação de Tratamento de Óleo) que é uma

unidade responsável pela separação do óleo da água. Os poços estão ligados por estradas, isto é, não existem poços isolados.

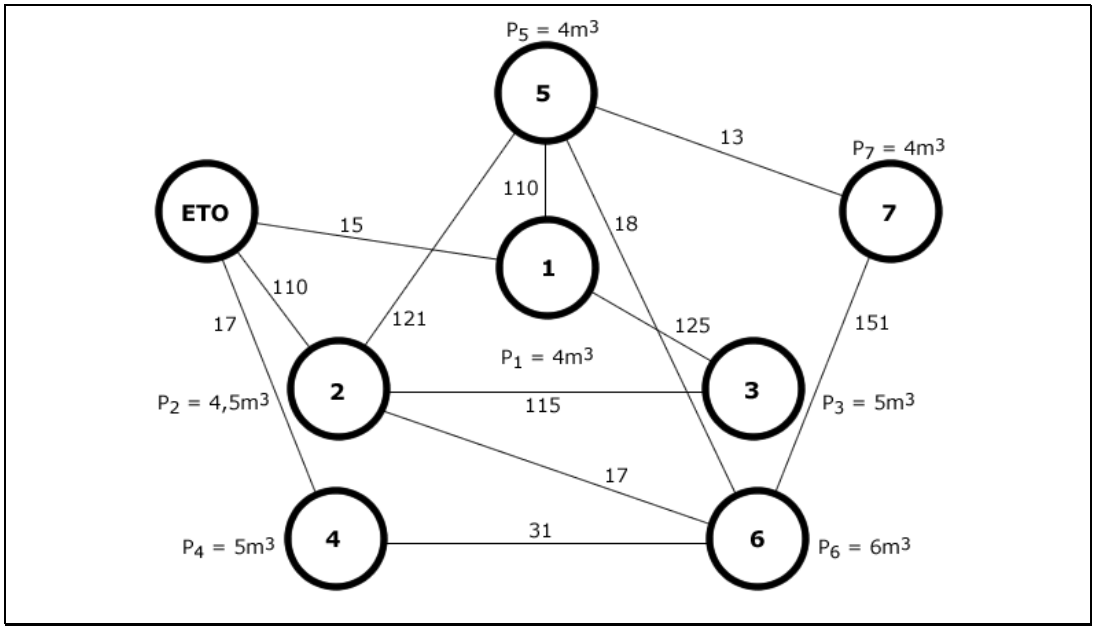

Figura 2.3: Exemplo de um problema RUMP.

De nosso conhecimento, existem poucos trabalhos desenvolvidos para a solução do RUMP e todos eles de autoria de pesquisadores do departamento DIMAP da UFRN [46, 47, 4].

Sobre o problema RUMP podemos citar um AG proposto em [46]. Neste trabalho, utiliza-se um algoritmo genético básico usando um operador *crossover* PMX-adaptado que procura gerar novos filhos através da troca de genes marcados nos dois pais. A mutação é feita pela alteração de uma ou mais características em um indivíduo.

Em [47] é descrito um GRASP para o RUMP. A estratégia gulosa, GRASP, é utilizada para gerar uma população inicial mais elaborada e não aleatória como é feito pelos AGs convencionais. Numa outra etapa, ocorre o aperfeiçoamento da solução inicialmente criada, através de um módulo de busca local aplicado no melhor indivíduo.

Em [4] é proposta uma metaheurística baseada em conceitos de colônia de formigas (*Ant Colony Systems*) para a solução do RUMP. O princípio deste trabalho é selecionar uma rota, na qual os poços são determinados com base em uma função probabilística. Esta função probabilística leva em conta os poços que mais contribuíram para a maximização da função objetivo.

Na literatura existem problemas similares ao RUMP tais como, *The Traveling Purchase Problem* (TPP) [10, 51, 71], *The Orienteering Problem* (OP) [13, 14], e *The Prize Collecting Problem* (PCP) [38].

Capítulo 3 - Algoritmo Genético

3.1 Introdução

Os algoritmos genéticos (AGs) são programas evolutivos, baseados no princípio da seleção natural, onde os indivíduos mais aptos sobrevivem e os menos aptos tendem a ser descartados; e da hereditariedade, onde as características dos pais são transmitidas para os filhos. Dessa forma, os AGs favorecem os candidatos mais promissores para a solução de um problema.

Os algoritmos evolutivos fazem parte da Computação Evolutiva (CE). A CE propõe uma maneira alternativa ao processamento de dados convencional. Ela não exige conhecimento prévio para a busca de uma solução para o problema e baseia-se em mecanismos encontrados na natureza, tais como: adaptabilidade e sobrevivência.

Os Algoritmos Genéticos (AGs) foram inicialmente propostos por John Holland (1975) da Universidade de Michigan, mas somente a partir dos anos 80 é que realmente começaram a se popularizar.

A idéia inicial de Holland (1975) foi tentar imitar algumas etapas do processo de evolução natural das espécies incorporando-as a um algoritmo computacional.

Basicamente, o ponto de referência foi gerar, a partir de uma população de indivíduos, novos indivíduos com propriedades genéticas superiores às de seus antecedentes. Esta idéia foi então associada a soluções de problemas onde, a partir de um conjunto de soluções atuais, são geradas novas soluções superiores às antecedentes, sob algum objetivo pré-estabelecido.

Os Algoritmos Genéticos, na sua forma original (AGB), trabalham normalmente com uma representação binária que é associada a uma solução do problema abordado; o processo de seleção de indivíduos reprodutores é muitas vezes aleatório e utiliza mecanismos de mutação aleatória e/ou *crossover* para a fase de reprodução de novos indivíduos. Ainda nas versões básicas, o critério de sobrevivência de indivíduos é definido a partir de uma função que mede o nível de aptidão de cada indivíduo; e normalmente um indivíduo filho (*offspring*) somente ocupa o lugar do pai, se este for considerado melhor, segundo esta função de aptidão.

Estes mecanismos, assim definidos, não têm se mostrado eficientes na solução de diversos problemas complexos de otimização combinatória; em parte pelas próprias deficiências no processo de busca, causadas pela escolha aleatória de reprodutores e pela formação prematura de uma população homogênea causada pelo critério de escolha dos indivíduos sobreviventes.

Um outro fator responsável por este baixo desempenho dos AGs básicos é a existência de algoritmos heurísticos muito eficientes para a maioria dos problemas da área de Otimização, tais como o Problema do Caixeiro Viajante e algumas de suas generalizações.

Com isso, apesar do seu caráter genérico, o desempenho dos AGs, na sua forma básica, não é satisfatório em termos da qualidade da solução gerada para muitos problemas que já possuem algoritmos heurísticos eficientes para as suas soluções. Além disso, outro gargalo dos AGs, é o fato de normalmente estes exigirem um tempo computacional maior que a maioria das heurísticas e metaheurísticas.

Estes fatos têm levado diversos pesquisadores a proporem modificações nos AGs básicos, incorporando técnicas de busca local e/ou ferramentas de outras metaheurísticas tais como: *Simulated Annealing*, *Tabu Search* (TS), *Scatter Search*, entre outros, para melhorar a qualidade das soluções dos AGs ou desenvolvendo versões paralelas para melhorar o desempenho dos AGs, no tocante aos tempos computacionais exigidos.

Uma das variantes dos AGs é conhecida como *Scatter Search* (SS), que é vista como uma versão determinística dos AGs, incorporando, além disso, conceitos de TS. No SS, também são geradas populações de soluções como nos AGs, mas as etapas de seleção de indivíduos pais e de reprodução, são efetuadas através de critérios determinísticos. A geração de novas soluções a partir de informações de um conjunto de soluções atuais é feita através de combinações lineares das soluções atuais [21, 22, 24, 25, 26, 27].

Outras variantes de AGs básicos incluem nestes, módulos de busca local e são conhecidos na literatura como AGs Híbridos com Busca Local ou Algoritmos Meméticos [5, 44].

Na fase de otimização local, um único indivíduo sofre exaustivas combinações em cada um dos genes, até que se produza uma melhoria na aptidão. Geralmente, o resultado substitui o indivíduo original caso a aptidão obtida com a otimização local seja melhor do que a aptidão original.

Um Algoritmo Memético (AM) consegue melhores resultados globais se a função de otimização local for conseguir refinar as soluções geradas pelos operadores de *crossover* e mutação de um algoritmo genético.

Outra variação importante de um AG caracteriza-se por não utilizar apenas soluções completas do problema em seus indivíduos, mas partes de soluções, denominadas esquemas. O Algoritmo Genético Construtivo (AGC) procura utilizar uma medida de adaptação do esquema aos indivíduos de uma população. Os novos esquemas são gerados através do cruzamento de vários outros, presentes a cada geração. Um processo de avaliação de esquemas é responsável por identificar os esquemas mais promissores e assim, preservá-los para a geração seguinte. Em [16], os autores utilizam um AGC para resolver o problema de coloração de grafos.

Outra variante dos AGs, são chamados algoritmos co-evolutivos que procuram, decompor uma determinada tarefa complexa em diversas sub-tarefas mais simples. Cada sub-tarefa sofre ação de um algoritmo evolutivo até que seus indivíduos se tornem bastante especializados no que diz respeito à sub-tarefa atribuída. Em seguida, um critério de reprodução global é utilizado de maneira a combinar as melhores soluções de

cada um dos algoritmos evolutivos e assim conseguir resultados satisfatórios para serem aplicados à tarefa complexa.

A decomposição de uma tarefa complexa nem sempre é uma tarefa fácil e envolve aspectos importantes, tais como: o nível de dependência das sub-tarefas em relação à tarefa complexa; o conhecimento a priori da quantidade de sub-tarefas geradas a partir de uma tarefa complexa; a importância de cada sub-tarefa na função de cálculo da aptidão global e, finalmente, se as sub-tarefas descobertas abrangem todo o espaço de pesquisa.

Em [54], é descrito um estudo de caso, onde um algoritmo co-evolutivo é aplicado à construção de uma rede neural artificial. Já em [55], um algoritmo co-evolutivo procura treinar um robô de maneira a apanhar um objeto em uma sala cheia de obstáculos em movimento.

Nos últimos anos, além de significativas contribuições sobre desenvolvimento de novos algoritmos seqüenciais para AGs, encontramos muitos trabalhos sobre algoritmos genéticos paralelos, utilizando ferramentas do tipo: PVM (*Parallel Virtual Machine*), MPI (*Message Passing Interface*) [15, 29, 36, 48, 49, 50, 59, 61, 66] e outros.

A idéia central de um AGB é a seguinte: Uma população inicial é formada por um conjunto de indivíduos gerados aleatoriamente ou através do uso de uma heurística simples, que constituem normalmente soluções viáveis para a resolução do problema desejado. Esta população por sua vez, é enviada a uma função de avaliação que estabelece uma aptidão para cada indivíduo. Esta aptidão denota o quão apto está este indivíduo para prosseguir nas gerações seguintes. Os membros menos aptos tendem a ser eliminados, e efetua-se a ação da mutação ou o *crossover* dos indivíduos mais aptos para se gerar novos descendentes. A esta etapa, chamamos de reprodução. Um critério de parada é estabelecido como parâmetro de entrada. Dentre os critérios mais usados estão: número máximo de gerações, número máximo de gerações sem atualização da melhor solução gerada até o momento, e outros.

Descrevemos a seguir, os passos de um AGB:

```
1:  t =0;
2:  Gerar população inicial P(0);
3:  Para cada indivíduo i da população atual P(t) faça
4:     avaliar aptidão do indivíduo i
5:  Fim para;
6:  Armazenar melhor solução encontrada até o momento;
7:  Enquanto critério de parada não for satisfeito faça
8:    t = t+1;
9:    selecionar população P(t) a partir de P(t-1);
10:   aplicar operadores de reprodução sobre   P(t);
11:   avaliar P(t);
12:   Armazenar melhor solução encontrada até o momento;
13: Fim enquanto;
14: Apresentar melhor indivíduo gerado pelo AG;
```

Figura 3.1 – Passos de um AGB;

Na linha 1 da Figura 3.1, iniciamos o contador t, que indica o número atual da geração do AG. O passo seguinte consiste em gerar uma população de indivíduos. Na linha 3, avalia-se cada membro gerado atribuindo uma aptidão ou um índice, que irá classificá-lo como "mais" ou "menos" apto à solução (linha 4). Na linha 6, armazenamos o melhor indivíduo encontrado até o momento. Na linha 7, inicia-se um processo iterativo que só termina quando o critério de parada for satisfeito. Na linha 8, incrementamos o contador da geração. Na linha 9, selecionamos, a partir da população anterior, alguns indivíduos para sofrerem a ação dos operadores (mutação ou *crossover*) que serão aplicados na linha 10. Na linha 11, faz-se nova avaliação da população gerada e em seguida armazena-se o melhor indivíduo encontrado até o momento (linha 12). E o processo retorna à linha 7, onde testamos novamente para verificar se atingimos o critério de parada. Se não atingimos, executamos a linha 8, caso contrário, executamos a linha 14 que vai exibir a solução do problema proposto.

3.2 Componentes de um AGB

A seguir, trataremos das seguintes questões referentes ao projeto do AG na sua forma básica:

- Representação de soluções
- População inicial
- Função de avaliação do indivíduo
- Mecanismos de reprodução
- Parâmetros do AG

3.2.1 Representação Genética de Soluções Viáveis

Cada indivíduo de um AG representa normalmente uma solução viável, que sofrerá repetidas melhorias até que se aponte uma solução final para o problema analisado.

Existem dois tipos de representações bastante conhecidas:

- A representação binária: utiliza somente os dígitos 0 (zero) e 1 (um) para indicar ausência ou presença de uma determinada característica no indivíduo. Podemos ter o seguinte exemplo de representação binária, com tamanho igual a n=10 poços:

  (0 1 0 0 1 1 1 0 1 0)

  Considerando o RUMP, significaria que os poços 1,4,5,6 e 8 seriam selecionados para visita da UMP.

- A representação por inteiros: utiliza números inteiros. No caso do RUMP, os números teriam valores máximos iguais a n, onde n=|N| representaria o número de poços existentes, cujo volume acumulado é suficiente para compensar a visita do veículo coletor, por exemplo: (0 4 5 1 6 3 2 7)

3.2.2 População Inicial

O processo de construção desta população varia desde a forma mais simples, onde os indivíduos da população são criados aleatoriamente, até estratégias mais complexas e elaboradas.

Seja qual for o processo de construção, normalmente a população inicial é constituída por soluções viáveis ao problema em que se aplica.

3.2.3 Função de Avaliação

A função de avaliação de um indivíduo é uma maneira de avaliar a qualidade de solução que este representa, à medida em que o AG vai sendo executado. Ao longo do processo evolutivo, cada membro da população recebe uma nota ou um índice que determina o quão adaptado este está perante o problema em que foi aplicado. Esta nota recebe o nome de aptidão. Os AGs procuram então, preservar os indivíduos de mais alta aptidão e eliminar os de menor aptidão.

3.2.4 Reprodução

Nesta etapa, novos indivíduos são gerados a partir da combinação de indivíduos da população atual através de mecanismos conhecidos como operadores de reprodução.

Os operadores têm o objetivo de, a partir de uma população, gerar uma nova população, combinando características dos indivíduos pais mais aptos. Tradicionalmente, os operadores mais conhecidos são: mutação e *crossover*.

Após um determinado número de gerações, as soluções encontradas por um AG tendem a ficar similares. Os operadores de mutação têm também a função de diversificar uma população do AG permitindo, desta forma, escapar de ótimos locais ainda distantes de um ótimo global.

Este mecanismo seleciona, arbitrariamente, um ou mais componentes de indivíduos para serem alterados. Abaixo apresentamos um exemplo de mutação:

(Antes) $p_1 = (0\ 4\ 5\ \underline{\mathbf{0}}\ 1\ 6\ \underline{\mathbf{3}}\ 2\ 7\ 0)$

(Depois) $o_1 = (0\ 4\ 5\ \underline{\mathbf{3}}\ 1\ 6\ \underline{\mathbf{0}}\ 2\ 7\ 0)$

Para este exemplo, podemos observar que houve uma alteração no gene 4 e no 7 de p_1.

O operador *crossover* envolve geralmente dois indivíduos pais (*parents*), que trocam parte do seu material genético para geração de um ou mais filhos (*offsprings*) a partir de cortes em algum ponto do indivíduo. Este operador é muito utilizado num algoritmo evolutivo, pois é rápido e de certa forma, mantém nos filhos características dos pais.

Este operador pode ser utilizado de várias maneiras, descritas a seguir:

No *crossover* de um único ponto, os pais sofrem um único corte e as informações são trocadas a partir deste corte, produzindo filhos com material de ambos os pais. Para ilustrar este tipo de operador, utilizaremos o caracter "|" para indicar o corte nos indivíduos. P_1 e p_2 são os indivíduos pais e o_1 e o_2 os indivíduos filhos. Na representação abaixo,

$p_1 = (0\ 1\ 2\ |\ 3\ 4\ 5\ 6) = (p_{11}|p_{12})$ e $o_1 = (0\ 1\ 2\ |\ 6\ 5\ 4\ 3) = (p_{11}|p_{22})$

$p_2 = (0\ 2\ 1\ |\ 6\ 5\ 4\ 3) = (p_{21}|p_{22})$ e $o_2 = (0\ 2\ 1\ |\ 3\ 4\ 5\ 6) = (p_{21}|p_{12})$

Para este exemplo, notamos que a parte esquerda do indivíduo p_1 se uniu com a parte direita do indivíduo p_2, o que acabou gerando o indivíduo filho o_1, que possui características combinadas dos dois pais. O indivíduo o_2 é constituído de maneira análoga a o_1.

No *crossover* multi-corte ocorre a presença de dois ou mais cortes nos indivíduos pais. A idéia base é a mesma do indivíduo de único corte, onde neste caso, apenas o segmento do meio é trocado entre os indivíduos, como mostra o exemplo abaixo.

$p_1 = (\ 0\ 1\ |\ 2\ 3\ 4\ |\ 5\ 6) = (p_{11}|p_{12}|p_{13})$ e $o_1 = (0\ 1\ |\ 4\ 3\ 2\ |\ 5\ 6) = (p_{11}|p_{22}|p_{13})$

$p_2 = (\ 0\ 5\ |\ 4\ 3\ 2\ |\ 1\ 6) = (p_{21}\ |\ p_{22}|p_{23})$ e $o_2 = (0\ 5\ |\ 2\ 3\ 4\ |\ 1\ 6) = (p_{21}\ |\ p_{12}|p_{23})$

Infelizmente, para a maioria dos problemas de otimização combinatória de elevada complexidade, o uso de operadores mutação e *crossover* num AG não se mostra tão eficiente como outras heurísticas existentes. Isso motivou pesquisadores a proporem outros passos de reprodução e/ou a inclusão de módulos adicionais num AG.

3.2.5 Parâmetros do AG

Os ajustes aos parâmetros dos algoritmos genéticos são importantes para o desempenho do algoritmo. Alguns parâmetros normalmente presentes num AG são descritos a seguir:

Tamanho da população

Experiências computacionais mostram que o aumento do tamanho da população normalmente contribui para encontrar a solução mais eficiente. Por outro lado, o tempo computacional se torna maior devido ao volume maior de operações a serem executadas em uma geração. Recomenda-se ajustar este parâmetro considerando o tempo computacional e a qualidade da solução.

Taxa de mutação

A taxa de mutação também tem um importante impacto sobre a busca da solução de um algoritmo evolutivo. Se a taxa de mutação for elevada, a busca pode tornar-se excessivamente aleatória. Caso contrário, corremos o risco de depararmos com uma convergência prematura, ou seja, a busca pode ficar estagnada num ótimo local ainda distante de um ótimo global.

Critério de parada

Um dos critérios de parada, mais utilizados num AG, é o número de gerações.

Outros critérios de parada, usualmente presentes são: número máximo de gerações consecutivas sem melhora da melhor solução e taxa de renovação de uma população. Neste último caso, quando a taxa de renovação é muito pequena, ou o atinge-se o critério de parada ou ativa-se um procedimento para diversificar a atual população.

3.3 Algoritmos Genéticos Paralelos

Uma das limitações dos Algoritmos Genéticos Seqüenciais é que, normalmente, o tempo de execução requerido é grande quando comparado com os exigidos por outras heurísticas. Sabemos que os AGs são implicitamente paralelos, o que incentiva o desenvolvimento de versões paralelas [15, 17, 29, 36, 39, 45, 52, 59, 61, 62, 70].

Existem diferentes formas para se paralelizar um Algoritmo Evolutivo (AE). As maiores diferenças encontram-se na organização da população e no método utilizado para selecionar indivíduos para o processo de reprodução.

A classificação de Algoritmos Genéticos Paralelos (AGP) pode ser feita segundo a granularidade do processamento [7].

Um AGP de granularidade fina apresenta um modelo onde, a população é dividida em muitas outras pequenas sub-populações e onde existe uma intensa troca de mensagens entre elas. Geralmente, o *crossover* ocorre entre i indivíduos alocados em processadores diferentes. O custo de comunicação entre os processadores é alto, o que o torna inadequado usualmente para execução nos sistemas distribuídos atuais.

Por outro lado, AGPs que apresentam um certo grau de isolamento entre as sub-populações (menos numerosas), são ditos exibir um paralelismo de granularidade grossa, e em geral produzem resultados melhores em sistemas distribuídos [6]. Este modelo introduz o conceito de migração, onde se pode enviar um grupo de um ou mais indivíduos a outras sub-populações. Para esta classificação o modelo é denominado "ilha", por fazer analogia às ilhas, que são ambientes naturais isolados e onde os animais que se desenvolvem são bastante adaptados ao cenário em que se encontram. Cada sub-população pode ser tratada como uma unidade de reprodução diferente, sendo controlado por um AG convencional [70].

Uma outra categoria de AGP é a denominada de "paralelismo global". Nesta categoria, a população é tratada como um todo, e a cada processador é atribuído uma etapa do algoritmo genético. Este modelo é empregado tanto em arquiteturas de memória compartilhada quanto em ambientes de memória distribuída. Na primeira arquitetura, toda memória é vista por todos os processadores. Na segunda arquitetura, um modelo mestre-escravo é aplicado. O modelo mestre-escravo é caracterizado pela evolução natural do algoritmo seqüencial [39]. Neste modelo existe um processador que supervisiona os outros processadores. O mestre delega tarefas aos escravos que após completá-las retornam ao mestre o seu resultado. Desta forma, o processador mestre pode passar ou não novas tarefas aos escravos. Além disso, o mestre pode ou não realizar outras tarefas, além de gerenciar os escravos.

Algumas desvantagens deste modelo são:

- O mestre assume um papel vital para aplicação. Se este processador falhar, toda aplicação pára.
- O modelo possui limitação no que diz respeito à escalabilidade, no caso do número de processos ser muito grande.

Capítulo 4 - Mineração de Dados

Mineração de Dados (MD) é um processo que envolve etapas como a preparação dos dados, busca por padrões e avaliação do conhecimento. Conhecimento este, que pode ser utilizado em alguma etapa decisória em sistemas inteligentes ou ainda para melhorar o desempenho de algoritmos. Em nosso trabalho, a extração de conhecimento de uma base de dados é feita através de algoritmos para identificação de regras de associação.

A definição de MD mais aceita atualmente por pesquisadores foi elaborada por Fayyad, Piatetsky-Shapiro e Smith [11]: "Extração de conhecimento de base de dados ou mineração de dados é o processo de identificação de padrões válidos, novos, potencialmente úteis e compreensíveis embutidos nos dados".

Na literatura, podemos citar alguns algoritmos eficientes para identificar regras de associação, dentre eles: *Apriori* [2], *Tree Projection* [1], *Partition* [57], *SETM* [32], *Eclat* [53], *DHP* [73], entre outros.

Neste trabalho, vamos nos concentrar somente na etapa de gerar regras eficientes de associação de elementos de uma massa de dados relacionada aos algoritmos aqui propostos.

4.1 Regras de Associação

Uma regra de associação corresponde a um padrão de itens em uma base de dados de uma aplicação que ocorre com uma certa freqüência [68].

A idéia por traz da regra de associação, é poder prever o comportamento de certos itens dentro de uma base de dados, identificando padrões de relacionamento entre eles. As regras de associação são apresentadas da seguinte forma: **SE** (um conjunto de atributos preditos) **ENTÃO** (um conjunto de atributos conseqüentes), onde a interseção entre os atributos destes dois conjuntos (preditos e conseqüentes) deve ser um conjunto vazio.

Um exemplo do uso de regras de associação é a análise de transações de compra [68]. Seja D, uma base de dados transacional definida como um conjunto de transações, contendo informações sobre venda de produtos e serviços automotivos, a regra de associação: {carro novo, ar condicionado} $\Rightarrow$ {seguro}, indica que, **SE** o cliente compra um carro novo e o acessório de ar condicionado, **ENTÃO** adquire também o seguro deste carro com uma certa certeza. Fazendo uma analogia ao RUMP, a regra de associação {Poço 1, Poço 2} $\Rightarrow$ {Poço 7}, indica com um certo grau de certeza, que o caminhão que percorre a seqüência do poço 1 para o poço 2 atinge o poço 7 na etapa seguinte da trajetória.

O conjunto de itens de uma aplicação pode ser definido da seguinte forma: Seja $I = \{i_1, i_2, i_3, ..., i_n\}$, tal que $n \geq 1$ e i_n é um produto ou serviço automotivo de uma base de dados D.

A representação matemática de uma regra de associação é expressa na forma: $X \Rightarrow Y$, onde X e Y são conjuntos de itens pertencentes à base de dados em questão. X é chamado antecedente e Y conseqüente da regra.

O grau de certeza de uma regra é definido a partir de dois índices:

- O fator de suporte que é a freqüência com que um item distinto pode aparecer na base de dados.

- O fator de confiança que é a percentagem em que uma regra de associação é válida no repositório de dados.

Em termos formais, podemos definir o fator de suporte e o fator de confiança como sendo:

Fator de Suporte = $| X \cup Y | / n$, onde n é o número total de transações; $|X \cup Y|$ é o número de transações que contém os itens dos conjuntos X e Y.

Fator de Confiança = $|X \cup Y| / |X|$, onde $|X \cup Y|$ é o número de transações que contém os itens dos conjuntos X e Y e $|X|$ é o número de transações onde ocorrem os itens de X.

Para se extrair conhecimento através de regras de associação, é preciso efetuar dois passos. O primeiro deles é verificar o conjunto de itens que têm fator de suporte superior ao informado pelo usuário. O passo seguinte procura analisar se o fator de confiança deste conjunto resultante é superior ao fator de confiança informado pelo usuário [37]. É através destes parâmetros que avaliamos a exatidão do conhecimento extraído pela Mineração de Dados.

Em [20], é apresentado um exemplo de como uma regra de associação pode representar conhecimento de uma base transacional. Este exemplo é mostrado na tabela 4.1 e refere-se a um conjunto de transações de uma padaria. Na tabela 4.1 a coluna 1 mostra 10 transações (compras) e as colunas 2 a 8 indicam se o produto associado foi ou não adquirido em cada transação. Os valores de fator de suporte, chamado FSup, e fator de confiança FConf foram respectivamente iguais a 0.3 e 0.8 previamente fixados pelo usuário. A tabela 4.2 ilustra as regras extraídas da tabela 4.1.

ID	LEITE	CAFÉ	CERVEJA	PÃO	MANTEIGA	ARROZ	FEIJÃO
1	Não	Sim	Não	Sim	Sim	Não	Não
2	Sim	Não	Sim	Sim	Sim	Sim	Não
3	Não	Sim	Não	Sim	Sim	Não	Não
4	Sim	Sim	Não	Sim	Sim	Não	Não
5	Não	Não	Sim	Não	Não	Não	Não
6	Não	Não	Não	Não	Sim	Não	Não
7	Não	Não	Não	Sim	Não	Não	Não
8	Não	Não	Não	Não	Não	Não	Sim
9	Não	Não	Não	Não	Não	Sim	Sim
10	Não	Não	Não	Não	Não	Sim	Não

Tabela 4.1: Exemplo de uma base transacional de uma padaria.

Conjunto de itens freqüentes: Café, Pão. FSup= 3/10 = 0,3 Regra: se (X=Café) então Y=Pão . FConf= 3/3 = 1
Conjunto de itens freqüentes: Café, Manteiga. FSup = 3/10 = 0,3 Regra: se X=Café então Y=Manteiga. FConf = 3/3 = 1
Conjunto de itens freqüentes: Pão, Manteiga. FSup = 4/10 = 0,4 Regra: Se X=Pão então Y=Manteiga. FConf = 4/5 = 0,8
Conjunto de itens freqüentes: Café, Pão, Manteiga. FSup = 3/10 = 0,3 Regras: Se X=(Café e Pão) então Y=Manteiga. FConf = 3/3 = 1 Regras: Se X=(Café e Manteiga) então Y=Pão. FConf = 3/3 = 1 Regras: Se X=(Café) então Y=(Pão e Manteiga). FConf = 3/3 = 1

Tabela 4.2: Regras extraídas da Base de Dados da Tabela 4.1

O algoritmo Apriori foi desenvolvido por Agrawal [2]. Este algoritmo minerador procura extrair conhecimento de um conjunto de dados especial chamado *Basket Data*. Em um *Basket Data*, uma transação consiste num conjunto de atributos chamados itens. Cada item pode assumir o valor verdadeiro ou falso, se as características que estes itens representam estiverem presentes ou ausentes, respectivamente, nesta transação.

Fazendo uma analogia ao problema RUMP, cada indivíduo (solução) corresponderia a uma transação e cada poço (vértice) estaria associado a um item. Portanto, numa transação do RUMP, um item (poço) pode assumir valor "SIM" se estiver presente na solução e um valor "NÃO" se não estiver presente.

A idéia central de utilizar a MD em nossa proposta consiste, em procurar identificar os poços que estão presentes nos melhores indivíduos. A partir da descoberta, o objetivo é tentar inseri-los nos indivíduos das populações seguintes, caso estas inclusões melhorem a qualidade das soluções destes indivíduos. Além disso, através de uma modificação no Apriori, também consideraremos a ordem dos poços nas soluções.

A ordem dos poços é importante, pois uma seqüência de visitas de poços em uma dada rota pode produzir um tempo de percurso diferente. Vejamos um exemplo, de acordo com a Figura 2.3, o tempo consumido para uma UMP percorrer as seqüências {1, 5, 2, 3} e {1, 3, 2, 5} seria respectivamente de 386 e 356 minutos.

Para implementar estas idéias, noções de esquemas poderiam ser usadas, onde parte dos genes de algum indivíduo de um AG seria fixada [16]. Entretanto, poucas contribuições práticas existem, por causa da dificuldade da definição de uma estratégia para fixar um gene num valor constante, a fim de melhorar a convergência de um algoritmo.

Resumidamente, o algoritmo Apriori calcula, inicialmente, o suporte de todos os conjuntos de tamanho 1 e elimina aqueles que ficaram abaixo de um valor mínimo chamado de "suporte mínimo". A partir dos conjuntos freqüentes de tamanho 1, a cada passo forma-se um conjunto freqüente de itens de tamanhos maiores os quais formarão novos conjuntos de itens candidatos. Enquanto esse conjunto não se tornar vazio, o algoritmo armazena estes conjuntos candidatos em uma árvore *hash* e para cada transação do banco de dados, verifica-se se o conjunto candidatos está ou não contido na transação. Se um conjunto candidato estiver contido na transação, então se incrementa um contador para este item candidato. Ao final do teste, eliminamos os candidatos que não conseguiram suporte superior ao mínimo. Os conjuntos restantes são incluídos no conjunto que será usado no próximo passo.

O princípio deste algoritmo considera duas propriedades que permitem encontrar de forma mais eficiente os conjuntos freqüentes, reduzindo o espaço amostral.

- **Propriedade 1:** Todo conjunto que contém um subconjunto não freqüente também não é freqüente. Tomando como exemplo os dados da tabela 4.1, {Leite, Café, Pão} não é conjunto freqüente, pois {Leite, Café} é conjunto não freqüente, já que possui suporte 0.2;
- **Propriedade 2:** Todo subconjunto de um conjunto freqüente é freqüente. Se o conjunto {Café, Pão, Manteiga} é freqüente, então os subconjuntos {Café, Pão}, {Café, Manteiga} e {Pão, Manteiga} também serão freqüentes.

A figura 4.1 ilustra o pseudo-código do algoritmo Apriori.

```
1:  k=1;
2:  f1 = conjuntos freqüentes de tamanho 1;
3:  enquanto (fk ≠ 0) faça
4:      k = k+1;
5:      Gerar Ck (todos os conjuntos candidatos de tamanho k) a partir de fk-1;
6:      para cada transação t pertencente a base de dados faça
7:            para cada conjunto de candidato c em Ck faça
8:                  se todos os itens de c pertencem a t então
9:                        incrementar o contador associado a c;
10:                 fim se;
11:           fim para;
12:    fim para;
13:    fk = conjuntos candidatos pertencentes a Ck com suporte ≥ SupMin;
14: Fim enquanto;
15: resposta = união de todos os conjuntos fk;
```

Figura 4.1: Ilustra a codificação do algoritmo Apriori.

Na Figura 4.1, na linha 1, a variável k recebe o valor 1 indicando que o algoritmo irá percorrer a base de dados pela primeira vez em busca de regras com um único item no conseqüente. Na linha 2, o algoritmo Apriori procura identificar os conjuntos de itens que possuem freqüência superior ao suporte mínimo. Este conjunto recebe o nome de "conjunto freqüente". Na linha 3, entramos num laço de repetição que será executado até que o conjunto de itens freqüentes seja vazio. Na linha 4, incrementamos o valor de k, o que indica que vamos para mais uma varredura no banco de dados a procura de regras com k itens em seu antecedente. Em cada iteração $k \geq 2$, os itens candidatos são gerados a partir dos itens freqüentes da varredura anterior (f_{k-1}) (linha 5). Das linhas 6 à 9, a base de dados é percorrida de maneira a atualizar o contador de cada item candidato de acordo com cada transação do banco de dados. Na linha 13, eliminamos os itens candidatos que possuírem suporte inferior ao suporte mínimo. Finalmente, na linha 15, são informados como resultado computacional, as regras de associação que foram resultantes da união de todos os conjuntos f_k.

Capítulo 5 - Algoritmos propostos

O objetivo principal desta dissertação é o de apresentar propostas para melhorar o desempenho dos Algoritmos Evolutivos (AEs) e, mais especificamente, dos Algoritmos Genéticos (AGs).

Para isto, partimos de duas limitações tradicionais dos AGs. A primeira refere-se à dificuldade dos AGs em efetuar uma busca local eficaz e a segunda, aos tempos computacionais exigidos pelos AGs, normalmente maiores que os exigidos por outras heurísticas tradicionais de construção e busca local e até mesmo maiores que os tempos exigidos por outras metaheurísticas como GRASP, VNS e Busca Tabu.

Para tornar os AGs mais eficientes em relação à qualidade de solução propomos um AG Básico (AGB) utilizando técnicas presentes em modelos clássicos de AGs, como descritas no capítulo 3 e também desenvolvemos variações a partir do AGB. A primeira versão denotada por AG substitui o operador *crossover* do AGB por uma heurística do vizinho mais próximo. A segunda versão incorpora ao AG um módulo de busca local (AG+BL). Uma terceira versão incorpora ao AG+BL um módulo de mineração de dados (AG+BL+MD).

Numa segunda etapa, a fim de reduzirmos os tempos computacionais, propomos oito versões de algoritmos genéticos paralelos, chamadas de AGP1, AGP2, AGP3, AGP4, AGP5, AGP6, AGP7 e AGP8.

As versões paralelas AGP1, AGP3, AGP5 e AGP7 são versões paralelas do AGB, AG, AG + BL e AG+BL+MD respectivamente, que empregam um modelo sem troca de informação entre processos.

Também implementamos versões paralelas do AGB, AG, AG + BL e AG+BL+MD com troca de informação que foram chamadas AGP2, AGP4, AGP6 e AGP8, respectivamente. Nestes algoritmos, em determinadas gerações, as melhores soluções são migradas entre os processadores, a fim de tentar obter soluções de melhores qualidades utilizando de tempos em tempos, as informações relevantes obtidas por cada processo.

Neste capítulo, procuramos detalhar cada uma destas versões e seus componentes aplicados ao problema RUMP descrito no capítulo 2.

5.1 Algoritmos Seqüenciais

Os quatro algoritmos seqüenciais serão descritos um a um nos próximos segmentos.

5.1.1 Algoritmo 1 –AGB

Este modelo é baseado na descrição de um AG descrito no capítulo 3, portanto, os métodos de reprodução (*crossover* de um único ponto) e mutação, foram descritos na seção 3.2.4.

A geração da população inicial do AGB é bastante simples. A escolha de poços que farão parte da solução é realizada de maneira aleatória.

A função de aptidão de uma solução é medida pela função objetivo do problema. No nosso problema, o objetivo é maximizar o volume do óleo coletado nos poços visitados sem violar a restrição de tempo máximo para concluir uma rota. A cada vértice (poço) que é incluído em um indivíduo, nossa função armazena o tempo necessário para percorrer a solução. Todos os indivíduos que tiverem um tempo de percurso total da rota superior a 480 min. (8h diárias), serão considerados inviáveis e, portanto, serão descartados pelo algoritmo.

O AGB utiliza o número máximo de gerações como critério de parada do algoritmo. A cada geração, a melhor solução da atual população é comparada com *best*, o melhor indivíduo gerado pelo AGB até o momento e o *best* será atualizado se for o caso.

5.1.2 Algoritmo 2 –AG

A partir desta versão, descrevemos apenas as modificações em relação à versão anterior.

Ao contrário do AGB, para gerar um conjunto de soluções viáveis iniciais é usado um critério de prioridades de visitas a cada poço j ainda não visitado, com base nas informações sobre o volume acumulado de óleo (vazão(j)) e tempo de percurso do poço j ao poço i incluído, mais recentemente, na solução parcial (tempo (i,j)).

Assim, na construção de uma solução, considere como sendo i o poço incluído mais recentemente na solução (sempre inicializamos uma solução com a origem i = 0). A partir daí, selecionamos uma lista de candidatos (LC) composta de todos os poços j ainda não selecionados.

A prioridade de escolha de um poço j, a partir de um poço já selecionado i, é medida pelo quociente [vazão (j) / tempo (i,j)], ou seja, uma estratégia gulosa.

Como o tamanho desta lista pode ser muito elevado, selecionamos uma lista restrita de candidatos (LRC) composta dos k melhores candidatos da LC (onde k é um parâmetro de entrada). Em seguida, escolhemos aleatoriamente um elemento da LRC para ser incluído na atual solução parcial.

Este procedimento é repetido gerando novas listas LC e LRC até que uma solução completa seja alcançada. A escolha aleatória em LRC permite que diferentes soluções sejam geradas por esta técnica.

A função de avaliação é idêntica ao do AGB.

Em testes preliminares realizados com o AGB, observamos que o desempenho de operadores tradicionais como a mutação e *crossover* produzem resultados que poderiam ser melhorados.

Desta forma, continuamos a utilizar a mutação do AGB, porém uma nova heurística baseada em conceitos de vizinho mais próximo (VMP) foi empregada. A mutação é usada num percentual pequeno (cerca de 10% de uma população pode sofrer mutação) . O operador VMP gera uma solução (filho) a partir de p ($p \geq 2$ é um dado de entrada) soluções da população atual (pais). Escolhidos os p pais, construímos para cada vértice k presente em pelo menos um pai, duas listas: a primeira lista, viz(k), é a lista dos n vizinhos mais próximos de k. Uma segunda, LA(k), é a lista dos vértices adjacentes de k nas p soluções. O processo de gerar uma solução filho, inicializado na origem i = 0 e a seqüência de vértices pertinentes a sua rota, é obtido da forma descrita a seguir. Chamamos o vértice alocado mais recentemente na solução filho de *current*. Verificamos na lista viz(*current*) de *current*, o vértice j mais próximo daquele e pertencente a LA(k); Se este já estiver presente na solução parcial, selecionamos o próximo vértice até a lista LA(j) se esvaziar. Se a lista LA(j) se esvaziar, selecionamos um vértice ainda não alocado aleatoriamente em viz (i).

Uma alternativa para gerarmos mais de um filho neste procedimento, é adotar uma lista de candidatos restrita composta (LCR) dos s vizinhos mais próximos de *current* (onde s é um parâmetro de entrada), e destes selecionar um, aleatoriamente. Esta escolha aleatória permite, a partir de p soluções pais, gerar vários filhos distintos.

Vejamos o seguinte exemplo, suponha que p = 2 e os indivíduos pais p_1 e p_2 gerem o_1, onde:

$$p_1 = \text{ETO} - 22 - 12 - 45 - \text{ETO} \quad \text{e} \quad p_2 = \text{ETO} - 14 - 4 - 19 - \text{ETO}$$

A determinação do filho o_1 segue os seguintes passos: Para cada poço i, onde inicialmente i = ETO, verificamos se entre os n = 10 vizinhos mais próximos de i = ETO, o poço 22 ou 14 que compõem a lista LA (ETO) se encontra presente. Escolhe-se aquele que tiver maior média, conforme fórmula [vazão (j) / tempo (i,j)]. Se nenhum deles estiver em viz(i), escolhe-se um poço de viz(i) aleatoriamente.

O AG utiliza o critério de parada idêntico ao do AGB.

5.1.3 Algoritmo 3 – AG + BL

Para esta versão, utilizamos o algoritmo AG e adicionamos um módulo de Busca Local, onde somente o melhor indivíduo de cada geração será considerado a solução base para esse procedimento.

O mecanismo de busca local aqui proposto funciona da seguinte forma. Dada uma solução viável do problema (solução base), para cada vértice desta solução criamos uma lista dos r vizinhos mais próximos (r é um dado de entrada) e que não estejam presentes na atual solução. A seguir, a partir da solução base, efetuamos uma substituição de um vértice por vez, trocando o da solução por um outro da sua lista de adjacências. Avaliamos a sua aptidão comparando (atualizamos, se for o caso) com a melhor solução alcançada até o momento pelo AG (*best)* e retornamos à solução base e efetuamos outra troca. Após varrer as alternativas deste vértice, partimos para a lista de adjacências do próximo vértice da atual solução.

O módulo de busca local é ativado sempre que o AG gerar uma solução promissora (5 % de alteração do volume coletado pelo melhor indivíduo até o momento).

5.1.4 Algoritmo 4 – AG + BL + MD

Nesta versão, utilizamos o algoritmo AG+BL incluindo um módulo de Mineração de Dados.

O módulo de mineração de dados foi incluído no algoritmo AG+BL com o objetivo de tentar produzir uma solução de melhor qualidade. O módulo usa como base o algoritmo Apriori proposto por Rakesh Agrawal [2], com algumas adaptações que o tornaram mais eficiente para o RUMP. A implementação do algoritmo Apriori foi feita por Cristiane Norbiato Targa e aplicada em sua defesa de tese recentemente [68].

A extração de conhecimento se dá em um sub conjunto de soluções denominado conjunto elite, composta pelos k melhores indivíduos, onde k é um parâmetro de entrada.

As adaptações no algoritmo Apriori permitem que o algoritmo de mineração reconheça as seqüências de poços que mais aparecem no conjunto elite de soluções viáveis, além de permitir que em cada etapa da mineração de dados sejam geradas todas as combinações de *itemsets* distintos.

Cada *itemset* freqüente gerado corresponde a uma seqüência de poços presente no conjunto elite, e que satisfazem o suporte fornecido como parâmetro de entrada do programa.

O processo de mineração de dados é executado, sempre que atingimos um percentual igual ou superior a 40% de alterações nas soluções que fazem parte do conjunto elite. Conjunto este que é atualizado pelas iterações do AG+BL+MD.

Numa etapa seguinte, após a execução do módulo de mineração, cada solução da população corrente sofre uma tentativa de inclusão de cada *itemset* freqüente gerado. O *itemset* freqüente somente é incluído, se nenhum dos poços que fazem parte do *itemset* freqüente estiverem presentes no indivíduo. E, caso não proporcione uma melhoria na aptidão desta solução, restaura-se o estado anterior da solução e tenta-se incluir o *itemset* freqüente seguinte.

Um *itemset* freqüente é inserido na melhor posição da rota atual, ou seja, na posição que produzir a melhor avaliação, considerando-se todos os pares de vértices da solução. Eventualmente, com a inclusão de um *itemset* freqüente num indivíduo, alguns vértices podem ser descartados para atender às restrições do tempo limite do problema.

5.2 Algoritmos Paralelos

A partir das versões seqüenciais de AGB, AG, AG+BL e AG+BL+MD desenvolvemos versões paralelas que serão descritas nas próximas seções.

Em nosso trabalho, o modelo que adotamos foi o Mestre-Escravo. A população foi dividida em sub-populações de acordo com o número de processadores envolvidos. Cada processador realiza cálculos em sua sub-população, de maneira a trabalhar com um conjunto menor de indivíduos. Assim que a tarefa de cada processador terminar, ele transmite a melhor solução encontrada para o processador Mestre. O processador Mestre ao receber a melhor solução de cada nó, elege a melhor solução global.

Em especial, a versão paralela do AG+BL+MD não seguiu o modelo mestre-escravo, mas sim um modelo descentralizado. A justificativa desta escolha se deve a um aumento na variação no tempo consumido por cada geração. Pois, numa determinada geração, podem existir processos que não executam a mineração de dados, enquanto que outros podem realizar tal tarefa, obrigando o processo mestre a esperar, às vezes, por longos períodos de tempo até que todos processos escravos tenham terminado a computação daquela geração. Já o sistema descentralizado, trabalha da seguinte forma: A cada 25 gerações, cada processador analisa se conseguiu melhorar a melhor solução até o momento. Em caso afirmativo, envia o indivíduo *best* para os demais processadores, propagando assim sua descoberta. Ao terminar sua computação, o processador propaga que terminou sua tarefa informando o tempo gasto de sua computação. Quando todos os processadores tiverem propagado o fim da computação, um processo analisa o maior tempo e imprime a melhor solução encontrada e o tempo computacional.

5.2.1 Algoritmo 1 - AGP1 (AGB Paralelo sem Comunicação)

Este modelo é uma versão paralela do AGB sem comunicação entre os processadores. O modelo de paralelismo empregado é o Modelo Mestre-Escravo. Neste modelo, é importante ressaltar que o processador Mestre também executa o algoritmo genético. Ao atingir o critério de parada, todos os processadores escravos enviam as

suas melhores soluções ao Mestre, que as analisa e decide pela melhor solução como a solução do algoritmo.

Os processadores escravos realizam a computação genética na tentativa de especializar sua sub-população em busca da solução do problema aplicado. A população de um algoritmo genético paralelo (AGP) é dividida em sub-populações igualitariamente ao número de processadores disponíveis para rodar a aplicação.

5.2.2 Algoritmo 2 – AGP2 (AGB Paralelo com Comunicação)

Este algoritmo é análogo ao AGP1, diferindo em apenas um aspecto. O AGP2 realiza uma migração da melhor solução de um determinado processador entre os demais processadores a cada 25 gerações. O processador Mestre é responsável pela análise e propagação destas soluções, além de realizar tarefas computacionais referentes ao algoritmo evolutivo.

Os processos escravos ao receberem a melhor solução propagada pelo mestre, passam a gerar seus descendentes a partir desta nova informação.

5.2.3 Algoritmo 3 - AGP3 (AG Paralelo sem Comunicação)

Para esta versão, utilizamos como algoritmo base o algoritmo seqüencial AG e a estratégia de paralelismo aplicada é a mesma aplicada ao AGP1.

5.2.4 Algoritmo 4 - AGP4 (AG Paralelo com Comunicação)

Esta versão faz uso do AG, mas com comunicação entre os processadores. A estratégia de paralelismo empregada é a do AGP2.

5.2.5 Algoritmo 5 - AGP5 (AG + BL Paralelo sem Comunicação)

Este algoritmo paralelo incorpora os módulos AG com módulos de busca local, utilizado em AG+BL. Cada processador atua em um subconjunto da população, utilizando as mesmas estratégias do algoritmo seqüencial AG+BL. O modelo de paralelismo empregado é idêntico ao usado no algoritmo AGP1.

5.2.6 Algoritmo 6 – AGP6 (AG + BL Paralelo com Comunicação)

Esta versão é análoga ao AGP5, mas neste caso, os processadores se comunicam migrando as melhores soluções, como descrito no modelo AGP2.

5.2.7 Algoritmo 7– AGP7 (AG+BL+MD Paralelo sem Comunicação)

Este modelo é a versão paralela do AG+BL+MD sem comunicação. A estratégia de paralelismo empregada permite que cada processador propague sua melhor solução aos demais processadores, ao atingir o fim da sua computação.

5.2.8 Algoritmo 8– AGP8 (AG+BL+MD Paralelo com Comunicação)

Este modelo é a versão paralela com comunicação do algoritmo AG+BL+MD. A estratégia de comunicação permite que a cada 25 gerações, um processador ao constatar que houve melhoria na qualidade de solução obtida até o momento, propague a nova solução encontrada para os demais processadores. Como neste modelo só há propagação quando ocorre melhoria, o número de mensagens recebidas é variável e, portanto, houve a necessidade de implementar um algoritmo de terminação.

No algoritmo de terminação, cada processador possui um vetor que retrata os estados dos processos envolvidos. Cada elemento deste vetor indica se o processo atingiu o estado de terminação local ou não. Inicialmente todas as entradas são preenchidas com zero (0), indicando que nenhum processo terminou sua computação. À medida que alguns processos vão terminando sua computação local, eles alteram o valor correspondente a sua entrada no vetor de estados, colocando o valor de tempo computacional consumido por ele. Em seguida, ele propaga uma mensagem aos vizinhos com o valor do tempo computacional transcorrido desde o início da sua execução. Cada processo ao receber a mensagem de terminação, atualiza a entrada do processo remetente com o valor do tempo computacional enviada por ele. Ao final, quando todas as entradas estiverem preenchidas com valores diferentes de zero (0), a aplicação é finalizada.

5.3 Exemplo de execução

Para ilustrar o funcionamento dos algoritmos propostos na resolução do problema RUMP, elaboramos um programa denominado TRUMPS (*Trace Route for Unit Mobile Piston System*).

Tomamos como exemplo, o grafo da Figura 2.2 que retrata um campo petrolífero (fictício) com 7 poços. Cada aresta possui um peso associado, que representa a distância entre os poços e o tempo para percorrer a estrada de um poço ao outro. O algoritmo armazena o grafo em uma matriz de distâncias de maneira simétrica.

Para generalizar o emprego do TRUMPS, torna-se necessário converter este grafo direcionado para um grafo completo e não direcionado de distâncias mínimas. Isto é feito aplicando-se o algoritmo de Dijkstra [30] onde, a transformação do grafo direcionado para o grafo não direcionado completo, conta com a introdução de uma aresta do poço de destino para o poço de origem com o mesmo peso da aresta do poço de origem para o poço de destino. A matriz de distância do grafo resultante são mostrados na Figura 5.1 e Tabela 5.1.

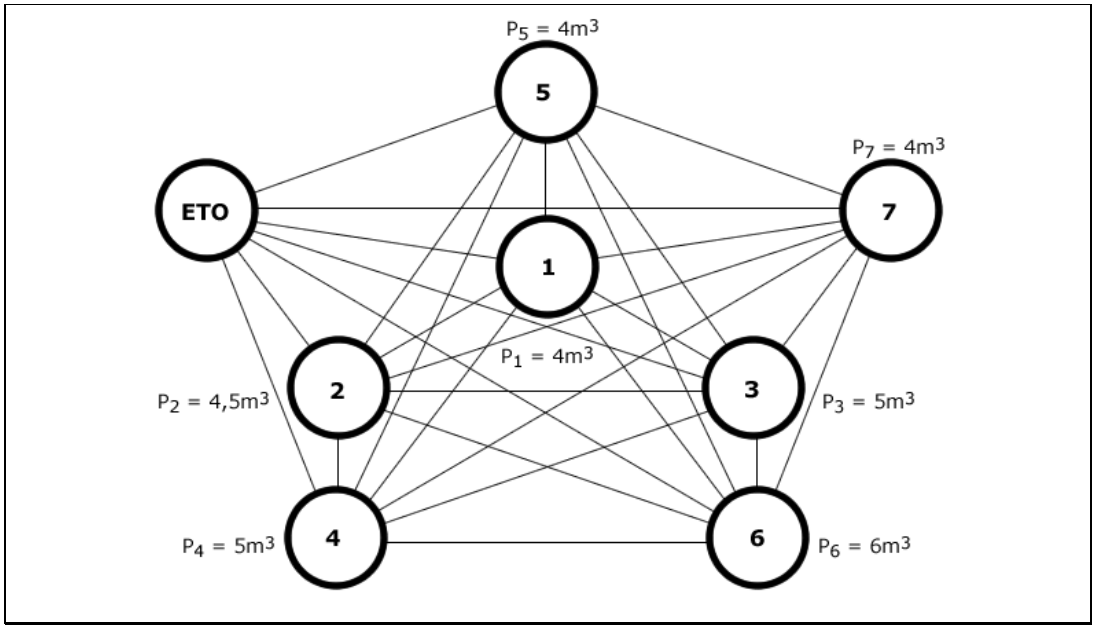

Figura 5.1: Grafo ilustrativo de um problema RUMP.

Poços	ETO	1	2	3	4	5	6	7
ETO	0	15	65	140	17	66	48	79
1	15	0	80	125	32	81	63	94
2	65	80	0	115	48	35	17	48
3	140	125	115	0	157	150	132	163
4	14	32	48	157	0	49	31	62
5	66	81	35	150	49	0	18	13
6	48	63	17	132	31	18	0	31
7	79	94	48	163	62	13	31	0

Tabela 5.1: Matriz de distância associado ao grafo da figura 5.1

A partir da tabela 5.1, podemos calcular o tempo e a distância gastos numa dada trajetória. Nesta fase, uma população de indivíduos é criada e analisada.

A população é definida como uma matriz de número de linhas igual a um parâmetro de entrada. A cada gene, ou seja, a cada elemento da matriz é atribuído um inteiro não negativo que recebe o valor zero (0) quando o poço não participa da solução e um valor positivo e inteiro, em caso contrário, onde o inteiro representa o poço a ser visitado na rota codificada dentro do indivíduo.

Vejamos o exemplo abaixo. Considere o indivíduo segundo a representação abaixo, que pode ser uma linha qualquer da matriz de população.

0	6	2	1	3	4

Esta representação indica que o poço 0 é a origem, o poço 6 é o 1º poço a ser visitado, o poço 2 é o 2º poço a ser visitado, o poço 1 é o 3º poço a ser visitado, o poço 3 é o 4º poço a ser visitado, o poço 4 é o 5º poço a ser visitado.

Este indivíduo define a seguinte rota: ETO – 6 – 2 – 1 – 3 – 4 – ETO

A aptidão do indivíduo pode ser calculada percorrendo-se a rota, e somando os valores dos tempos intermediários entre cada um dos segmentos que estão presentes em uma rota. Todos os indivíduos que atingirem uma soma superior a 480 minutos serão descartados e, portanto, serão considerados como soluções inviáveis para o problema.

Para o indivíduo, da representação acima, o somatório dos valores é igual a 441, o que significa que a rota será percorrida em 441 minutos e a distância será de 441 km.

O volume de óleo coletado pode ser obtido pelo somatório dos volumes de cada poço visitado no percurso. Para este indivíduo, o custo coletado foi 0+6+4.5+4+5+5+0 = 24.50 m^3. Um vetor de poços contém as informações sobre a capacidade de cada um dos poços. Neste algoritmo não é considerado o tempo de montagem e desmontagem do equipamento da UMP em cada poço da rota.

Capítulo 6 - Resultados Computacionais

Os algoritmos aqui propostos foram avaliados em testes computacionais com até 1000 vértices e os resultados computacionais obtidos são muito promissores; mostrando que os procedimentos propostos podem obter soluções de boas qualidades em tempos computacionais razoáveis.

Em nossos testes foram levados em consideração os seguintes parâmetros:

- O número de Gerações foi de 500 iterações;
- O tamanho da população foi de 200 indivíduos;
- A taxa de reprodução foi de 90% de reprodução (heurística utilizada em cada versão) e 10% por mutação;
- O tamanho da População Elite foi de 5 indivíduos;
- A taxa de execução para o módulo de MD foi de 30% dos indivíduos alterados no conjunto elite;
- A saturação da folha da árvore *hash* no algoritmo de mineração de dados (MD) foi de 500;
- A ordem da árvore *hash* no algoritmo de MD: 50;
- O suporte no algoritmo de MD: 40%;
- A taxa de execução para módulo de Busca Local: 5% de alteração do volume coletado pelo indivíduo *best* (melhor solução gerado até o momento pelo AG);

Os resultados computacionais extraídos foram baseados em dois objetivos:

- Maximizar o volume coletado (m^3);
- Manter o tempo de processamento a um nível sustentável (segundos);

Para atingir nossos dois objetivos, descritos no capítulo anterior, dividimos em duas frentes os testes computacionais a serem realizados.

A primeira frente procura maximizar a extração do volume de petróleo. Os testes foram efetuados num processador AMD ATHLON XP 1900 - 1.6GHz; Sistema Operacional: Windows 2000 Server; 1 GB de memória RAM DDR.

Para realizar os testes, foram projetadas dez instâncias do problema RUMP com 50, 70, 100, 120, 300, 500 e 1000 poços, totalizando com isto 70 instâncias. O tempo máximo de uma trajetória foi fixado em 480 minutos. Os resultados foram expressos em forma de tabelas e gráficos, onde cada célula representa o valor de volume ou tempo médio obtido nas 10 execuções de cada um dos 7 problemas testes.

A outra frente procura reduzir o tempo de processamento consumido pelos algoritmos seqüenciais. Para isto, utilizou-se um Cluster de 32 processadores do tipo THIN (RS 6000) mod. 390, AIX 4.1.5, com 256 MB RAM, desempenho de 266 MFlops. Todos os algoritmos seqüenciais e paralelos executados nesta segunda bateria de testes foram executados em modo exclusivo, usando a biblioteca MPI.

Nesta bateria de testes, foram utilizados os mesmos cinco primeiros problemas testes com 50, 70, 100 e 120 poços descritos anteriormente. O tempo máximo para a trajetória de uma rota permaneceu em 480 minutos, mas a distância (tempo) entre os poços variou em cada uma das instâncias aqui testadas. As versões AGB, AG e AG+BL+MD foram executadas em 4 e 8 processadores, enquanto que a AG+BL envolveu o uso de 4, 8, e 16 processadores. Esta limitação no número de processadores utilizados deveu-se às dificuldades por nós encontradas para efetuar estes testes computacionais. Os resultados médios de cada instância em relação ao tempo despendido bem como em relação à média dos valores da função objetivo são mostrados nas tabelas e gráficos das próximas seções.

Todos os algoritmos aqui propostos foram implementados na linguagem C (padrão ANSI). O algoritmo Apriori utilizado foi desenvolvido em [68].

6.1 Resultados de Algoritmos Seqüenciais

As figuras 6.1 à 6.7 mostram que o algoritmo AG+BL+MD obtém as melhores soluções em todas as instâncias. As versões AG e AG+BL aparecem com desempenho praticamente igual em segundo lugar, seguido da versão básica AGB que obteve as piores soluções. Em relação ao desempenho do AG e AG+BL, a similaridade nas suas soluções se deve provavelmente a um desempenho do operador heurístico VMP que, na maioria das vezes, já encontra ótimos locais no seu processo de busca e com isso, a presença do módulo BL fica praticamente desnecessária. Quanto ao fato do AG muitas vezes obter soluções melhores que o AG+BL, isso se deve às escolhas aleatórias num AG, ou seja, um mesmo AG executado 2 vezes pode produzir soluções totalmente distintas.

Este foi o principal motivo para, em cada problema teste com determinada dimensão, gerarmos 10 instâncias aleatoriamente deste problema. Outro aspecto que podemos notar é, que à medida em que as dimensões crescem, a superioridade da versão com mineração de dados fica mais evidente quando comparada com o desempenho das demais versões anteriores.

Podemos observar que os grafos das 5 primeiras instâncias (até 300 poços) tiveram uma coleta maior do que as instancias de 6 a 10. Isto acontece porque os grafos gerados para as instâncias de 1 a 5 apresentam custos menores entre poços de origem e poços de destino.

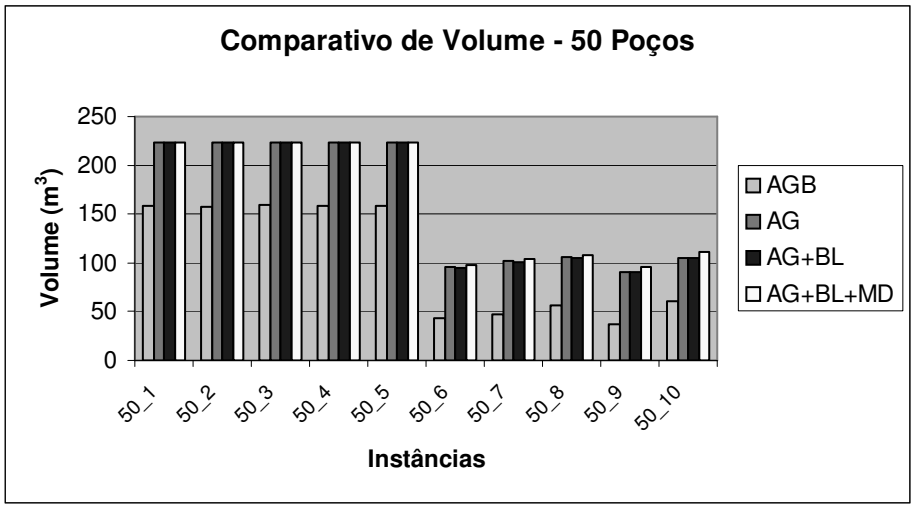

Figura 6.1: Soluções para instâncias de 50 vértices.

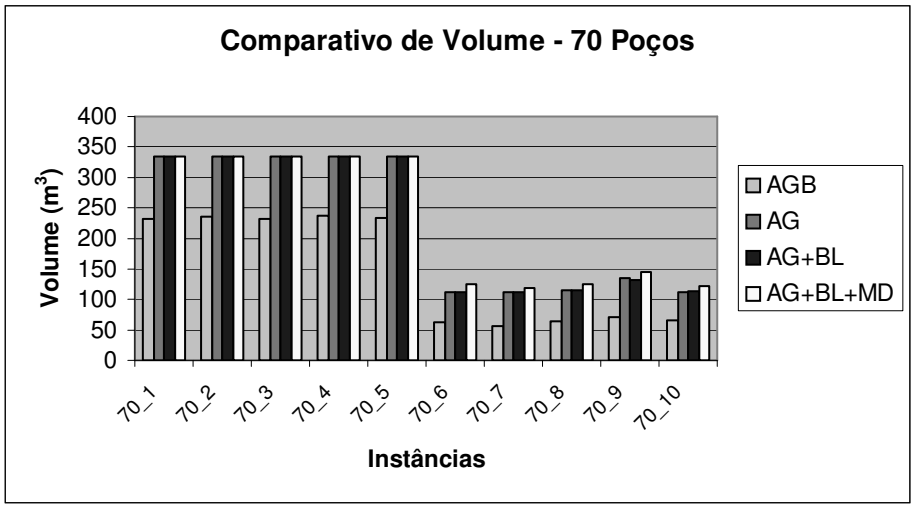

Figura 6.2: Soluções para instâncias de70 vértices.

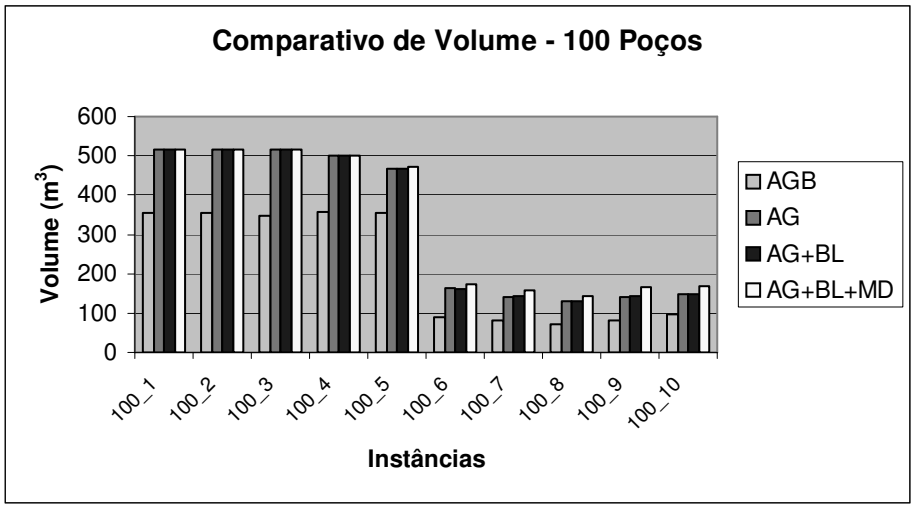

Figura 6.3: Soluções para instâncias de 100 vértices.

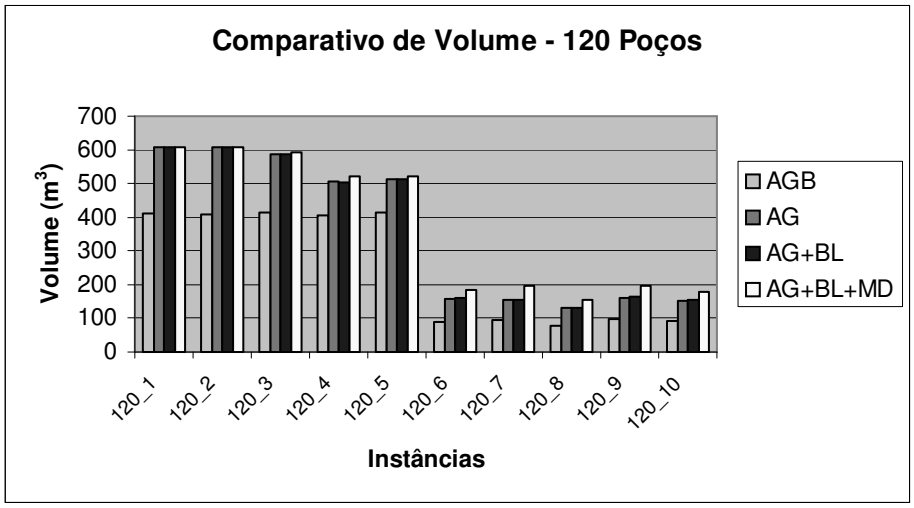

Figura 6.4: Soluções para instâncias de 120 vértices.

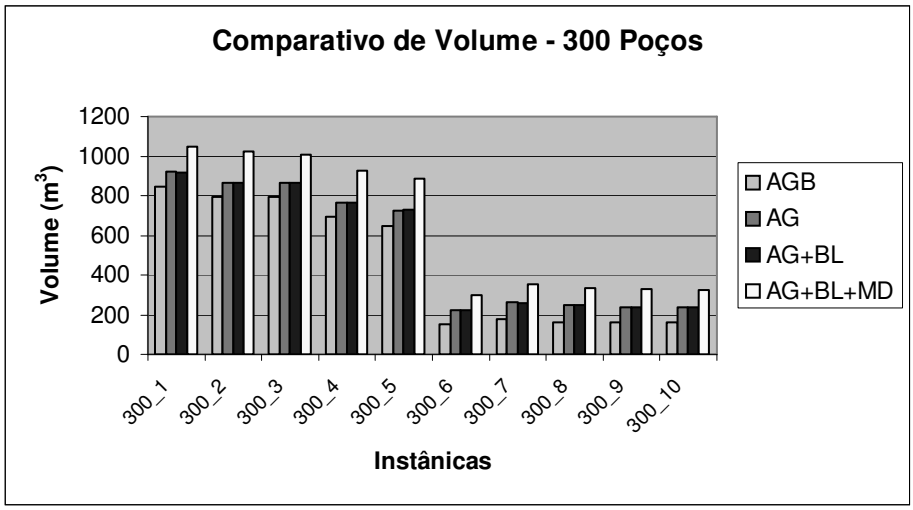

Figura 6.5: Soluções para instâncias de 300 vértices.

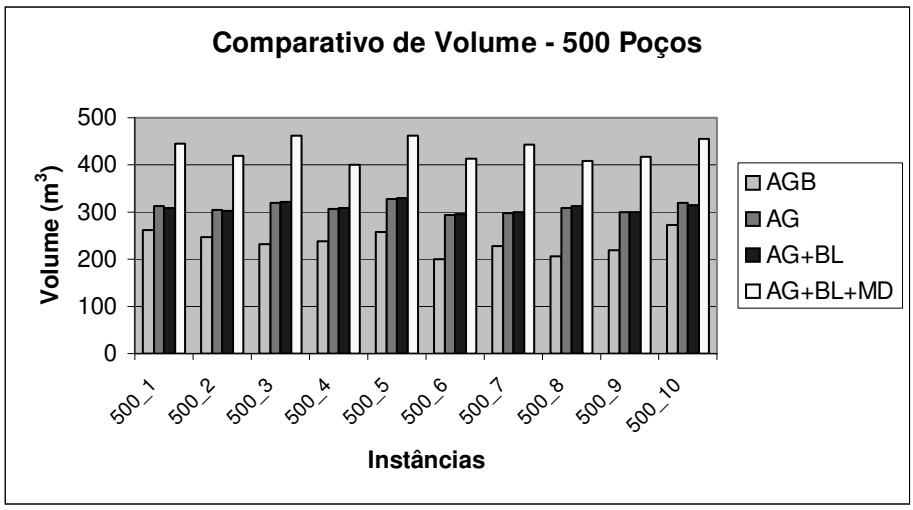

Figura 6.6: Soluções para instâncias de 500 vértices.

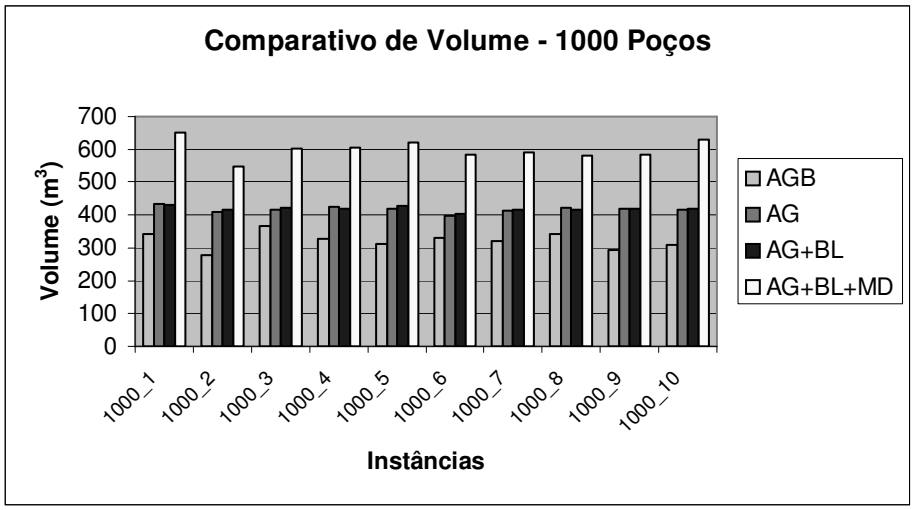

Figura 6.7: Soluções para instâncias de 1000 vértices.

Na Tabela 6.1, mostramos para cada instância, a "distância" entre a solução de cada algoritmo em relação à melhor solução obtida para a instância, considerando todos os algoritmos. O valor de cada célula corresponde à diferença percentual entre a melhor solução, considerando todos os algoritmos.

Volume	1	2	3	4	5	6	7	8	9	10
AGBASICO_50	29.14	29.57	28.71	28.58	29.78	55.27	54.81	47.17	61.18	45.7
AG_50	0	0	0	0	0	2	1.69	1.76	5.33	5.68
AGBL_50	0	0	0	0	0	2.55	3.02	2.27	5.44	5.86
AGBLMD_50	0	0	0	0	0	0	0	0	0	0
AGBASICO_70	30.31	29.42	30.51	29.17	29.85	50.51	53.29	47.87	51.34	46.46
AG_70	0	0	0	0	0.1	10.59	6.27	7.57	7.66	8.35
AGBL_70	0	0	0	0	0	10.41	6.65	7.18	9.19	7.48
AGBLMD_70	0	0	0	0	0	0	0	0	0	0
AGBASICO_100	30.97	30.94	32.44	28.55	24.95	48.92	48.81	49.14	50.61	42.41
AG_100	0	0	0	0.19	0.88	5.73	11.74	8.54	15.19	12.06
AGBL_100	0	0	0	0.2	1.26	7.16	10.57	9.19	13.98	12.2
AGBLMD_100	0	0	0	0	0	0	0	0	0	0
AGBASICO_120	32.54	32.92	29.99	21.85	20.65	51.21	52.17	50.48	50.34	47.88
AG_120	0	0.28	0.76	2.82	1.83	14.25	21.34	15.53	18.27	14.35
AGBL_120	0	0.16	0.9	3.2	2.05	13.14	21.76	16.64	16.64	13.71
AGBLMD_120	0	0	0	0	0	0	0	0		0
AGBASICO_300	19.27	22.2	21.23	25.38	27.01	49.55	49.94	51.5	50.52	50.06
AG_300	12.16	15.3	14.19	17.19	18.29	24.21	25.8	26.98	26.84	26.58
AGBL_300	12.79	15.35	14.08	17.51	17.84	24.22	26.13	26.96	27.75	26.4
AGBLMD_300	0	0	0	0	0	0	0	0	0	0
AGBASICO_500	41.08	41.07	49.81	40.41	44.2	51.58	48.59	49.79	47.15	40.29
AG_500	29.74	27.1	31.21	23.32	29.35	29.17	32.85	24.79	27.88	29.71
AGBL_500	30.57	27.87	30.46	22.97	28.7	28.68	32.27	23.54	28.07	30.74
AGBLMD_500	0	0	0	0	0	0	0	0	0	0
AGBASICO_1000	47.3	48.82	39.43	45.79	49.73	43.59	45.35	41.37	49.73	50.84
AG_1000	33.19	25.2	31.26	29.9	32.68	31.68	30.17	27.42	28.41	33.94
AGBL_1000	33.58	24.48	29.24	29.97	32.98	30.67	30.9	27.07	28.38	33.73
AGBLMD_1000	0	0	0	0	0	0	0	0	0	0

Tabela 6.1: Resultados comparativos entre as médias das soluções obtidas por cada algoritmo seqüencial.

Em outra análise, percebemos que o módulo de mineração de dados interferiu no tempo total execução do algoritmo evolutivo. As figuras 6.8 à 6.14 refletem a comparação de tempo total de processamento entre as versões dos AGs. O tempo computacional é expresso em segundos (s).

Pelos resultados das figuras 6.8 à 6.14 notamos que a melhora obtida pela versão AG+BL+MD possui seu preço, ou seja, a contrapartida desta versão, é um aumento do tempo computacional exigido que foi em média 113% maior que na versão AGB; 33.07% maior que na versão AG; 33.90% maior que na versão AG+BL.

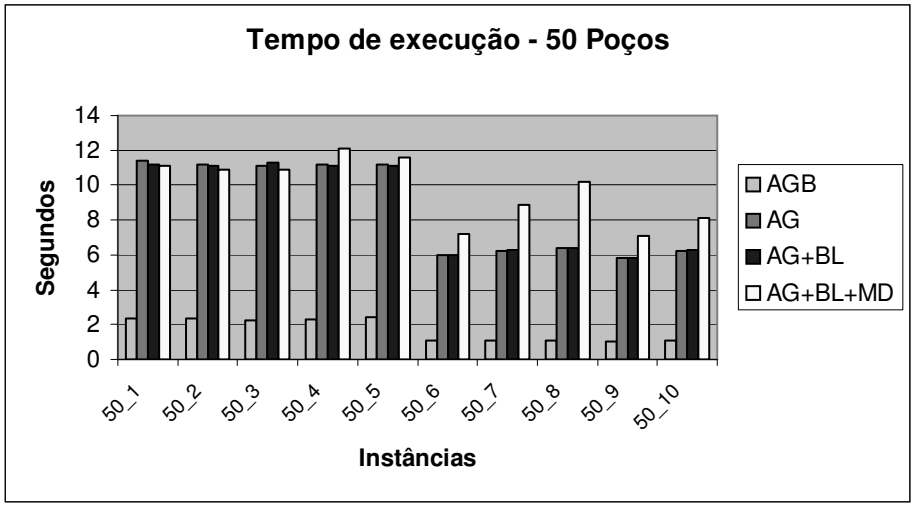

Figura 6.8: Tempo de execução para 50 vértices.

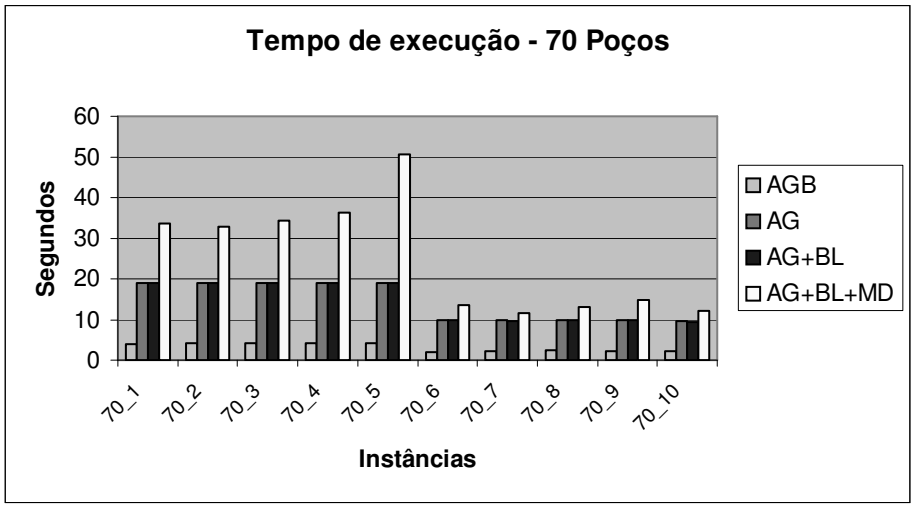

Figura 6.9: Tempo de execução para 70 vértices.

Figura 6.10: Tempo de execução para 100 vértices.

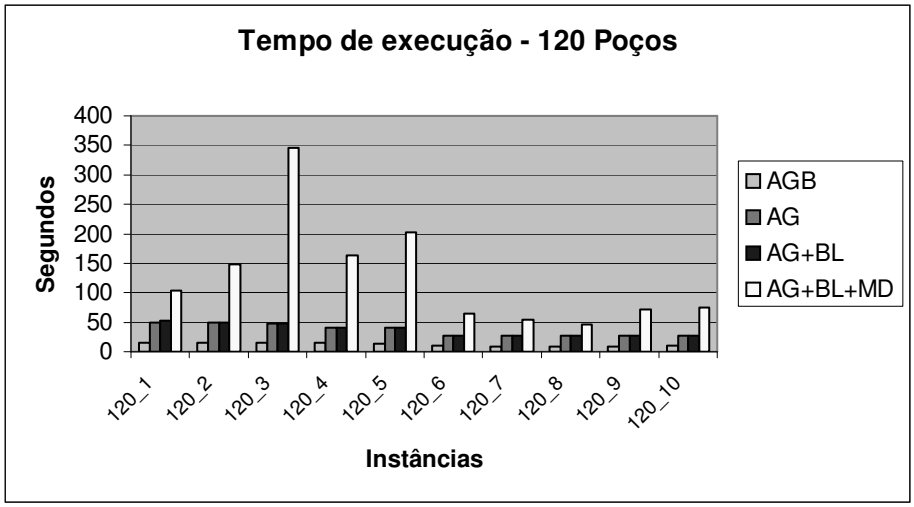

Figura 6.11: Tempo de execução para 120 vértices.

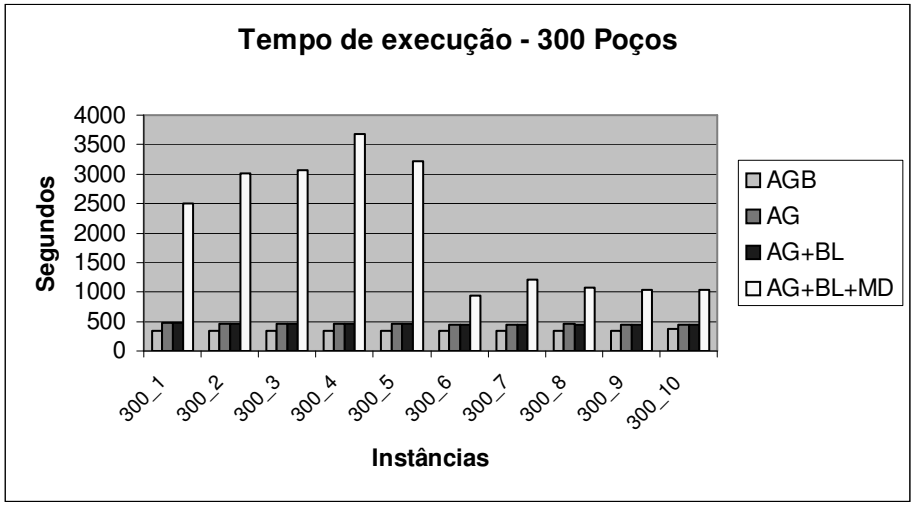

Figura 6.12: Tempo de execução para 300 vértices.

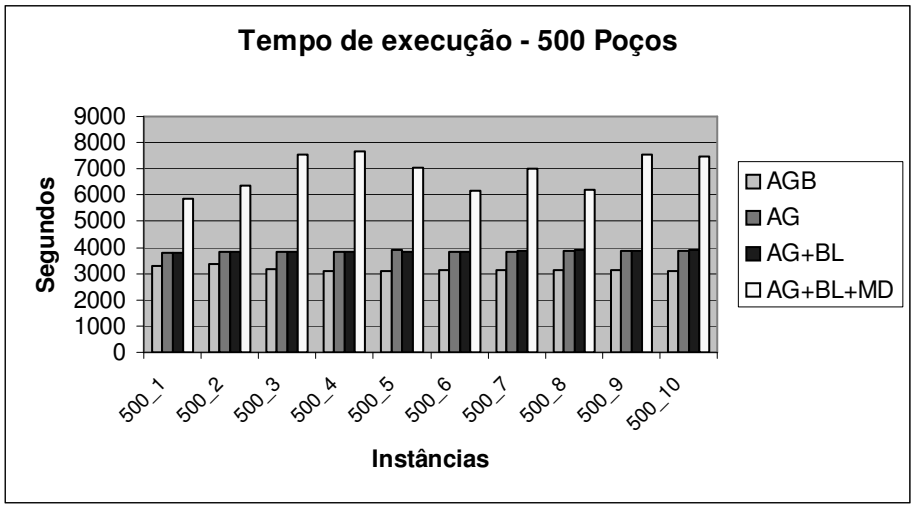

Figura 6.13: Tempo de execução para 500 vértices.

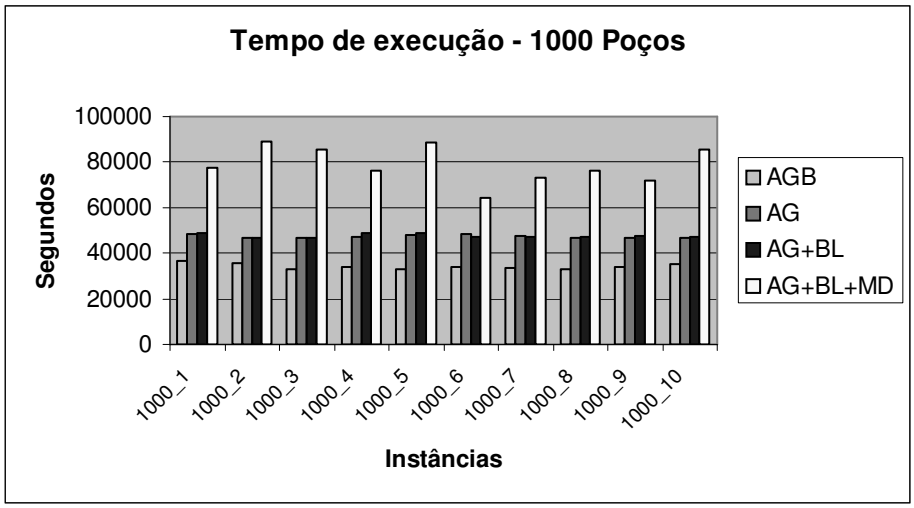

Figura 6.14: Tempo de execução para 1000 vértices.

Para tratar da questão de tentar manter a qualidade, mas reduzir os tempos computacionais exigidos, versões paralelas foram implementadas e os resultados computacionais são mostrados a seguir.

6.2 Resultados dos Algoritmos Paralelos

Nesta segunda bateria de testes computacionais, realizamos testes com 50, 70, 100 e 120 vértices e cada um deles foi executado em 4, 8 e 16 processadores para as versões com AG +BL; 4 e 8 processadores para AGB, AG e AG+BL+MD. Para manter a confiabilidade do nosso teste, executamos a versão seqüencial de cada um dos problemas descritos acima nesta máquina paralela, de modo a obter o *speed up* e a *eficiência* deste tipo de aplicação.

O *speed up* consiste em um parâmetro utilizado na medição da aceleração proporcionada pelos algoritmos paralelos quando comparamos com suas versões seqüenciais e a *eficiência* mede a fração média de tempo ao longo do qual cada processo permanece em efetiva execução. Desta forma, poderíamos definir o *Speed up* e a *Eficiência* como sendo:

Speed up (P) = tempo_seqüencial / tempo_paralelo(P)

e

Eficiência (P) = *Speed up* (P) / P, onde:

tempo_seqüencial é o tempo computacional do algoritmo seqüencial gasto no sistema distribuído; tempo_paralelo é o tempo computacional consumido pela versão paralela; e P é o número de processadores empregados.

Propomos neste trabalho, oito versões de algoritmos genéticos paralelos (AGP), que chamaremos de AGP1, AGP2, AGP3, AGP4, AGP5, AGP6, AGP7 e AGP8.

Algoritmo	Seqüencial	Comunicação
AGP1	AGB	Não
AGP2	AGB	Sim
AGP3	AG	Não
AGP4	AG	Sim
AGP5	AG+BL	Não
AGP6	AG+BL	Sim
AGP7	AG+BL+MD	Não
AGP8	AG+BL+MD	Sim

Tabela 6.2: Descrição dos algoritmos Paralelos

Como mostrado na Tabela 6.2, as versões AGP1, AGP3 e AGP5 se caracterizam por ser uma extensão do algoritmo evolutivo seqüencial convertido para executar em ambientes paralelos e, portanto, chamaremos de AGP sem comunicação.

O AGP sem comunicação divide a população de indivíduos entre os processadores e, portanto, cada um trabalha com uma quantidade menor de indivíduos, possibilitando uma melhor performance e rapidez.

A versão paralela com comunicação (AGP2, AGP4 e AGP6), troca as melhores soluções em determinadas gerações, a fim de obter uma melhor convergência sobre a solução do problema RUMP. Cada processo ao receber uma informação passa a gerar descendentes a partir dela, objetivando atingir soluções mais promissoras em um tempo computacional menor. A cada 25 gerações, cada um dos processadores escravos envia a sua melhor solução para o processador mestre, que decide qual delas é a melhor global. Com base nesta decisão, o mestre propaga a melhor solução global para os demais processadores que passam a gerar descendentes mais qualificados. Já as versões AGP 7 e AGP8 adotaram um modelo descentralizado de migração de indivíduos entre um processador e outro, ou seja, a cada 25 iterações, se houver melhoria na melhor solução cada processo envia esta melhor solução para cada um dos seus vizinhos, no caso todos os outros processos. (conforme descrito na seção 5.2.8).

Podemos notar através das figuras 6.15 à 6.22 que não existe uma diferença gritante de *speed up* e eficiência em relação às versões com comunicação e sem comunicação entre os processadores, se compararmos com as versões AGB, AG, AG+BL, AG+BL+MD. No entanto, o *speed up* e a eficiência dos processadores vão diminuindo à medida que a instância aumenta de tamanho. Outro ponto importante é que para uma mesma instância, o fato de dobrar o número de processadores não significa que o *speed up* ou a eficiência tenham seus valores alterados na mesma proporção.

Para maiores detalhes sobre os *speed ups* e as *eficiências* encontradas em cada algoritmo, consulte apêndice “A”, no capítulo 9.

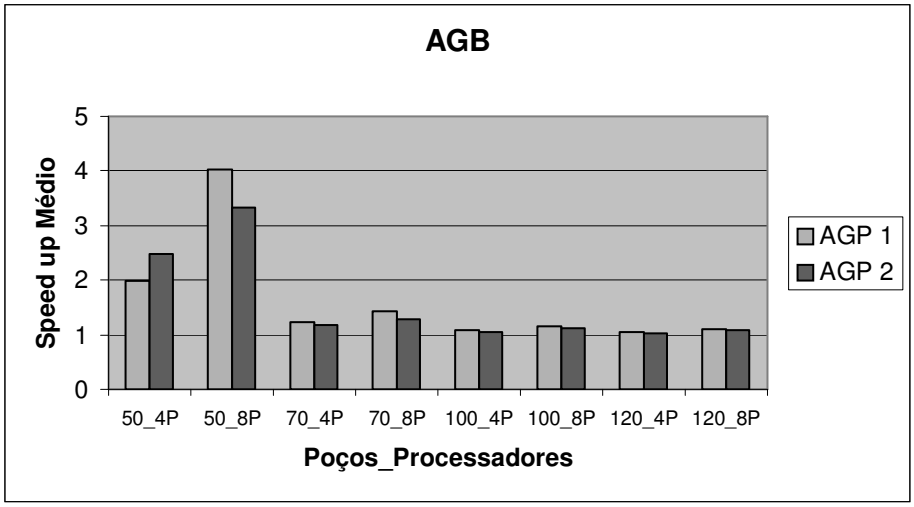

Figura 6.15: *Speed up* médio de AGB.

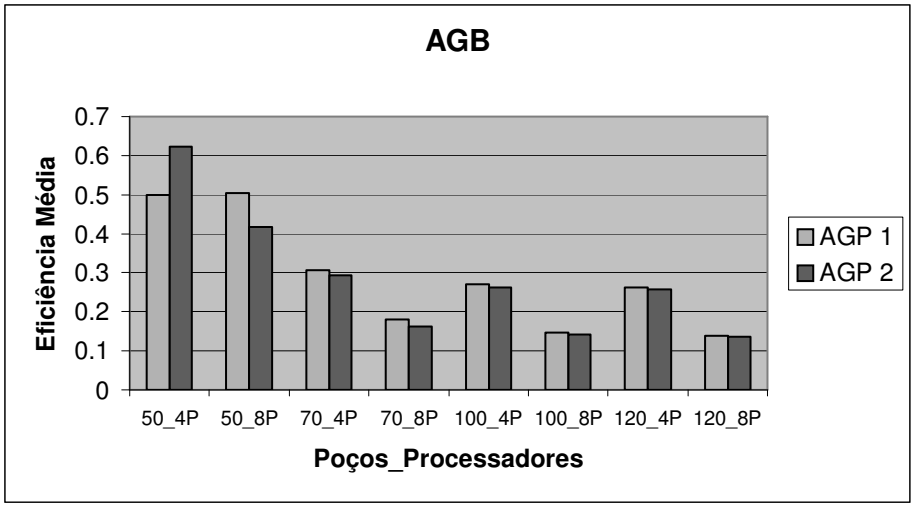

Figura 6.16: Eficiência Média de AGB.

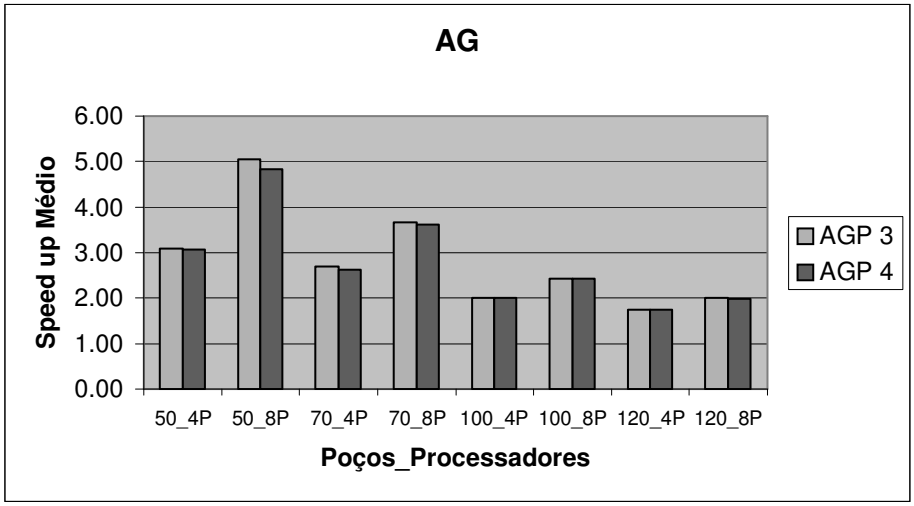

Figura 6.17*: Speed up* Médio do AG.

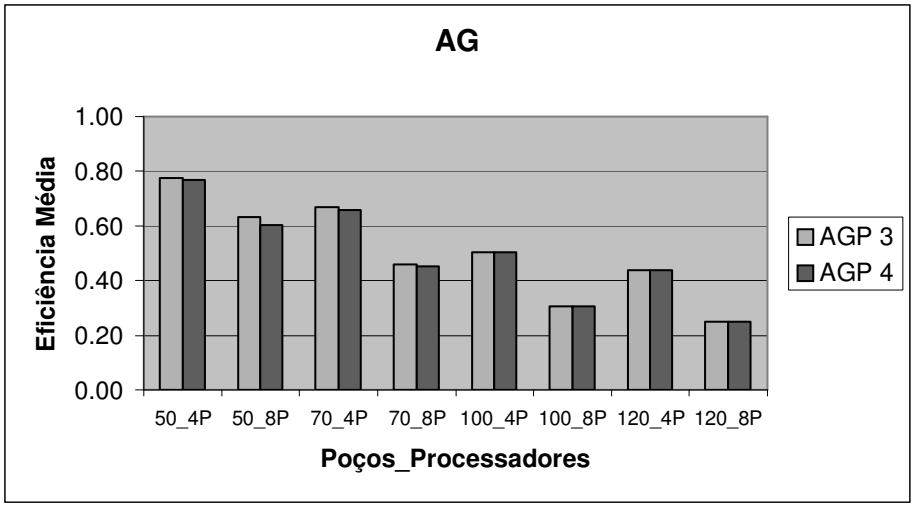

Figura 6.18: Eficiência Média de AG.

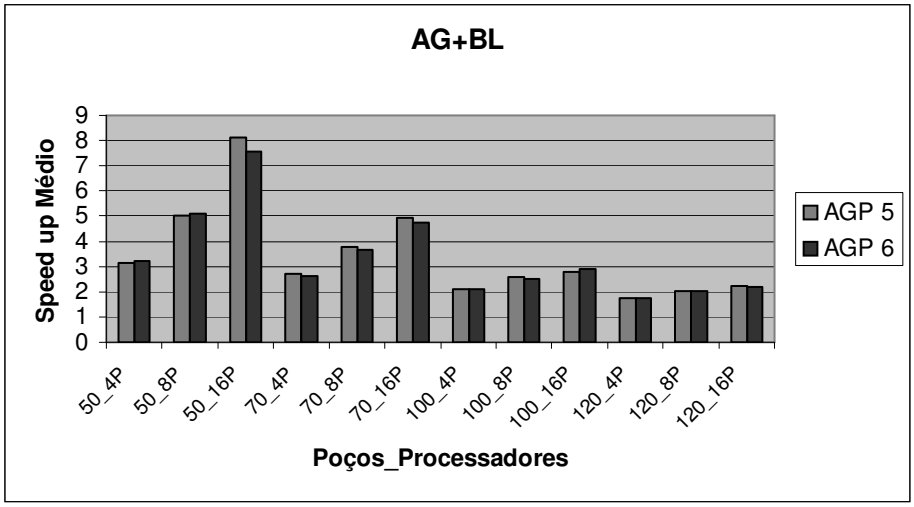

Figura 6.19: *Speed up* Médio de AG+BL.

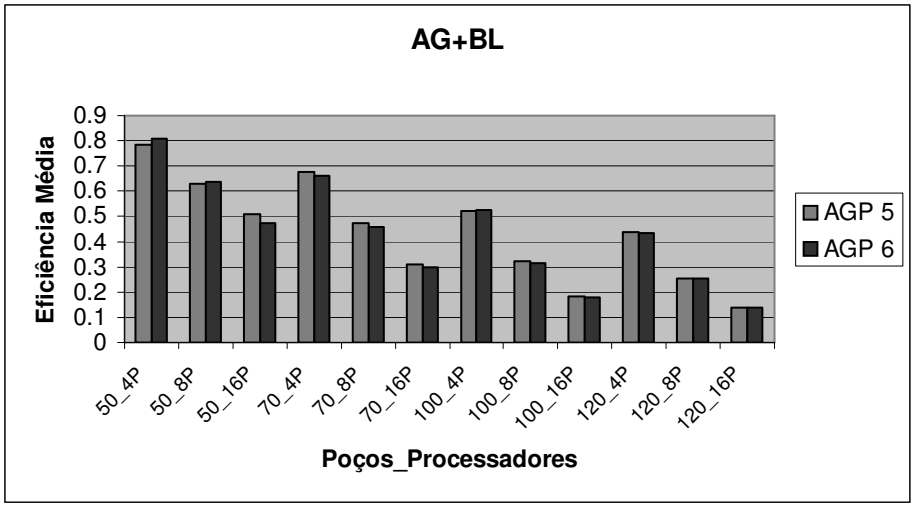

Figura 6.20: Eficiência Média de AG+BL.

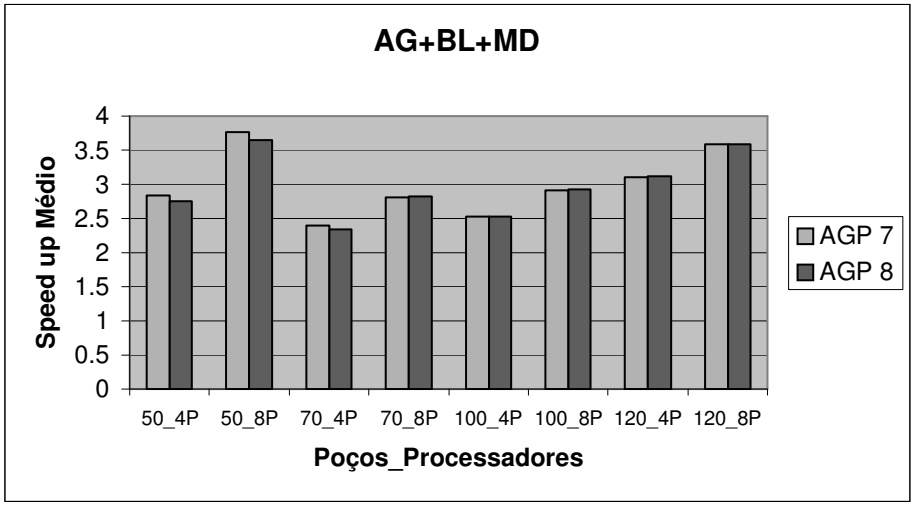

Figura 6.21: *Speed up* Médio de AG+BL+MD.

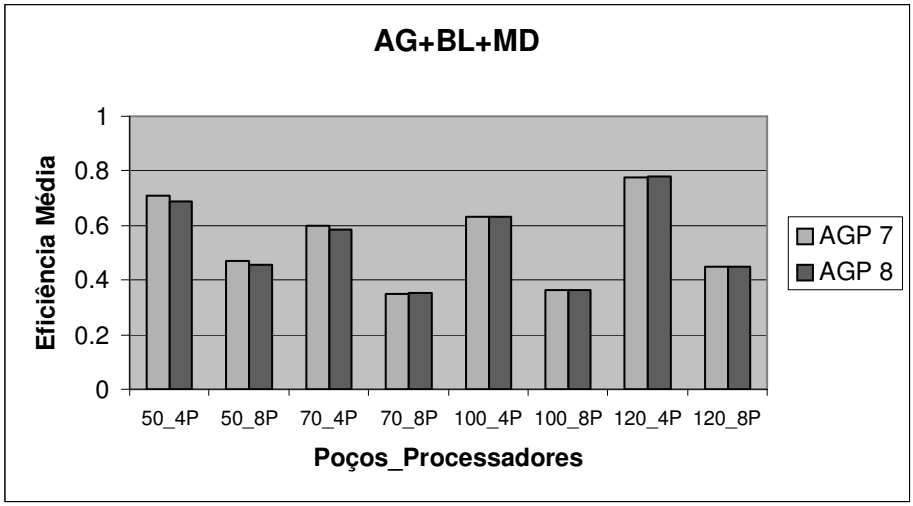

Figura 6.22: Eficiência Média de AG+BL+MD.

Ainda com base nos gráficos podemos notar que o TRUMPS não alcançou um *speed up* linear. Um *speed up* é considerado linear quando é igual ao número de processos empregados, portanto essa igualdade não aconteceu. O *speed up* não linear acontece, devido a uma parte do algoritmo que não foi paralelizada e também por causa do tempo gasto pela comunicação entre os processos. Esta fase é a responsável pela entrada de dados do algoritmo: inicialização do algoritmo, leitura do grafo, leitura das informações sobre os poços, conversão do grafo em grafo completo, inicialização do genético, e chamaremos de Fase 1.

A inicialização do algoritmo realiza atribuições às estruturas de armazenamento que serão utilizadas no decorrer do algoritmo que são: matriz de distâncias, LA (Lista de adjacência), LRC (Lista Restrita de Candidatos) e LC (Lista de Candidatos). Em seguida o algoritmo realiza a entrada de dados propriamente dita, ou seja, a leitura do grafo e as informações sobre os poços são realizadas. Numa etapa seguinte, o grafo lido é transformado em um grafo completo a fim de simplificar os cálculos a serem realizados pelos AGs. Numa outra etapa, o algoritmo genético é preparado para iniciar sua execução, ou seja, as estruturas "vetor de indivíduos" e "vetor de aptidão" são inicializadas.

Após esta fase, o processamento do algoritmo genético se inicia, buscando soluções promissoras, utilizando as heurísticas envolvidas em cada versão. Esta nova fase chamaremos de Fase 2 da aplicação.

A Fase 2, foi paralelizada. E esta sim, tem um *speed up* quase linear, devido ao paralelismo intrínseco dos AGs. Tanto o *speed up* quanto a eficiência se mantém a taxas constantes independente do número de poços envolvidos em cada instância, conforme comprovado através da figuras 6.23 à 6.30.

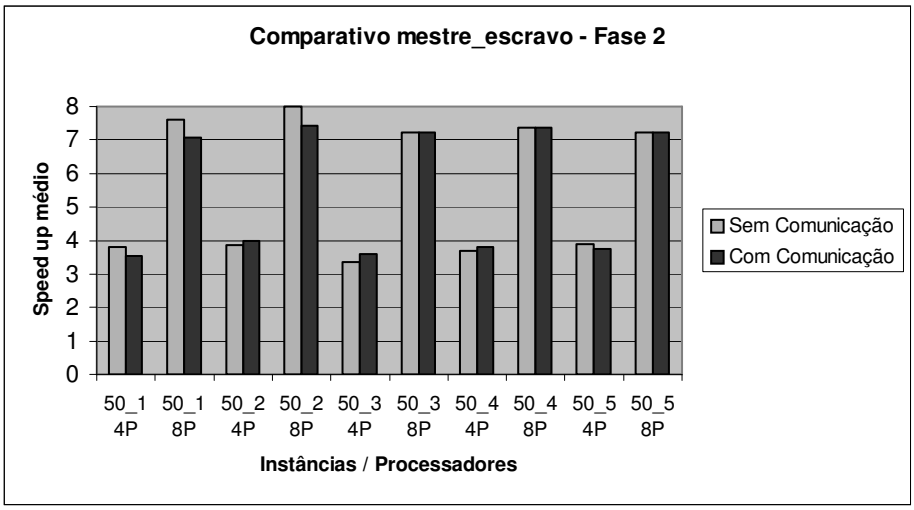

Figura 6.23: Comparativo de *speed up* médio (exceto AGP7 e AGP8).

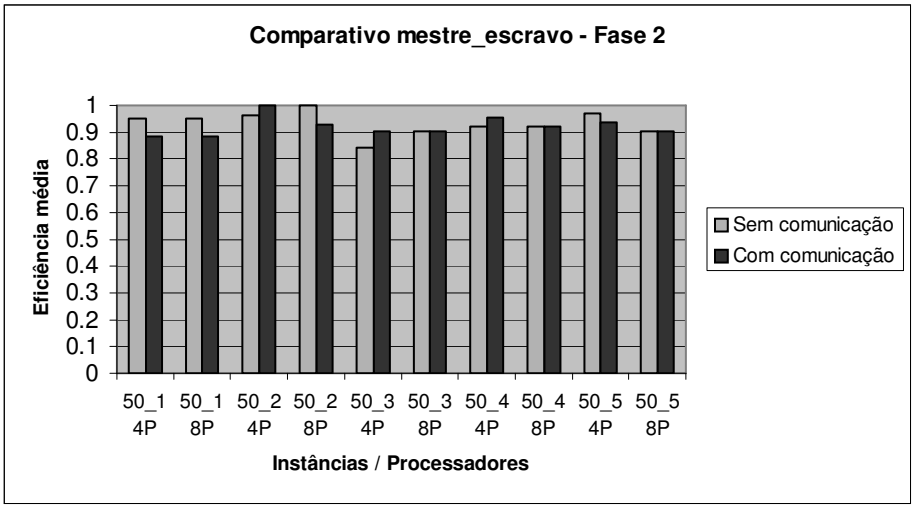

Figura 6.24: Comparativo de *Eficiência* média (exceto AGP7 e AGP8).

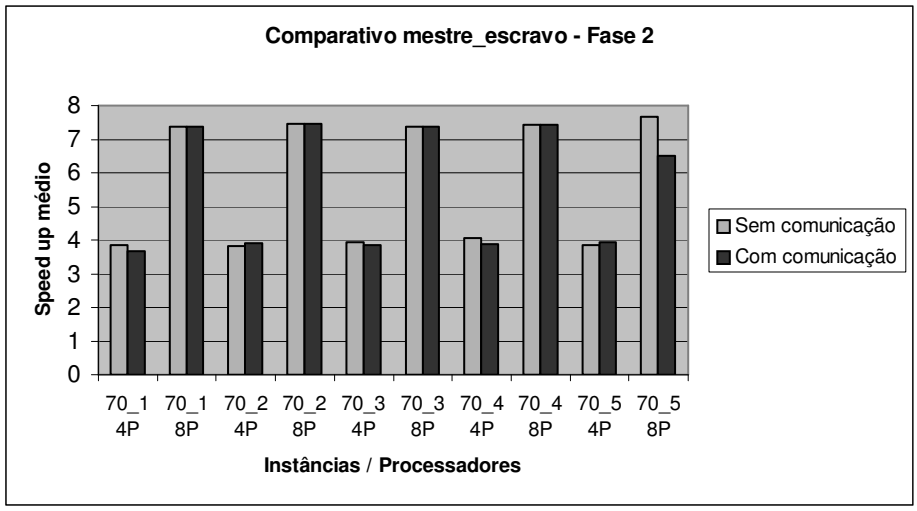

Figura 6.25: Comparativo de *speed up* (exceto AGP7 e AGP8).

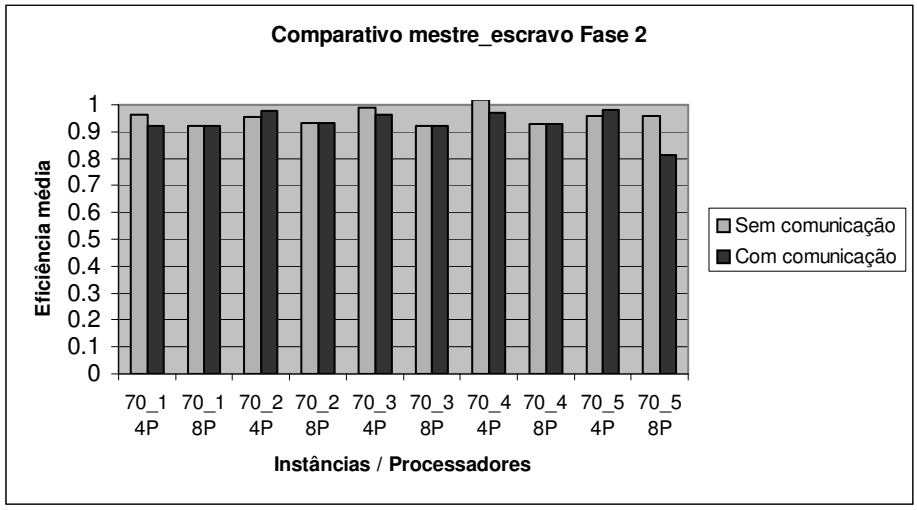

Figura 6.26: Comparativo de *Eficiência* média (exceto AGP7 e AGP8).

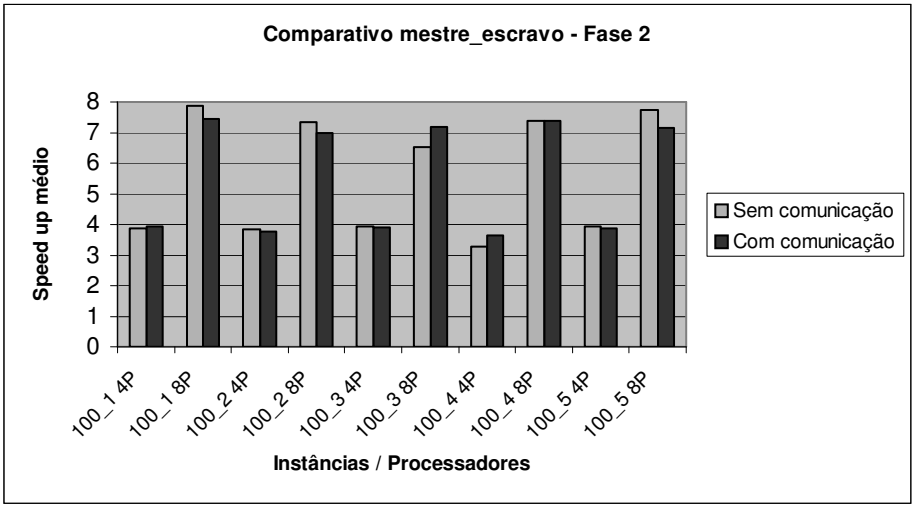

Figura 6.27: Comparativo de *speed up* médio (exceto AGP7 e AGP8).

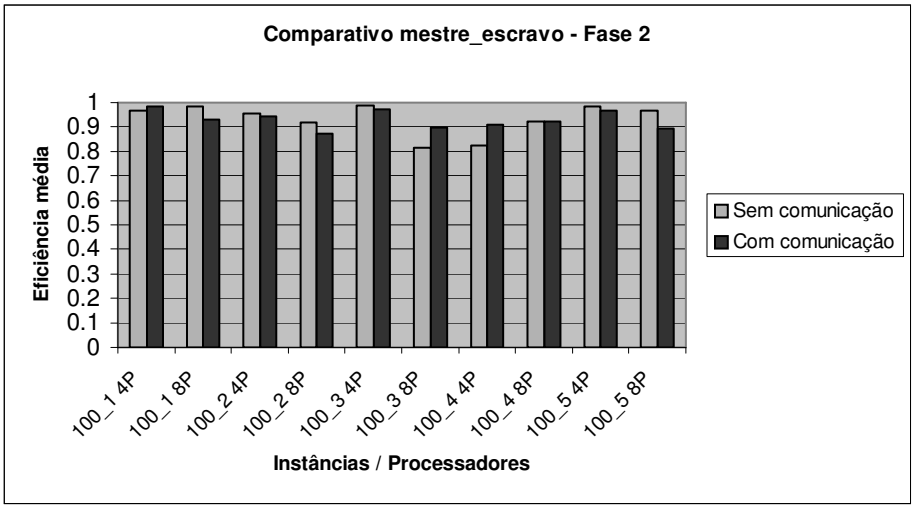

Figura 6.28: Comparativo de *Eficiência* (exceto AGP7 e AGP8).

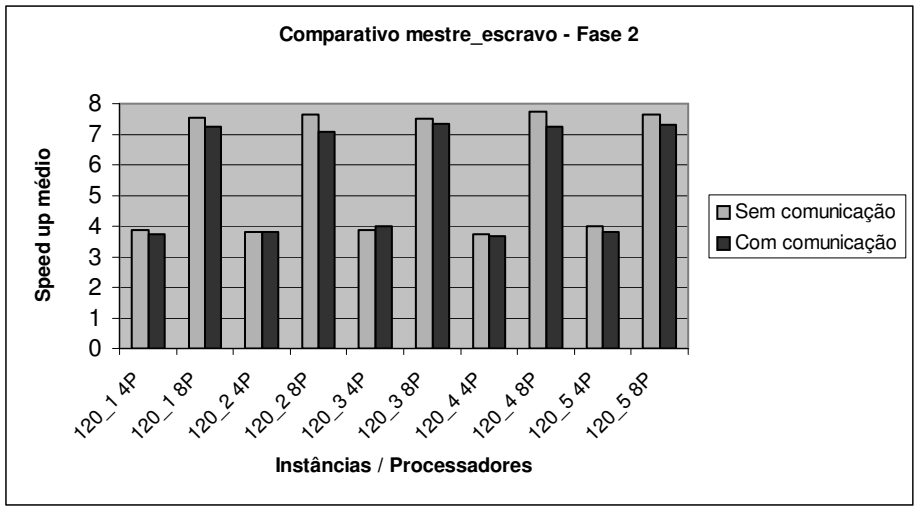

Figura 6.29: Comparativo de *speed up* (exceto AGP7 e AGP8).

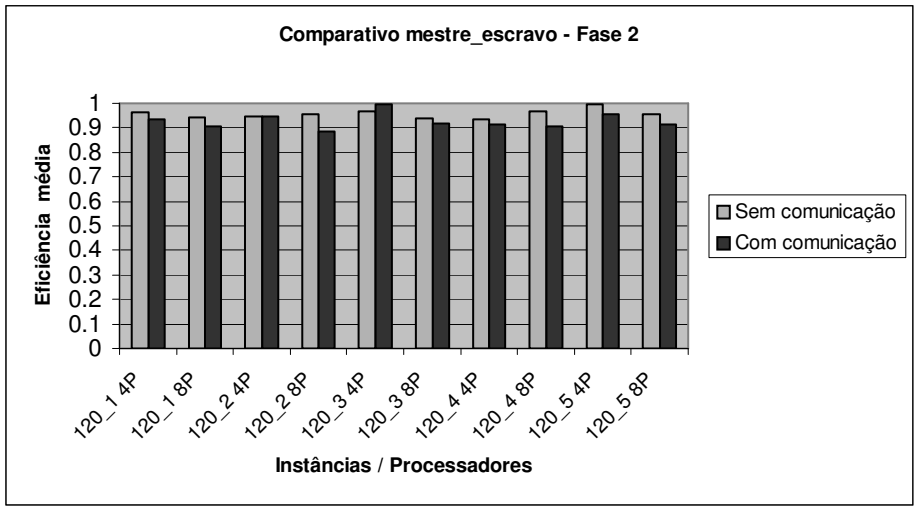

Figura 6.30: Comparativo de *Eficiência* (exceto AGP7 e AGP8).

Para ilustrar melhor esta situação, suponha que um algoritmo seqüencial levou 60 segundos de execução total. Suponhamos que a Fase 1 consumiu 20 segundos, enquanto que a Fase 2 consumiu 40 segundos. Se este algoritmo for executado em uma máquina com quatro processadores, o tempo total de execução nesta máquina seria: tempo = 20 + 40 / 4 = 30 segundos. Generalizando, poderíamos determinar o tempo computacional em paralelo de uma aplicação, como sendo, aproximadamente:

Tempo Paralelo Total = Tempo_Seqüencial (Fase 1) + (Tempo_Seqüencial (Fase 2) / P),

onde Tempo_Seqüencial (Fase 1) representa o tempo consumido pelo algoritmo durante a Fase 1, Tempo_Seqüencial (Fase 2) é o tempo consumido durante a Fase 2 e P é o número de processadores empregados.

Outro fator importante notado por este estudo, em relação ao tempo computacional total do algoritmo, foi que o *speed up* vai diminuindo à medida que o problema aumenta de tamanho. Isto se deve ao aumento de tempo da Fase 1 em relação ao tempo total de execução do algoritmo para instâncias grandes.

Os gráficos 6.31 à 6.36 comprovam esta tendência. À medida que, mais poços estão presentes em um campo petrolífero, maior será o tempo da Fase 1 (não paralelizada). O tempo de Fase 1 cresce exponencialmente, independentemente da variação do número de gerações e população.

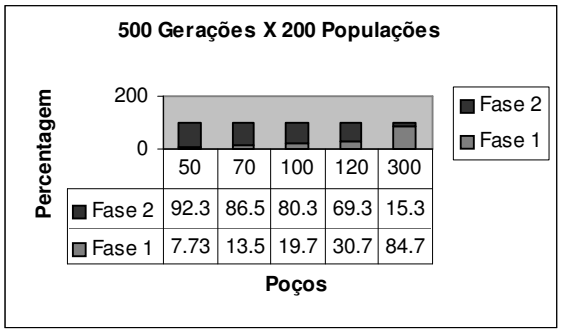

Figura 6.31: Percentual de tempo das Fases 1 e 2 em diferentes instâncias do RUMP.

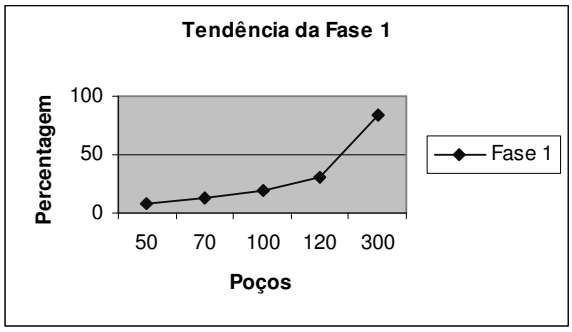

Figura 6.32: Tempo na Fase 1 para diferentes instâncias.

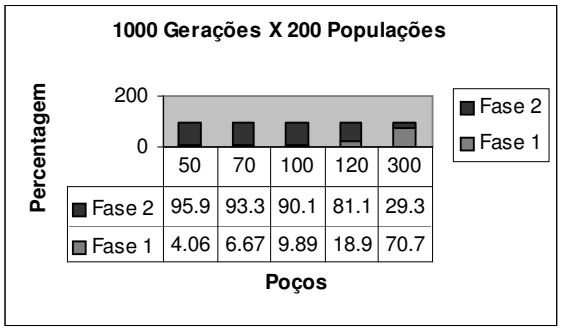

Figura 6.33: Percentual de tempo das Fases 1 e 2 em diferentes instâncias do RUMP.

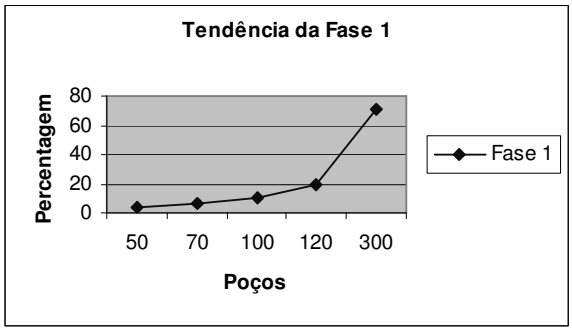

Figura 6.34: Tempo na Fase 1 para diferentes instâncias.

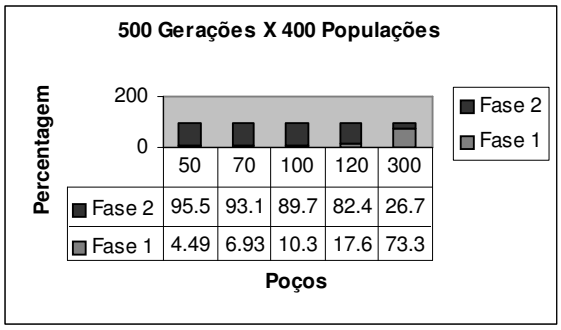

Figura 6.35: Percentual de tempo das Fases 1 e 2 em diferentes instâncias do RUMP.

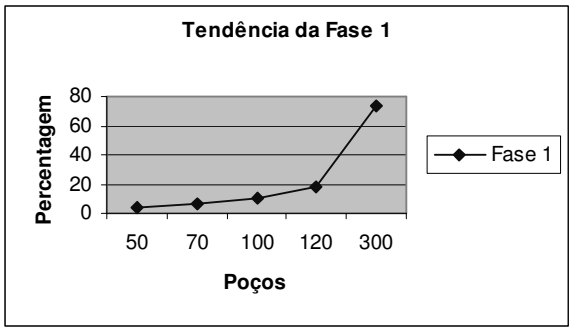

Figura 6.36: Tempo na Fase 1 para diferentes instâncias.

De maneira análoga, podemos realizar a mesma análise para a versão com módulo de busca local e mineração de dados, conforme podemos observar na tabela 6.3. Cada célula da tabela representa o resultado obtido, em temos percentuais, de análises de diferentes instâncias, onde se procurou variar o número de gerações e de indivíduos da população.

AG+BL+MD	500 Gerações 200 Populações		1000 Gerações 200 Populações		500 Gerações 400 Populações	
	Fase 1	**Fase 2**	**Fase 1**	**Fase 2**	**Fase 1**	**Fase 2**
50 Poços	0	100	0	100	4	96
70 Poços	4	96	4	96	2	98
100 Poços	7.5	92.5	5	95	3	97
120 Poços	13	87	7	93	5	95
300 Poços	49	51	9	91	6	94

Tabela 6.3: Resultados percentuais das Fases 1 e 2 em relação ao tempo computacional total.

Nesta análise, podemos notar que a Fase 2 das versões AGP7 e AGP8 é mais extensa do que a das outras versões. A diferença percentual da Fase 1, em relação a Fase 2, diminui de acordo com o número de gerações, com o tamanho da população e, principalmente, com o número de vezes em que o módulo de mineração de dados foi executado**.** Desta forma, o *speed up* alcançado por esta versão foi superior às demais. Para este caso, o *speed up* vai aumentando à medida que as instâncias aumentam de tamanho, conforme figuras 6.21 e 6.22. O motivo pelo qual isto acontece é que o tempo consumido pela Fase 2 consegue compensar o aumento exponencial da Fase 1, ou seja, AGP 7 e AGP8 se beneficiam mais com a paralelização do algoritmo. Baseado neste fator, a diferença entre a versão seqüencial que era muito grande para pequenas instâncias, vai diminuindo em relação à versão paralela, à medida que, as instâncias aumentam de tamanho. Este fato torna estes algoritmos mais competitivos com relação às outras versões quando utilizados para instâncias grandes e, além disso, tornaram se as duas melhores versões em relação aos *speed ups* e *eficiências*. Fazendo uma análise comparativa entre as outras versões, observarmos que para instâncias maiores, o tempo percentual da Fase 1 na versão do algoritmo com módulo de busca local e mineração de dados é menor do que se comparado as outras versões (figuras 6.31 à 6.36 e Tabela 6.3). Isto se deve ao aumento significativo no tempo da Fase 2 deste algoritmo, fato que contribuiu para um melhor desempenho e eficiência, em decorrência da Fase 2 ter sido paralelizada e a Fase 1 não paralelizada.

A opção de paralelizar o AGP7 e o AGP8 de forma descentralizada partiu de observações a respeito de uma implementação anterior destes algoritmos, onde foram adotados os modelos mestres-escravos.

O fato da Fase 2 dos algoritmos AGP7 e AGP8 (mestre-escravo) não ser linear, nos propiciou situações absurdas em que algumas instâncias do algoritmo paralelo consumiam mais tempo do que as correspondentes executadas seqüencialmente, conforme podemos verificar nos gráficos 6.37 à 6.40, fato que nos forçou a abandonar este modelo.

O fatores que justificam consumo exagerado na Fase 2 do algoritmo AGP7 e AGP8 mestre-escravo são:

1) Sabemos que neste modelo cada processador trabalha com uma sub-população diferente. Como a natureza do AG é produzir soluções aleatórias a partir de uma solução inicial, o número de *itemsets* gerados em cada execução do Apriori pode ser diferente para cada processador.
2) Além disso, o número de vezes que o Apriori é executado na aplicação (em cada processador) pode variar bastante.
3) Alguns processadores que venham a terminar a mineração de dados, devem ficar aguardando o término da computação dos demais processadores, a fim de receber a melhor solução encontrada até o momento, ou seja, alguns processos mais rápidos esperam algumas vezes, durante longos períodos, pelos processos mais lentos.

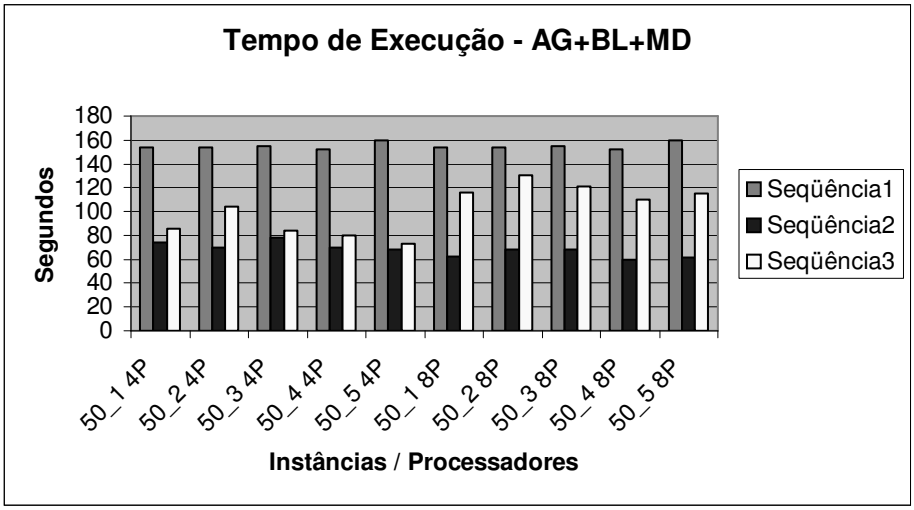

Figura 6.37: Tempo computacional entre versões AG+BL+MD (mestre-escravo).

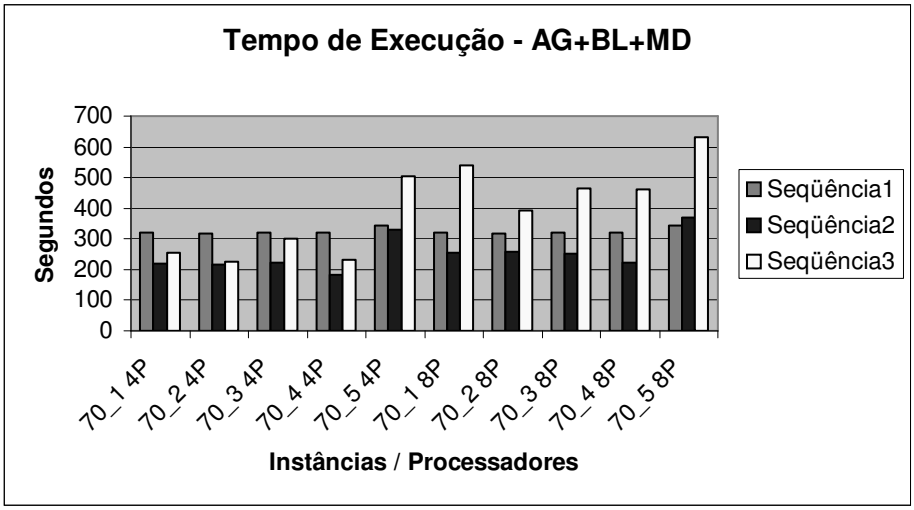

Figura 6.38: Tempo computacional entre versões AG+BL+MD (mestre-escravo).

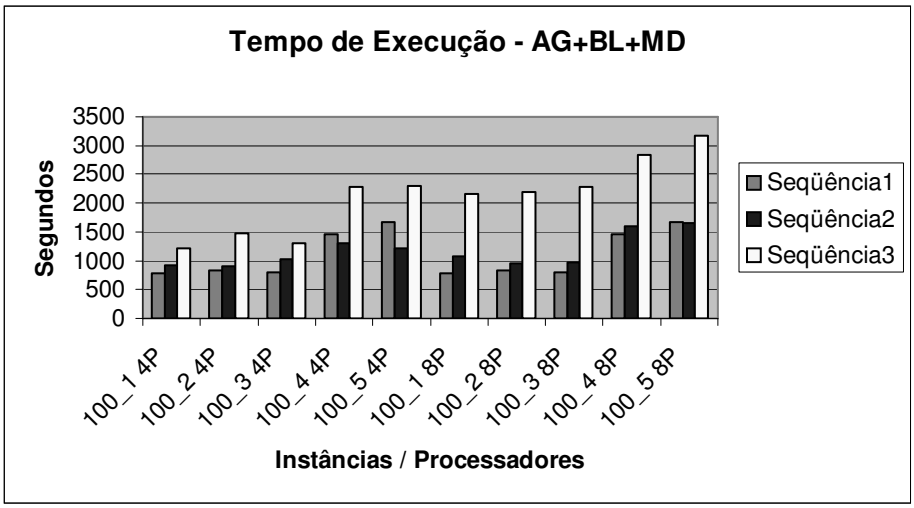

Figura 6.39: Tempo computacional entre versões AG+BL+MD (mestre-escravo).

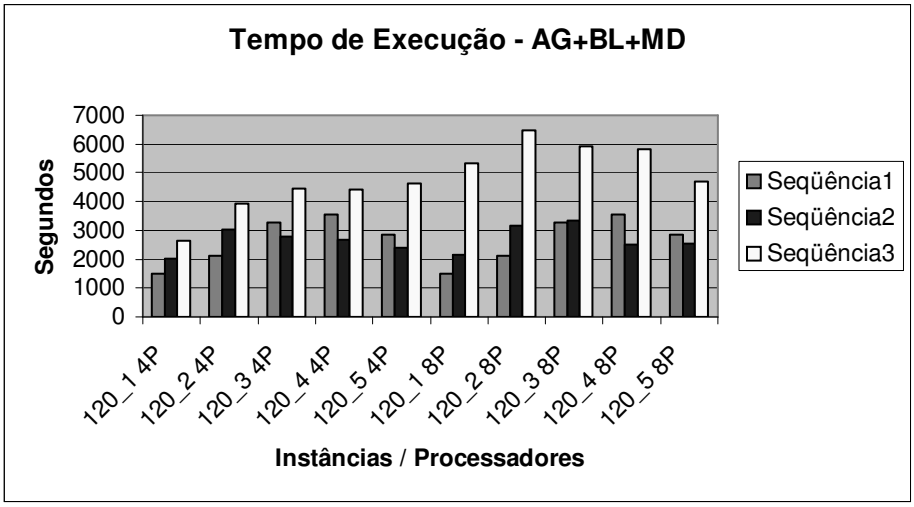

Figura 6.40: Tempo computacional entre versões AG+BL+MD (mestre-escravo).

6.3 Comparação entre Resultados dos Algoritmos Seqüenciais e Paralelos

Para melhor visualização dos tempos computacionais entre as versões dos algoritmos evolutivos, descrevemos, nas figuras seguintes, gráficos comparativos dos tempos computacionais entre versões seqüenciais e paralelas dos algoritmos aqui propostos.

Podemos observar através dos gráficos 6.41 à 6.56 que a diferença entre o tempo seqüencial e paralelo vai diminuindo à medida que a instância do problema cresce, logo o tempo do paralelismo cresce em relação ao aumento da instância do problema. Procurando generalizar a situação, podemos constatar que não é vantajoso executar o algoritmo paralelo para estas versões de AGs (exceto AGP7 e AGP8 descentralizados) com instâncias de tamanho muito grande, pois o tempo do algoritmo paralelo tende a se aproximar do tempo seqüencial.

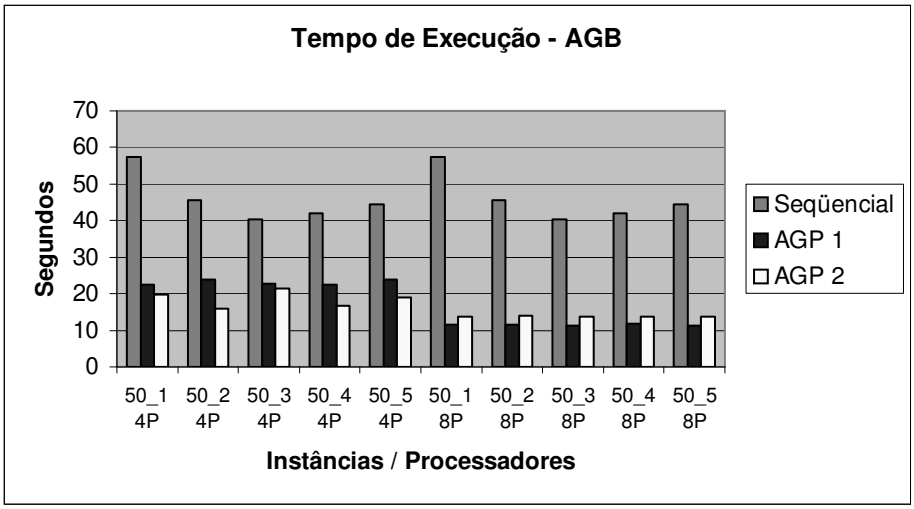

Figura 6.41: Comparativo de tempo entre seqüencial, AGP1 e AGP2 para 50 vértices.

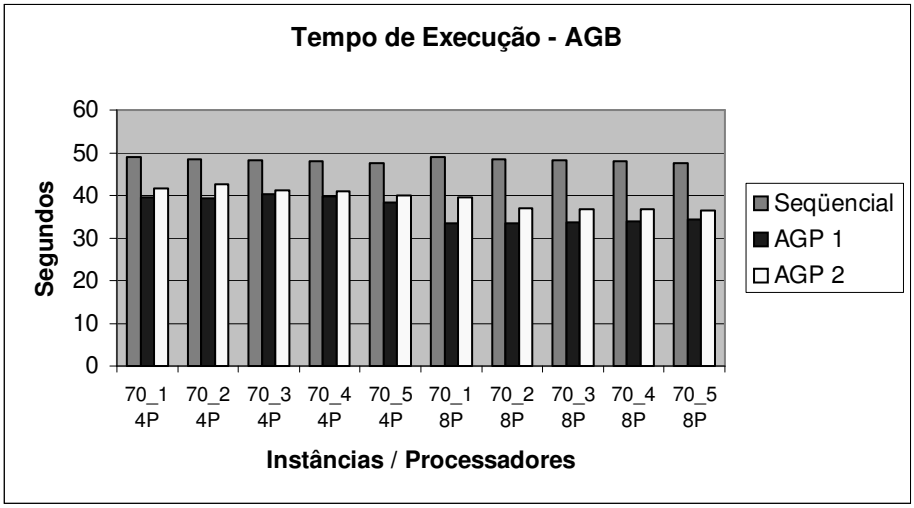

Figura 6.42: Comparativo de tempo entre seqüencial, AGP1 e AGP2 para 70 vértices.

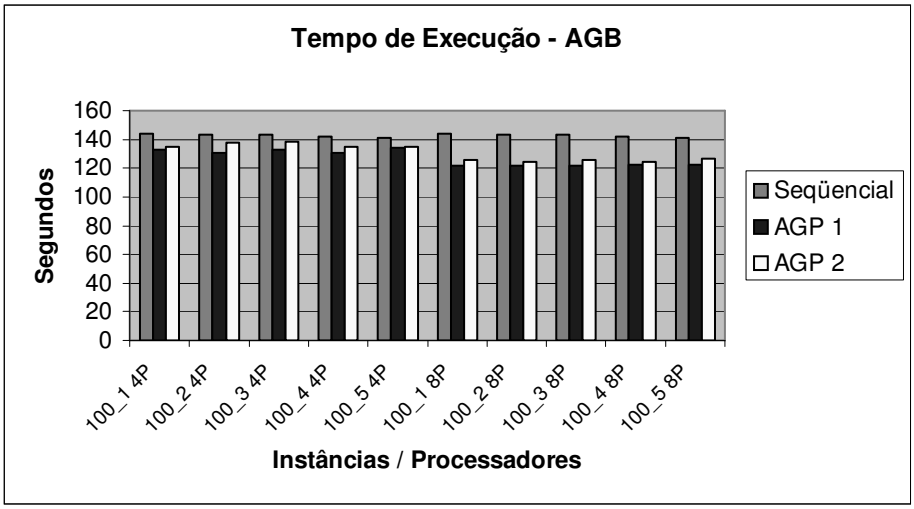

Figura 6.43: Comparativo de tempo entre seqüencial, AGP1 e AGP2 para 100 vértices.

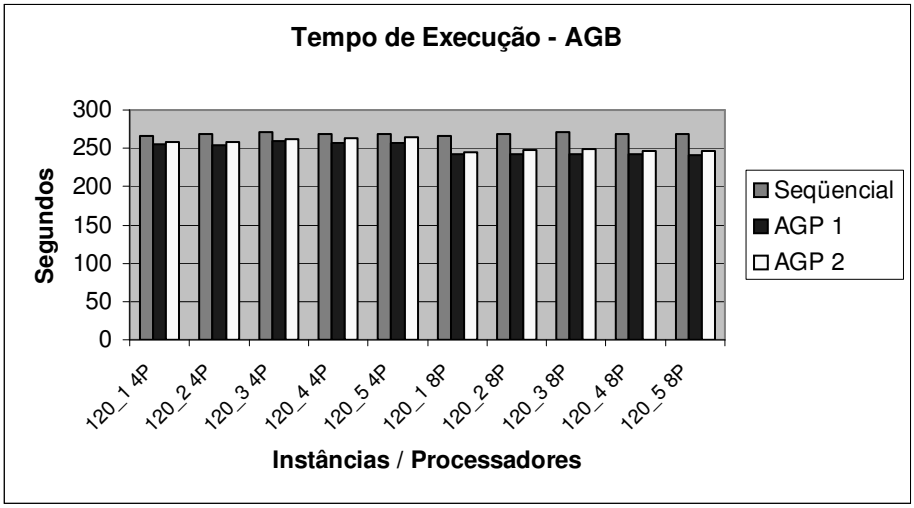

Figura 6.44: Comparativo de tempo entre seqüencial, AGP1 e AGP2 para 120 vértices.

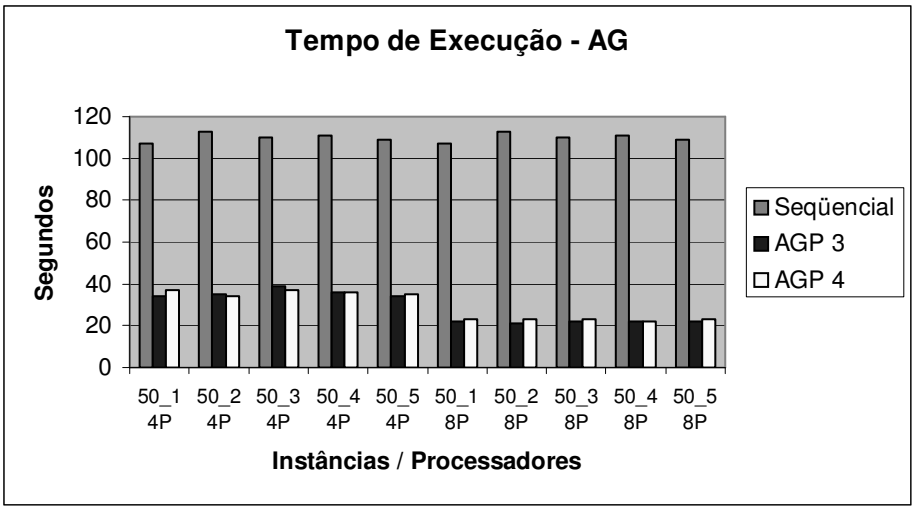

Figura 6.45: Comparativo de tempo entre seqüencial, AGP3 e AGP4 para 50 vértices.

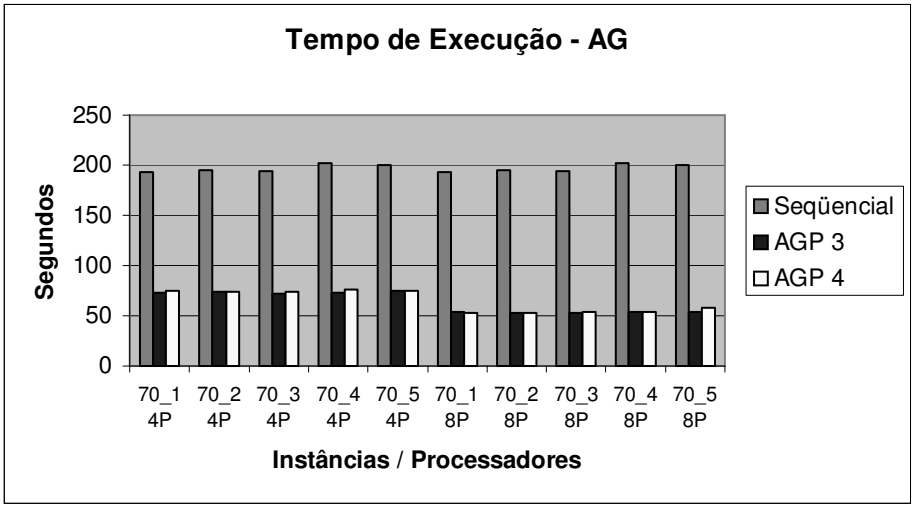

Figura 6.46: Comparativo de tempo entre seqüencial, AGP3 e AGP4 para 70 vértices.

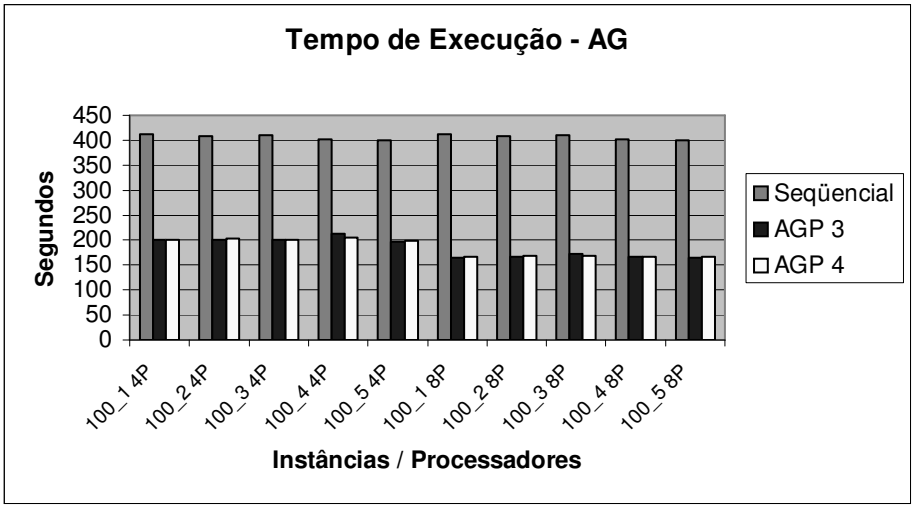

Figura 6.47: Comparativo de tempo entre seqüencial, AGP3 e AGP4 para 100 vértices.

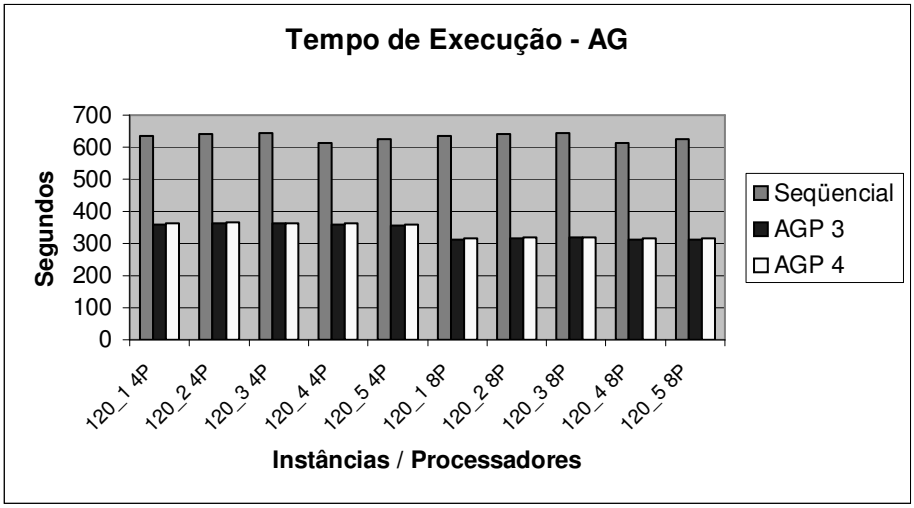

Figura 6.48: Comparativo de tempo entre seqüencial, AGP3 e AGP4 para 120 vértices.

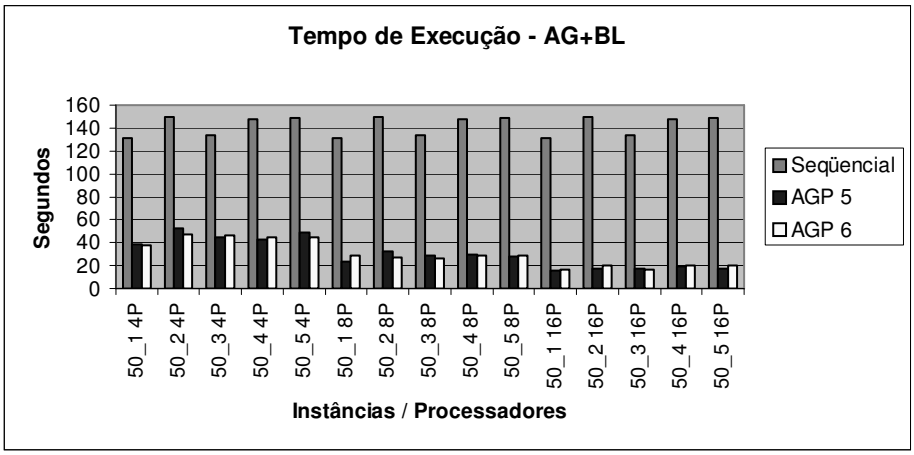

Figura 6.49: Comparativo de tempo entre seqüencial, AGP5 e AGP6 para 50 vértices.

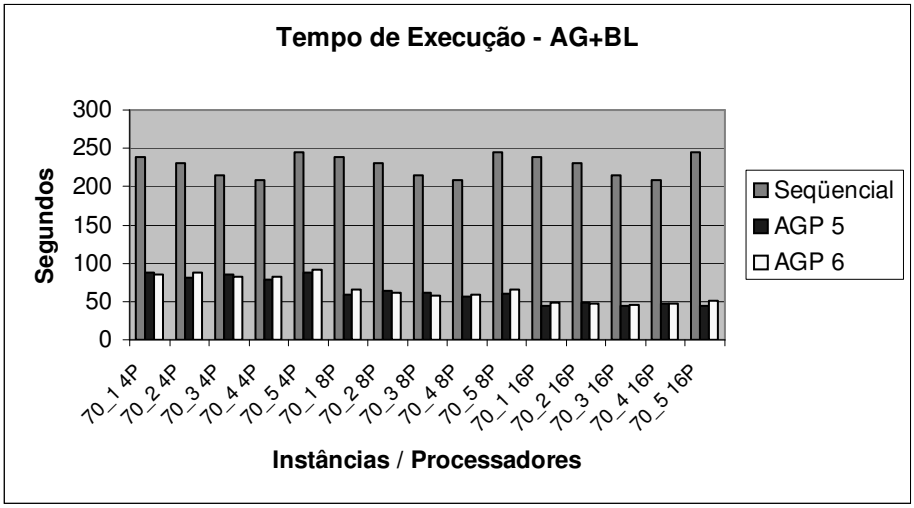

Figura 6.50: Comparativo de tempo entre seqüencial, AGP5 e AGP6 para 70 vértices.

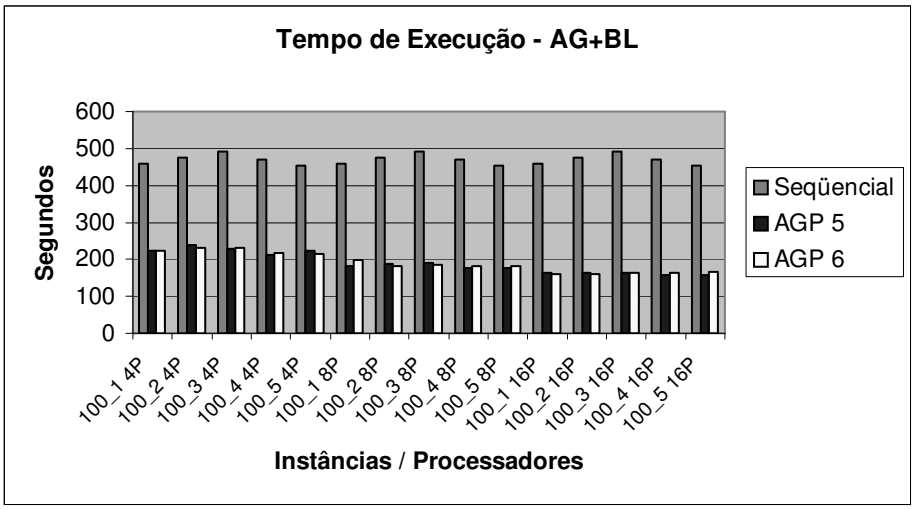

Figura 6.51: Comparativo de tempo entre seqüencial, AGP5 e AGP6 para 100 vértices.

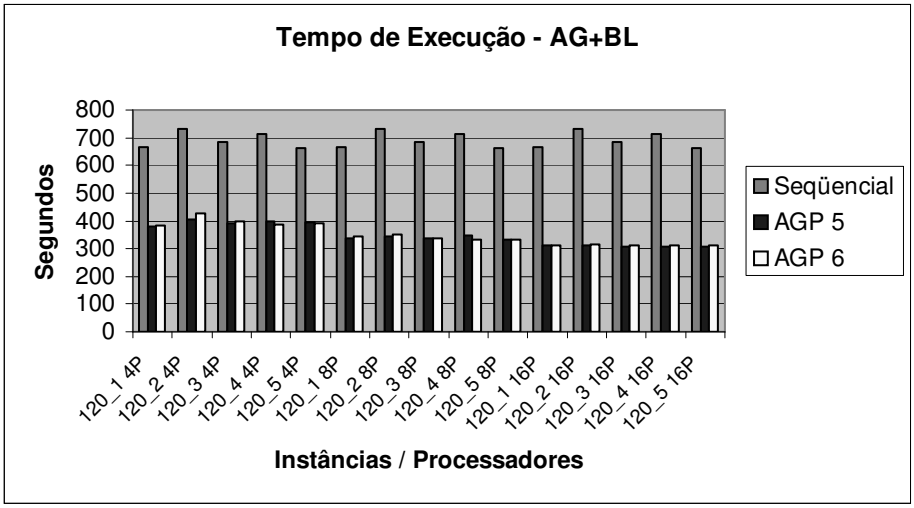

Figura 6.52: Comparativo de tempo entre seqüencial, AGP5 e AGP6 para 120 vértices.

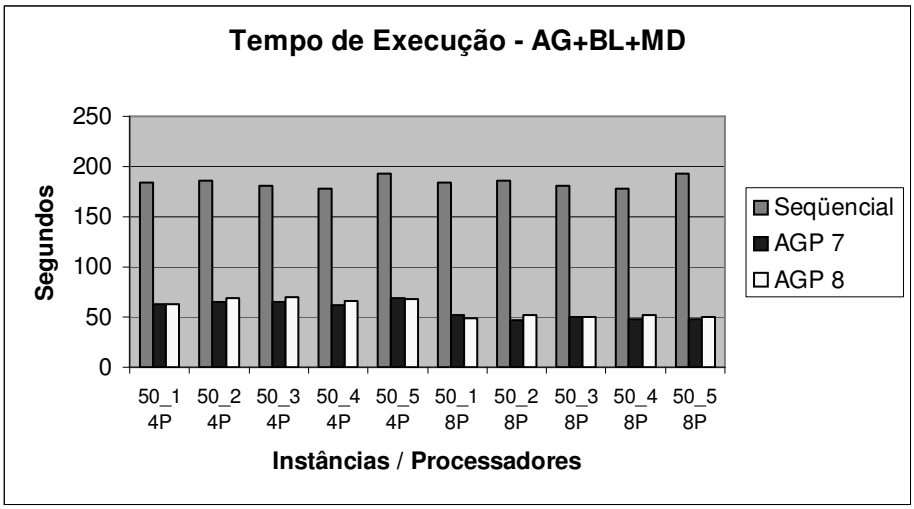

Figura 6.53: Comparativo de tempo entre seqüencial, AGP7 e AGP8 para 50 vértices.

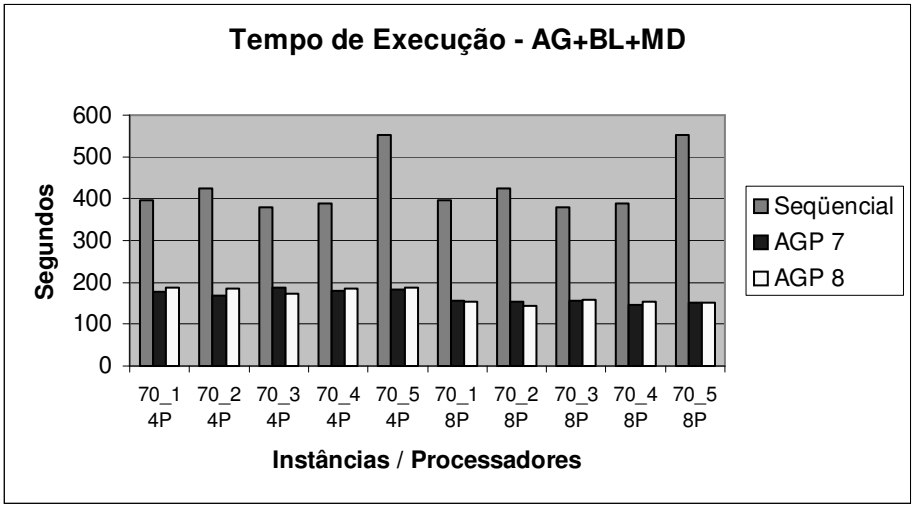

Figura 6.54: Comparativo de tempo entre seqüencial, AGP7 e AGP8 para 70 vértices.

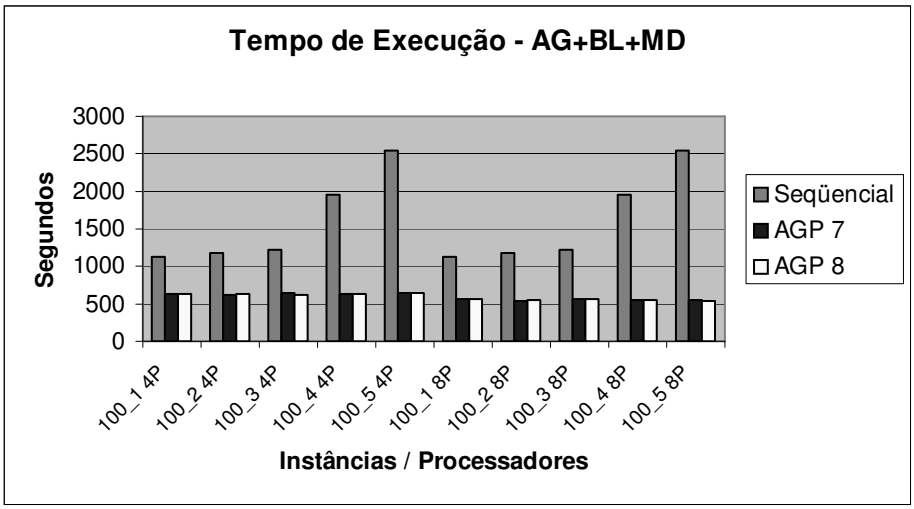

Figura 6.55: Comparativo de tempo entre seqüencial, AGP7 e AGP8 para 100 vértices.

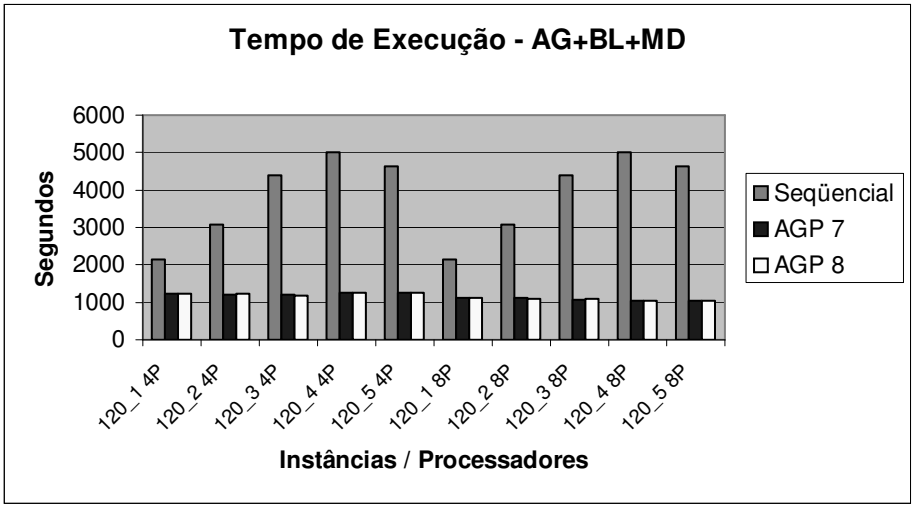

Figura 6.56: Comparativo de tempo entre seqüencial, AGP7 e AGP8 para 120 vértices.

É possível perceber, nas figuras 6.57 à 6.72, que se compararmos as versões paralelas com comunicação e sem comunicação de cada versão (no quesito qualidade de solução), a versão AGP2 obtém a maior diferença entre as soluções encontradas (não superior a 8%). Nas demais versões, a variação não conseguiu melhorar e nem prejudicar as soluções encontradas, mantendo a variação da solução praticamente estável, por volta de 0,17%. Em uma segunda análise que envolve comparação entre a qualidade da solução seqüencial e as soluções paralelas encontradas, a variação foi muito pequena em relação ao melhor das versões paralelas com a versão seqüencial. Praticamente, o algoritmo permaneceu estável quanto à qualidade da solução encontrada. A maior variação encontrada foi também na versão AGB, onde este valor não passou de 4% de melhoria.

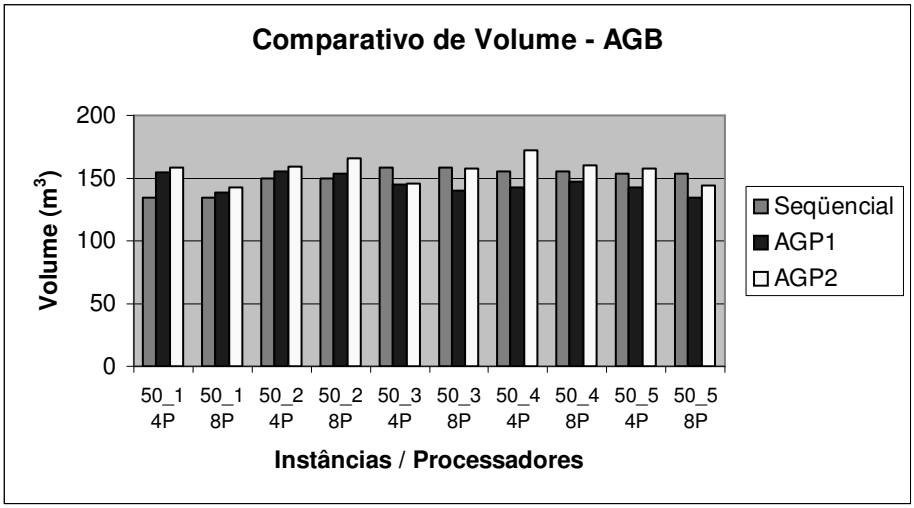

Figura 6.57: Comparativo de Volume entre AGB, AGP1 e AGP2.

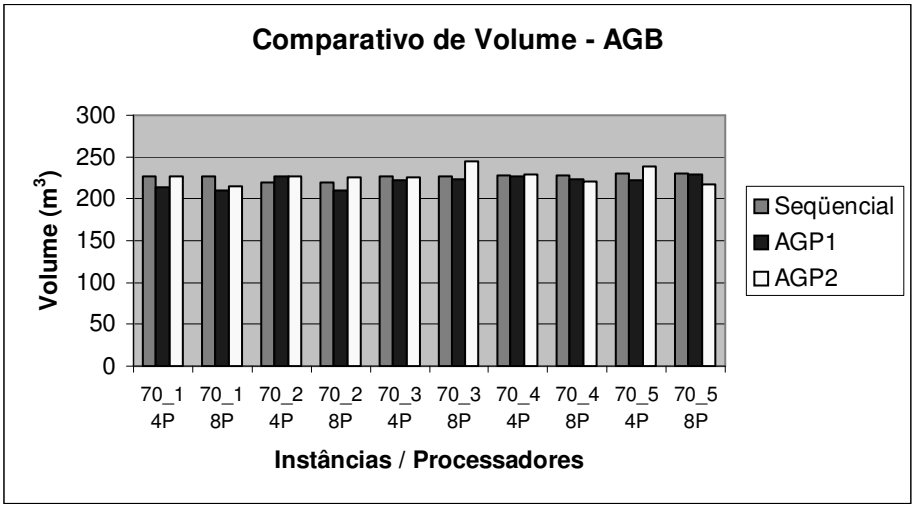

Figura 6.58: Comparativo de Volume entre AGB, AGP1 e AGP2.

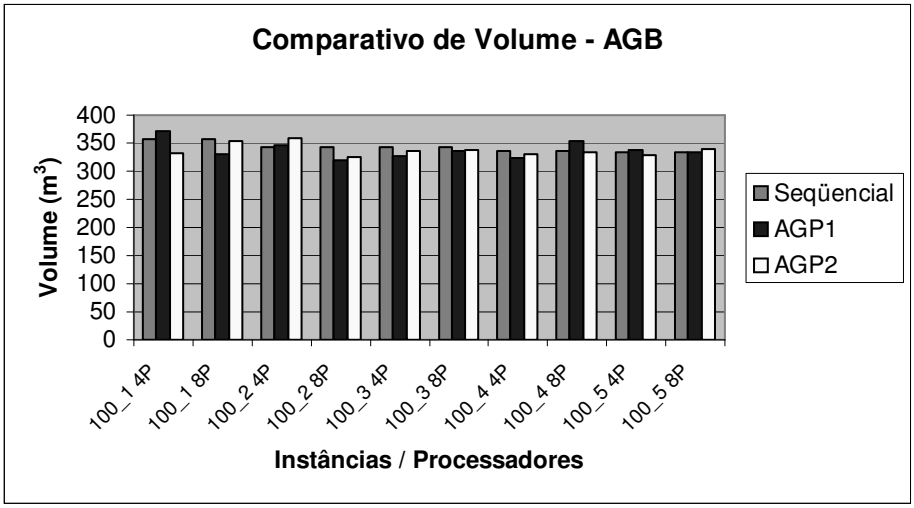

Figura 6.59: Comparativo de Volume entre AGB, AGP1 e AGP2.

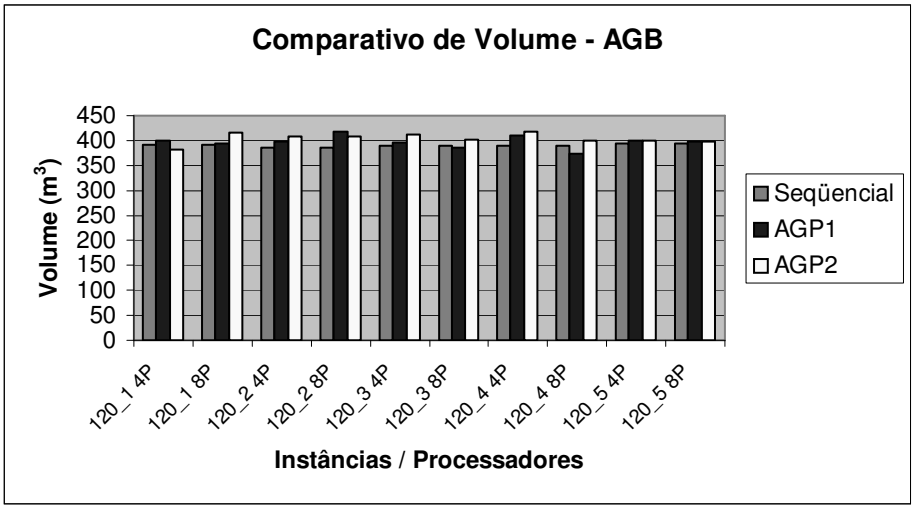

Figura 6.60: Comparativo de Volume entre AGB, AGP1 e AGP2.

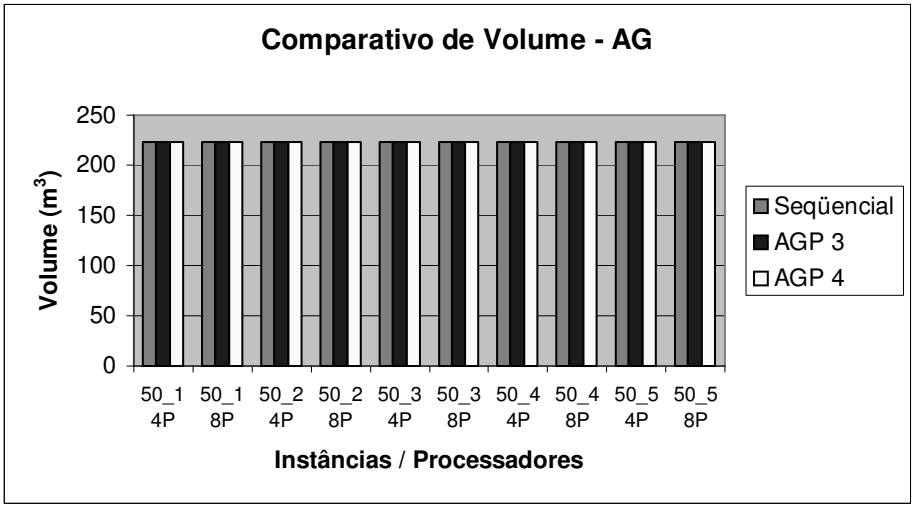

Figura 6.61: Comparativo de Volume entre AG, AGP3 e AGP4.

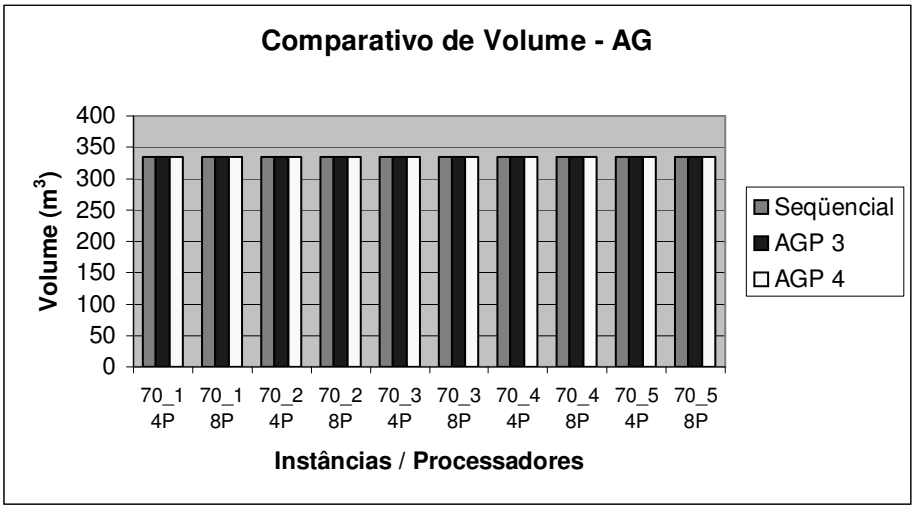

Figura 6.62: Comparativo de Volume entre AG, AGP3 e AGP4.

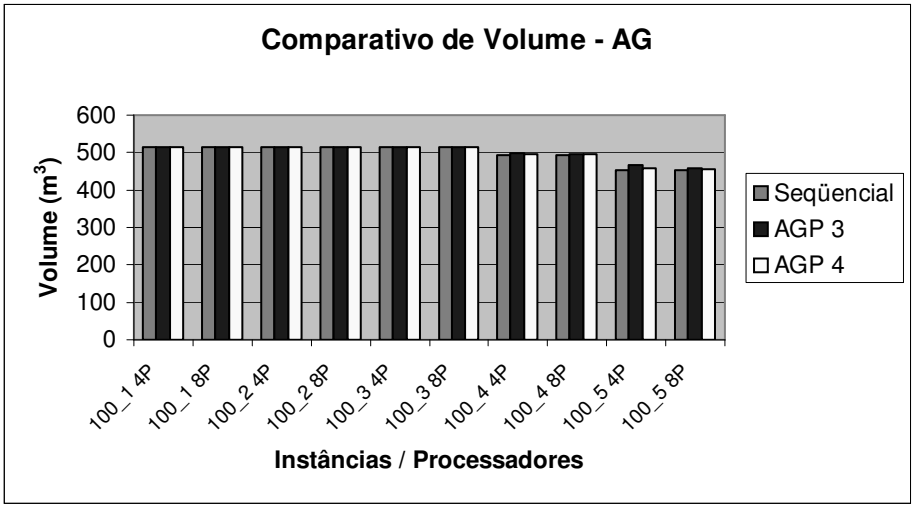

Figura 6.63: Comparativo de Volume entre AG, AGP3 e AGP4.

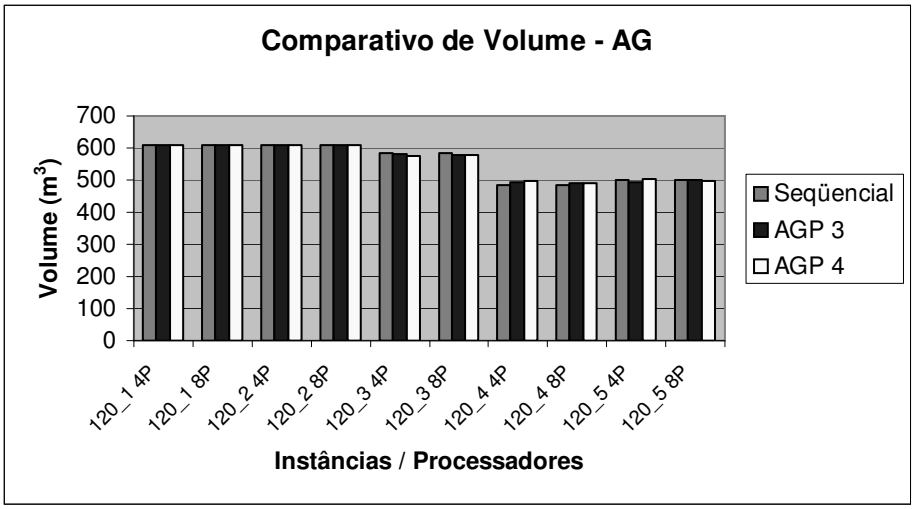

Figura 6.64: Comparativo de Volume entre AG, AGP3 e AGP4.

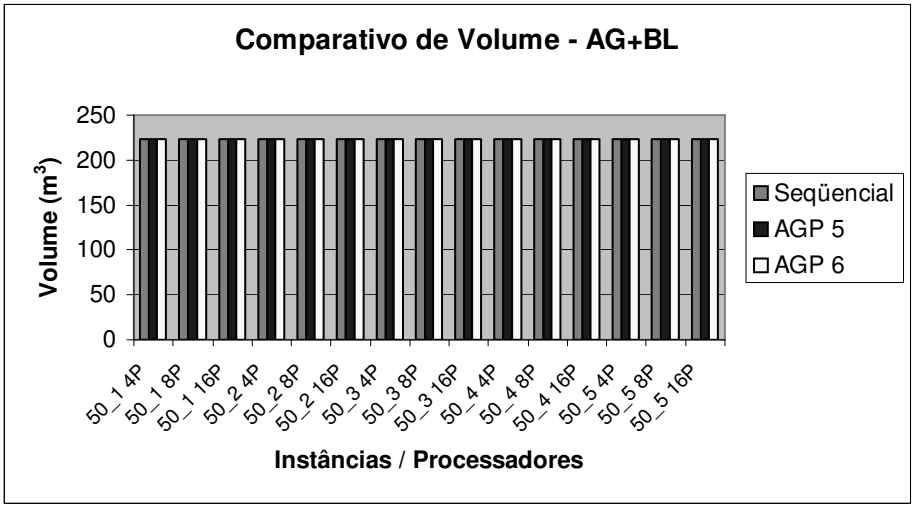

Figura 6.65: Comparativo de Volume entre AG+BL, AGP5 e AGP6.

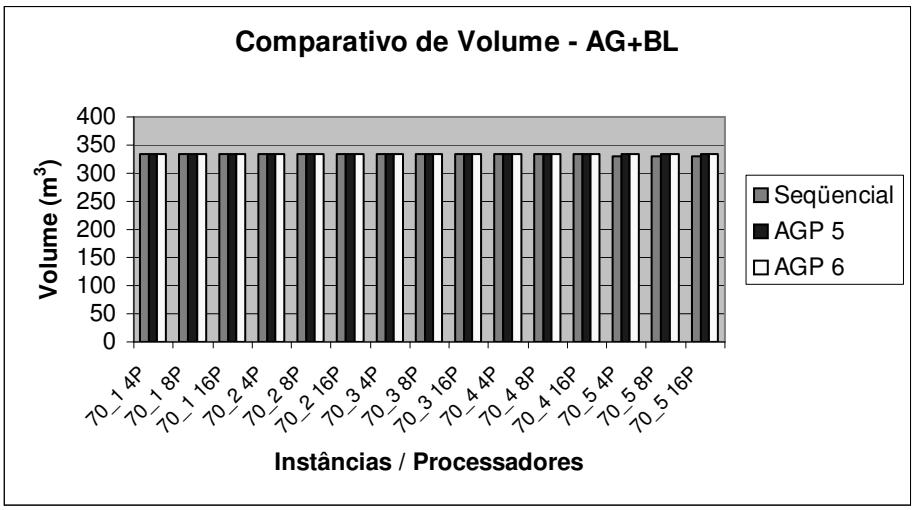

Figura 6.66: Comparativo de Volume entre AG+BL, AGP5 e AGP6.

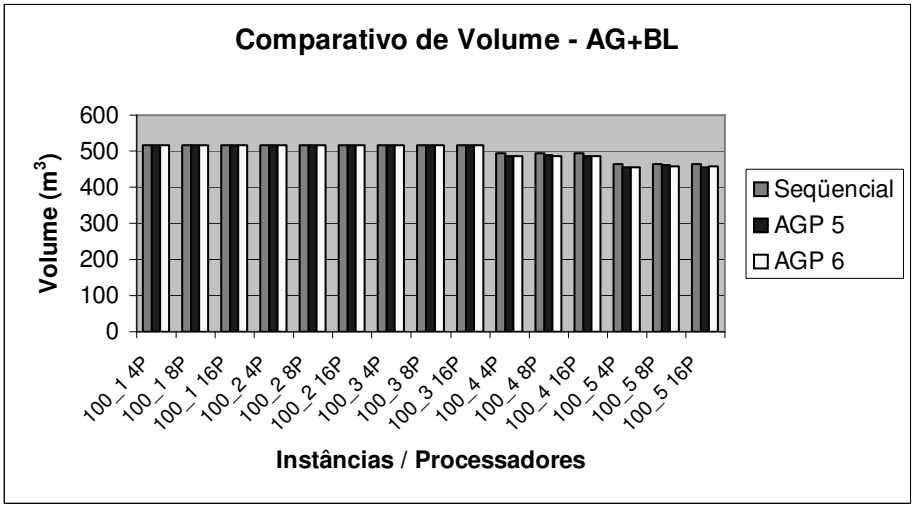

Figura 6.67: Comparativo de Volume entre AG+BL, AGP5 e AGP6.

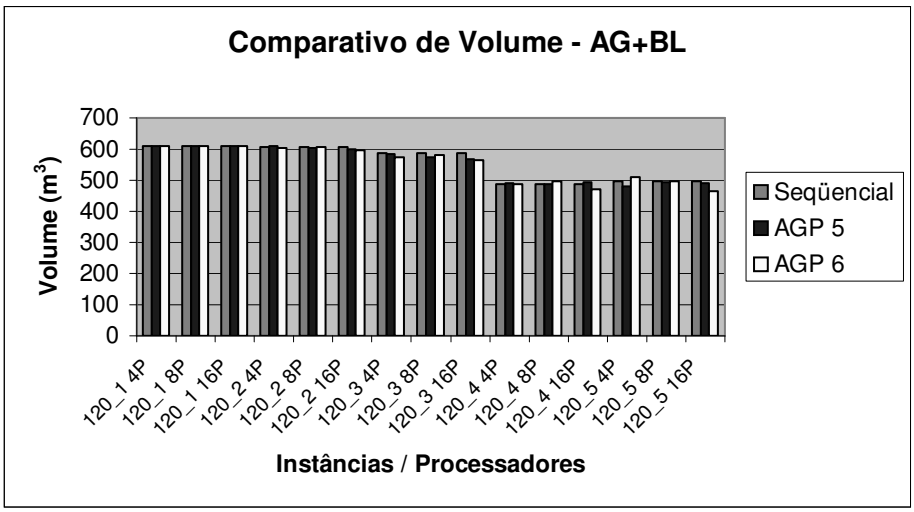

Figura 6.68: Comparativo de Volume entre AG+BL, AGP5 e AGP6.

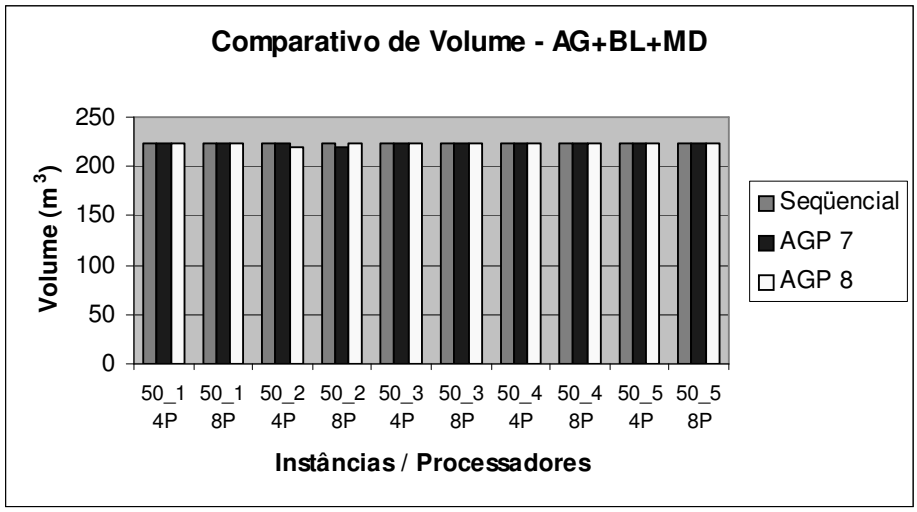

Figura 6.69: Comparativo de Volume entre AG+BL+MD, AGP7 e AGP8.

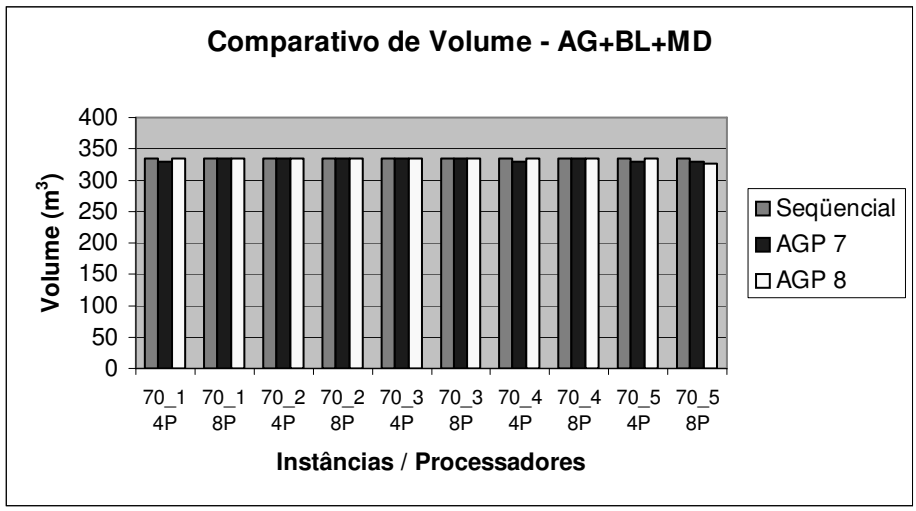

Figura 6.70: Comparativo de Volume entre AG+BL+MD, AGP7 e AGP8.

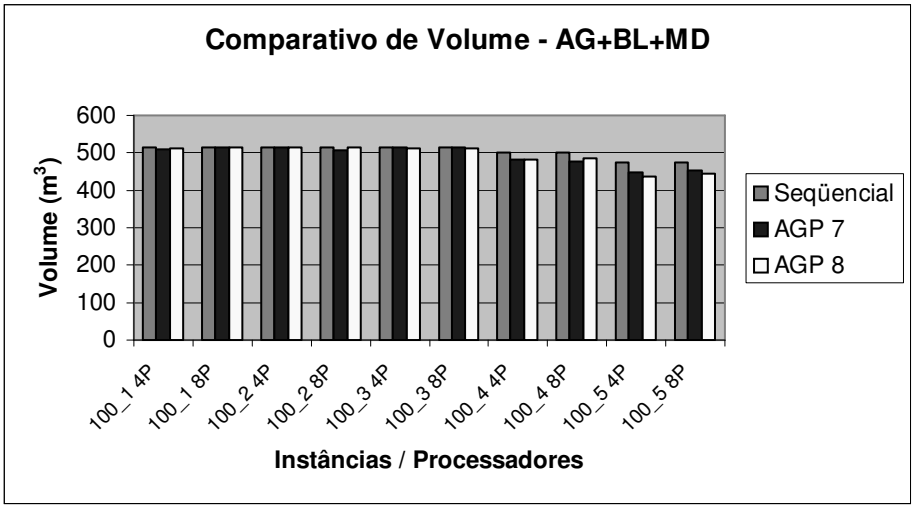

Figura 6.71: Comparativo de Volume entre AG+BL+MD, AGP7 e AGP8.

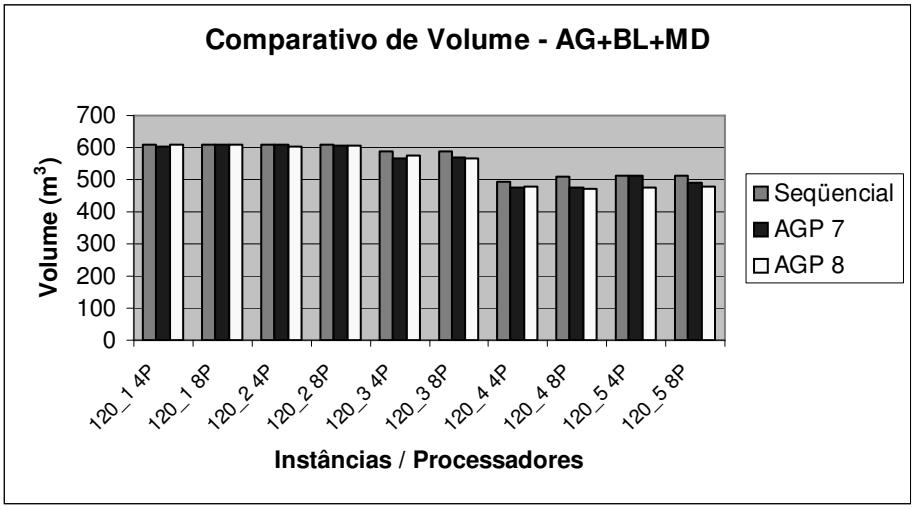

Figura 6.72: Comparativo de Volume entre AG+BL+MD, AGP7 e AGP8.

De acordo com o gráfico 6.73, os algoritmos paralelos AGP7 e AGP8 foram os que mais se beneficiaram com o paralelismo tendo em vista o tempo computacional em relação às demais versões. Outro aspecto importante foi que não houve melhora do tempo computacional quando aumentamos o número de processadores. Isto acontece, pois nos nossos algoritmos o paralelismo só é aplicado na Fase 2, portanto a eficiência do algoritmo diminui à medida que o número de processadores empregados aumenta, ou seja, cada processador ganha uma fatia de tempo menor da Fase 2 e, portanto, quanto maior for o número de processadores utilizados, menor será o tempo efetivo de execução de cada um deles na aplicação.

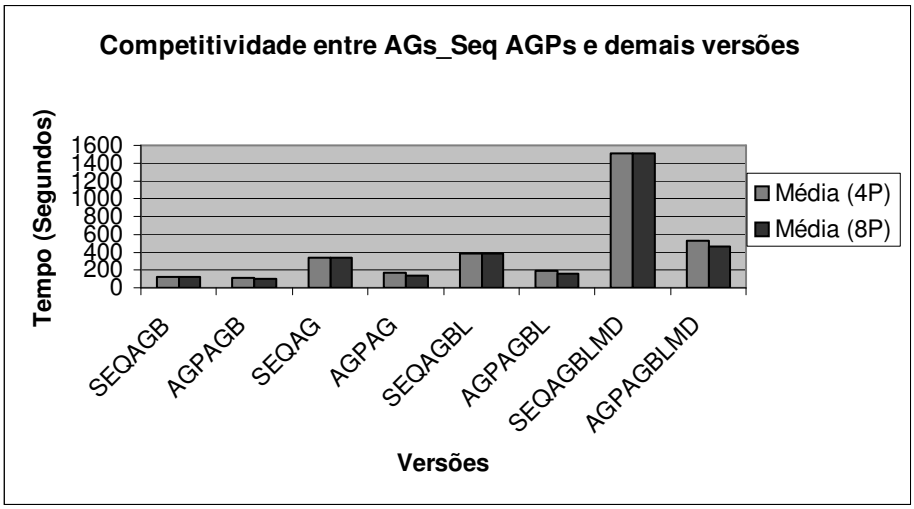

Figura 6.73: Comparação entre todas as versões de AGs.

A figura 6.73 mostra a média do tempo obtido em todas as instâncias dos seguintes algoritmos: SEQAGB (algoritmo AGB seqüencial); AGPAGB (versões paralelas de AGB); SEQAG (algoritmo AG seqüencial); AGPAG (versões paralelas de AG); SEQAGBL (algoritmo AG+BL seqüencial); AGPAGBL (versões paralelas de AG+BL); SEQAGBLMD (algoritmo AG+BL+MD seqüencial); AGPAGBLMD (versões paralelas de AG+BL+MD).

Capítulo 7 - Conclusões

Com base nos resultados experimentais realizados, podemos concluir que a proposta de incorporar módulos de mineração de dados num algoritmo evolutivo pode melhorar consideravelmente o seu desempenho em relação à qualidade das soluções geradas principalmente em instâncias com elevadas dimensões. Isto pode ser justificado pelo fato destes módulos reduzirem significativamente a região de busca em problemas altamente combinatórios. Com isso, podemos melhorar a convergência do algoritmo como também podemos, simultaneamente, permitir a exploração de ótimos locais de melhor qualidade, aumentando com isso as chances de se atingir um ótimo global em problemas de otimização. Este algoritmo, entretanto, apresenta os maiores tempos de execução.

A contribuição da MD quanto a qualidade alcançada pelo AG+BL+MD superou em mais de 100% o volume coletado pela versão AGB; e em mais de 60% em relação a versão com busca local e AG.

O módulo de busca local não teve uma contribuição tão significativa quanto a esperada. Isto provavelmente acontece por causa do emprego de heurísticas eficientes como a VMP, que na maioria das vezes, consegue apresentar bons resultados e por isso, o módulo de busca local não se faz necessário.

Quanto ao aspecto do paralelismo, o algoritmo AG+BL+MD foi o que mais se beneficiou com a paralelização, apresentando os melhores resultados. Este fato o tornou mais competitivo em relação às outras versões, quando consideramos o tempo de execução. Os AGPs com comunicação não apresentaram na média melhoria na qualidade da solução em relação aos AGPs sem comunicação.

Como sugestão para trabalhos futuros podemos incluir os seguintes tópicos: 1) Para melhorar os algoritmos seqüenciais, podemos propor o refinamento do algoritmo AG+BL+MD, para torná-lo mais econômico em relação ao tempo consumido, usando, por exemplo, uma busca local mais leve e/ou a sua ativação em intervalos maiores, principalmente quando o AG estiver utilizando operadores de reprodução eficientes, como a heurística VMP. Outro aspecto a ser considerado é limitar a aplicação dos

padrões obtidos pelo procedimento de MD a alguns indivíduos de uma população ao invés de aplicá-los em todos os indivíduos, como foi feito neste trabalho. Isso possivelmente nos permitiria obter soluções de qualidades similares às obtidas pelo AG+BL+MD, mas com uma redução maior nos tempos exigidos. 2) Para melhorar os algoritmos paralelos com comunicação, poderíamos tentar obter melhores resultados através do aumento do número de indivíduos migrados e o número de mensagens trocadas entre os processadores. Isto, possivelmente, dará mais chance para que estes indivíduos possam contribuir para a melhoria da solução final. Entretanto isto implicará num aumento no tempo paralelo. Outra alternativa seria paralelizar a etapa conversão do grafo para grafo completo, pois esta etapa seria a responsável pelos altos custos computacionais da Fase 1. Desta maneira, possivelmente a eficiência obtida seria maior e aproveitaríamos melhor o paralelismo.

Capítulo 8 - Referências

[1] AGARWAL, R.; AGGARWAL, C.; PRASAD, V. A tree projection algorithm for generation of frequent itemsets. In: Proceedings of High Performance Data Mining Workshop, Puerto Rico, 1999.

[2] AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A. Mining association rules between sets of items in large databases. Proc. of the ACM SIGMOD Conf. on Management of data, Washington, DC, USA, 1993.

[3] ALOISE, D. J.; SANTOS. A. C.; BARROS, C.A.; SOUZA, M. C.; NORONHA, T.F. Um algoritmo GRASP reativo aplicado ao problema da Unidade Móvel de Pistoneio (POE-UMP) In: Simpósio Brasileiro de Pesquisa Operacional XXXIII, Campos de Jordão, São Paulo, Nov. 2001.

[4] ALOISE, A.; ALOISE, D. J.; OCHI, L. S.;MAIA, R. S.; BITTENCOURT, V. G. Uma "Colônia de Formigas" para o Problema de Explotação de Petróleo e Otimização de Rotas de Unidades Móveis de Pistoneio. Anais do Congresso Brasileiro de Automação (CBA), 2002, Natal, RN, 2002.

[5] AREIBI, S.; MOUSSA, M.; ABDULLAH, H. A Comparison of Genetic/Memetic Algorithms and Heuristic Searching International Conference on Artificial Intelligence IC-AI 2001, Las Vegas, Nevada, June 25, 2001.

[6] BALUJA, S. Structured and performance of fine-grain parallelism in genetic search. In Stephanie Forrest, Editor, Proceedings of the Fifth International Conference on Genetic Agorithms, San Mateo, CA: Morgan Kaufmann Publishers, p.155-162, 1983.

[7] CANTÙ-PAZ, E. A summary of Research on Parallel Genetic Algorithms. IlliGAL Report Nº. 95007, University of Illinois, Illinois, USA, July 1995.

[8] CHAMBERS, L. (eds). Practical Handbook of Genetic Algorithms Vol. I CRC Press, Boca Raton, 1995.

[9] DAVIS, L. (Ed). Handbook of Genetic Algorithms. New York: Van Nostrand Reinhold, 1991.

[10] DRUMMOND, L. M. A.; VIANNA, L. S.; SILVA, M. B.; OCHI, L. S. Distributed Parallel Metaheuristic based on GRASP and VNS for solving The Traveling Purchaser Problem. In Proc. of the 2002 Int. Conf. On Parallel and Distributed Systems – ICPADS, sponsored by IEEE Computer Society, Taiwan, China, 2002.

[11] FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. From data mining to knowledge discovery: an overview. Em advances in Knowledge Discovery & Data Mining, pp. 1-34, 1996.

[12] FERREIRA, L. A. Estratégias de paralelização do GRASP para um Problema de coleta Seletiva de Prêmios. Niterói, 2002. Dissertação (Mestrado em Ciência da Computação), Departamento de Ciência de Computação, Universidade Federal Fluminense.

[13] FERREIRA, L. A.; DRUMMOND, L. M. A.; OCHI, L. S. Parallel metaheuristics based on GRASP and VNS for solving The Orienteering Problem. In Proc. of the PAREO 2002: Working Group on Parallel Processing in Operations Research, France, 2002.
[14] FERREIRA, L. A.; DRUMMOND, L. M.; OCHI, L. S. Strategies for the Parallel GRASP and VNS Metaheuristics for the orienteering problem. Proc of XI CLAIO – Latin-Ibero American Congress on Operations Research, Concepión, Chile, 2002.
[15] FILHO, J. L. R. GAME's Library Structure. In Parallel Genetic Algorithms: Theory and Applications(Ed. J.Stender), IOS Press Holland, 1993.
[16] FILHO, R; LORENA, G. A constructive genetic algorithm for graph coloring. Presented at APORS'97 conference, Melbourne /Australia, 1997.
[17] FLOCKHART, I. W.; RADCLIFFE, N. J. GA-MINER: parallel data mining with hierarchical genetic algorithms - final report. From EPCC-AIKMS-GA-MINER-Report 1.0., University of Edinburgh, 1995.
[18] FOGEL, D. B. (eds). Evolutionary Computation: The Fossil Record. IEEE Press, Piscataway, NJ, 1998.
[19] FOGEL, L. J.; ANGELINE, P, J.; BÄCK, T. (Eds.): Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming, San Diego, CA, USA, February 29 - March 2, 1996. MIT Press, 1996, ISBN 0-262-06190-2.
[20] FREITAS, A. A.; LAVINGTON, S. H. Mining Very Large Databases with Parallel Processing, Kluwer Academic Publishers, 1998.
[21] GLOVER, F. Heuristics for Integer Programming using surrogate constraints, Decision, 1977.
[22] GLOVER, F. Scatter Search and Star-Paths: Beyond the Genetic Metaphor, OR SPEKTRUM, v.17, 1995.
[23] GLOVER, F. Tabu Search - Part I, ORSA Journal on Computing, 1(3), 190-206, 1989.
[24] GLOVER, F. Tabu Search - Part II, ORSA Journal on Computing, v.2, p. 4-32, 1990.
[25] GLOVER,F. Tabu Search and Adaptive memory programming: Advances, applications and challenges, Interfaces in Computer Science and Oper. Res., Kluwer Academic Publ., pp.1-75, 1996.
[26] GLOVER, F. Tabu Search for Nonlinear and Parametric Optimization with links to Genetic Algorithms, Discrete Applied Mathematics, 49, pp. 231-255, 1994.
[27] GLOVER, F. Tabu Search Fundamentals and uses, University of Colorado at Boulder. Sciences, vol.8 (nº1), p. 156-166, 1995.
[28] GLOVER, F.; LAGUNA, M.; MARTI, R. Fundamentals of Scatter Search and Path Relinking. In Control and Cybernetics, vol. 39(3), 653-684, 2000.
[29] GORGES-SCHLEUTER, M. ASPARAGOS: An asynchronous parallel genetic optimization strategy. In Proc. of 3rd ICGA'89, Morgan Kaufmann Publ., p. 422-427, 1989.
[30] GROSS, J.; YELLEN, J. Graph Theory and Its Applications. Boca Raton, FL: CRC Press, 1999. 600 p.
[31] HOLLAND, J.H. Adaptation in natural and artificial systems, Univ. of Michigan Press, Ann Arbor, 1975.

[32] HOUTSMA, M.; SWAMI, A. Set-Oriented mining of association rules. Research Report RJ 9567, IBM Almaden Research Center, San Jose, California, October 1993.
[33] JOSHI, M. V.; KARYPIS, G.; KUMAR, V. Efficient Parallel Algorithms for Mining Associations, Lectures Notes in Computer Science / Lectures Notes in Artificil Intelligence, 2000.
[34] KOZA, J. R. Genetic Programming, MIT Press, MA, 1992.
[35] KOZA, J.R. Genetic Programming - 2, MIT Press, MA, 1994.
[36] LEVINE, D. A Parallel Genetic Algorithm for the Set Partitioning Problem, PhD Thesis, Illinois Institute of Technology, 1994.
[37] LOPES, Carlos H. Classificação de Registros em Banco de dados por evolução de Regras de associação utilizando algoritmos Genéticos. Rio de Janeiro, 1999. Dissertação (Mestrado), PUC, Pontifícia Universidade Católica do Rio de Janeiro.
[38] LYRA, A. R.; OCHI, L. S. Reduction Rules for the Prize Collecting Converting Tours Problem. Proc of the First International Conference on Optimization Methods and Software, Hangzhou, China, 2002.
[39] MARIN, F.; TRELLES-SALAZAR, O; SANDOVAL, F. Genetic Algorithms on lan-message passing architetures using pvm: Application to the routing problem. In: Y. Davidor, H.-P. Schwefel, and R. Männer, editors, Parallel Problem Solving from Nature – PPSN III, v. 866 of Lecture Notes in Computer Science, Berling , Germany: Springer- Verlag., p. 191-198, 1994.
[40] MCDONNELL, J. R.; REYNOLDS, R. G.; FOGEL, D. B. (Eds.): Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming, San Diego, CA, USA, March 1-3, 1995. A Bradford Book, MIT Press. Cambridge, Massachusetts. 1995, ISBN 0-262-13317-2
[41] MENDES, I. M. B.; PLASTINO, A.; OCHI, L. S. Regras de Associação: suas Diferentes Formas e seus Algoritmos de Mineração, Instituto de Computação – Universidade Federal Fluminense, Setembro, 2002.
[42] MICHALEWICZ, Z. Genetic Algorithms + Data Structures = Evolution Programs. (Third edition), Springer, 1996.
[43] MICHALEWICZ, Z.; FOGEL, D. How to solve it: Modern heuristics. Springer, 2000, 469 p.
[44] MOSCATO, P. On Evolution, Search, Optimization Algorithms and Martial Arts: Towards Memetic Algorithms. Report 826, Caltech Concurrent Computation Program, California Institute of Technology, Pasadena, 1989.
[45] NÉRI, F.; GIORDANA, A. A parallel Genetic Algorithm for concept learning. Proc. ICGA-95, 436-443, 1995.
[46] NEVES, A. J. Uma aplicação de algoritmo genético na otimização do emprego da unidade móvel de pistoneio. Natal, 2000. Dissertação (Mestrado), Departamento de Informática e Matemática Aplicada, UFRN, Universidade Federal do Rio Grande do Norte.
[47] NORONHA, T. F.; ALOISE, D. J. Algoritmos e Estratégias de solução para o problema do gerenciamento de sondas de produção terrestre na Bacia Petrolífera de Potiguar, XXI Congresso da Sociedade Brasileira de Computação, Fortaleza, CE, 2001.

[48] OCHI, L.S.; MONTENEGRO, A. A.; SANTOS, E.M.; MACULAN, N. A new GA for the TPP, Proc. of the Metaheuristic Int.Conf (MIC'95), Colorado-USA, Eds: H.Osman & J.Kelly, p. 52-57, Kluwer Academic Press, 1995.

[49] OCHI,L.S.; ROCHA, M.L. O problema de roteamento periódico de veículos: Uma abordagem via algoritmos genéticos, in Proc. of the XXIV SEMISH/SBC- Brasília-Brasil, 1997.

[50] OCHI,L.S.; SANTOS, A.M.; MARQUES, E. Design and Implementation of a Timetable System using Genetic Algorithm, Proc. of the 2nd Int. Automated Timetabling Conference – Toronto, 1997.

[51] OCHI, L. S.; SILVA, M. B; DRUMMOND, L. M.. Metaheuristics based on GRASP and VNS for solving the traveling purchaser problem. Proc of IV metaheuristic International Conference (MIC'2001), p. 489-494, Porto, Portugal, 2001.

[52] OCHI, L. S.; VIANNA, D. S.; DRUMMOND, L. M. A. An asynchronous parallel metaheuristic for the period vehicle routing problem. In Future Generations Computer Systems, Elsevier, v. 17, p. 379-386, 2001.

[53] PARK, J. S.; CHEN, M.; YU, P. S. An Effective Hash-Based Algorithm for Mining Associations Rules, Proceedings of the 21th Very Large Database Conference – VLDB'95, San Jose, 175-186, 1995.

[54] POTTER, M. A.; JONG, K. A. Cooperative Coevolution: An Architeture for Evolving Coadapted Subcomponents. Evolutionary Computation 8(1):1-29, The MIT Press, MA, 2000.

[55] POTTER, M. A.; JONG, K. A.; GREFENSTETTE, J. J. A Coevolutionary Approach to Learning Sequential Decision Rules. Proceedings of the Sixth International Conference (ICGA95), July 15-19, Pittsburgh, Pennsylvania. USA, 1995.

[56] ROCHA, P. S.; SOUZA, A. O.; CAMARA, R. J. O futuro da Bacia do Recôncavo, a mais antiga província petrolífera brasileira. SEI v.11, p. 32-44, Salvador, 2002.

[57] SAVASERE, A.; OMIECINSKI, E.; NAVATHE, S. An Efficient Algorithm for Mining Association Rules in Large Databases. VLDB 1995: p. 432-444, 1995.

[58] SAVASERE, A.; OMIECINSKI, E.; NAVATHE, S. Mining for strong negative association in large database of customer transactions. College of Computing, Georgia Institute of Technology, Atlanta GA 30332, 1998.

[59] SHAPIRO, B.; NAVETTA, J. A massively parallel genetic algorithm for RNA secondary structure prediction. The Journal of Supercomputing, v.8, p. 195-207, 1994.

[60] SHITANI, T.; KITSUREGAWA, M. Parallel Mining Algorithms for Generalized Association Rules with Classification Hierarchy, Proceedings of the ACM SIGMOD International Conference on Management of Data, 25-36, 1998.

[61] SHONKWILER, R. Parallel Genetic Algorithms, Proc. of the Fifth Int. Conf. on GA, University of Illinois at Urbana, 1993.

[62] SIPPER, M. Evolution of Parallel Cellular Machines: The Cellular Programming Approach, Springer-Verlag, Heidelberg, 1997.

[63] SIPPER, M. Co-evolving Non-Uniform Cellular Automata to Perform Computations, Physica D, v. 92, p. 193-208, 1996.

[64] SIPPER, M.; TOMASSINI, M. Generating Parallel Random Number Generators by Cellular Programing. International Journal of Modern Physics C, v. 7 (nº2), p.181-190, 1996.
[65] SIPPER, M.; RUPPIN, E. Co-evolving architectures for cellular machines, Physica D, v. 99, p. 428-441, 1997.
[66] SNIR, M.; OTTO, S.; HUSS –LEDERMAN, W., et al. MPI: The Complete. MIT Press, MA, USA, 1995.
[67] SRIKANT, R.; AGRAWAL, R. Mining Generalized Association Rules. In Proc. of the 21st Int'l Conference on Very Large Databases, Zurich, Switzerland, September 1995.
[68] TARGA, C. N. Mineração Eficiente de Regras de Associação através da Indexação de Conjuntos Candidatos. Niterói, 2002. Dissertação (Mestrado em Ciência da Computação), Universidade Federal Fluminense, Departamento de Ciência da Computação.
[69] TOIVONEN, H.; KLEMETTINEN, M.; RONKAINEN, P.; HATONEN, K.; MANNILA, H. Prunning and Grouping Discovered Association Rules, In: ECML-95 Workshop on Statistics, Machine Learning, and Knowledge Discovery in Databases, 47 - 52, Heraklion, Greece, April 1995.
[70] VIANNA, D. S. Um algoritmo evolutivo Paralelo para o problema de Roteamento de Veículos com Frota Heterogênea. Niterói, 2002. Dissertação (Mestrado em Ciência da Computação), Departamento de Ciência de Computação, Universidade Federal Fluminense.
[71] VIANNA, L. S.; SILVA, M. B.; DRUMMOND, L. M., OCHI, L. S. Sequencial and Parallel Metaheuristics based on GRASP and VNS for solving the traveling purchaser problem. Latin-Ibero American congress on Operations Research, Conception, Chile, 2002.
[72] WANG, P. C.; Korfhage, W. Process scheduling using genetic algorithms, IEEE Symposium on Parallel and Distributed Processing, San Antonio, USA, p. 638-641, 1995.
[73] ZAKI, M. J.; PARTHASARATHY, S.; OGIHARA
[74] , M.; LI, W. New Algorithms for Fast Discovery of Associations Rules, Proceedings of the 3rd Conference on Knowledge Discovery and Data Mining – KDD'97, California, 283-286, 1997.

Capítulo 9 – Apêndice “A”

Nas seções seguintes deste capítulo, apresentamos os gráficos comparativos de *Speed up* e *Eficiência* de cada uma das versões propostas neste trabalho.

9.1 Gráficos *Speed up* e *Eficiência* dos algoritmos AGP1 e AGP2.

As figuras à esquerda representam o *speed up*, e à direita, os gráficos de *eficiência* alcançados para cada um dos problemas executados em 4 e 8 processadores e que foram obtidos através dos algoritmos AGP1 e AGP2.

A seguir apresentamos gráficos comparativos de *speed up* e *eficiência* em detalhes, obtidos por esta versão.

Speed up **Eficiência**

50 Poços

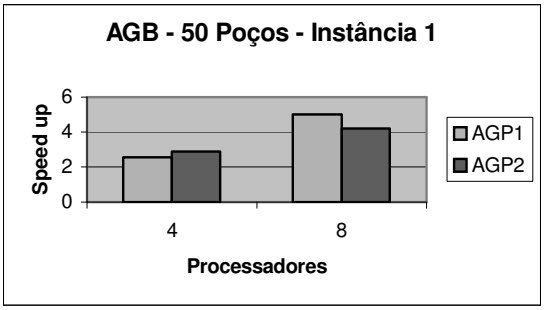

Figura 9.1: Comparativo de *Speed up* entre versões Paralelas do AGB

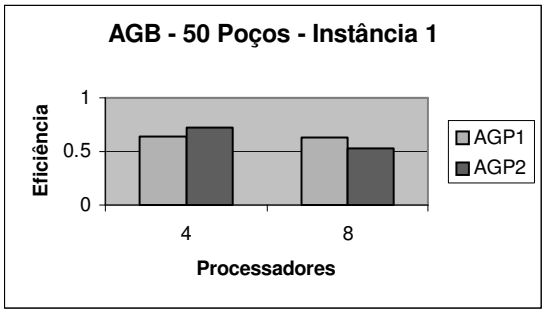

Figura 9.2: Comparativo de Eficiência entre versões Paralelas do AGB

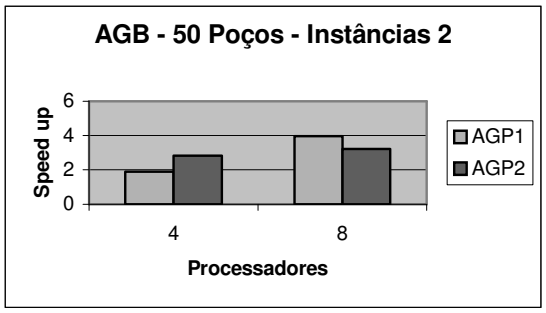

Figura 9.3: Comparativo de *Speed up* entre versões Paralelas do AGB

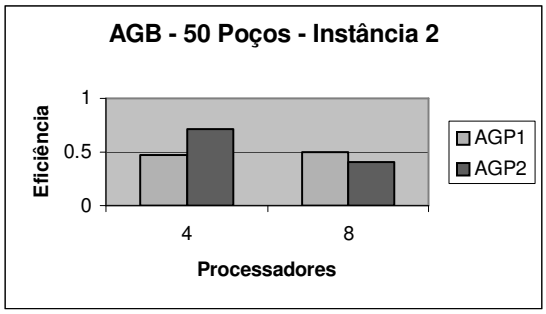

Figura 9.4: Comparativo de Eficiência entre versões Paralelas do AGB

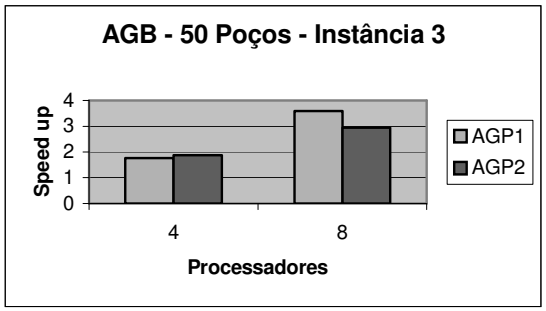

Figura 9.5: Comparativo de *Speed up* entre versões Paralelas do AGB

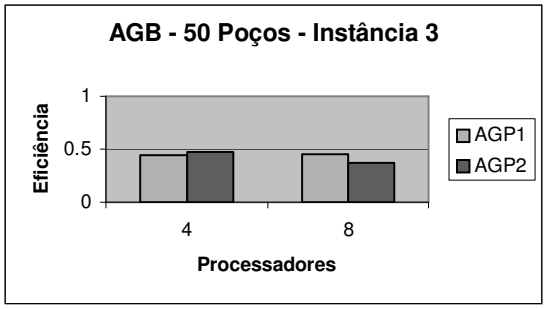

Figura 9.6: Comparativo de Eficiência entre versões Paralelas do AGB

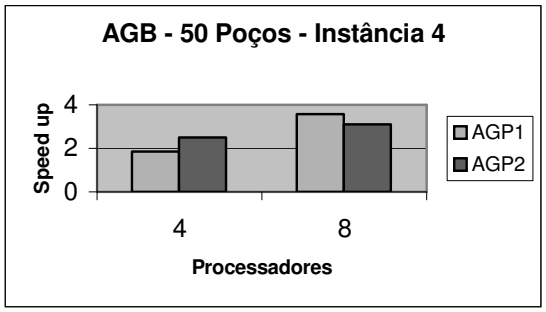

Figura 9.7: Comparativo de *Speed up* entre versões Paralelas do AGB

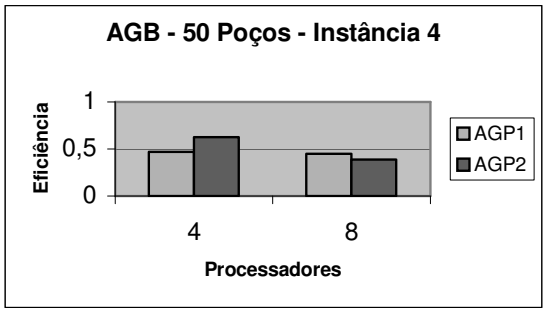

Figura 9.8: Comparativo de Eficiência entre versões Paralelas do AGB

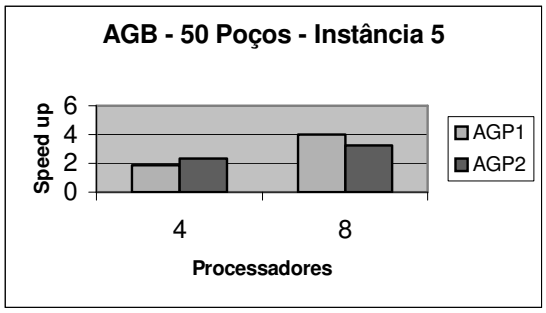

Figura 9.9: Comparativo de *Speed up* entre versões Paralelas do AGB

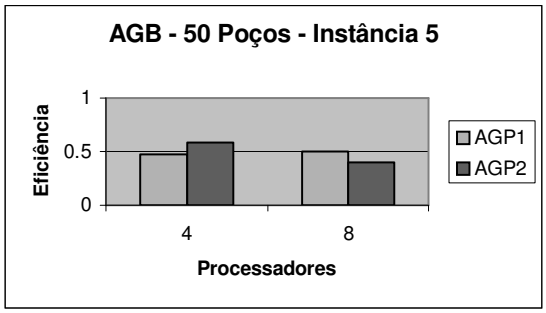

Figura 9.10: Comparativo de Eficiência entre versões Paralelas do AGB

Speed up **Eficiência**

70 Poços

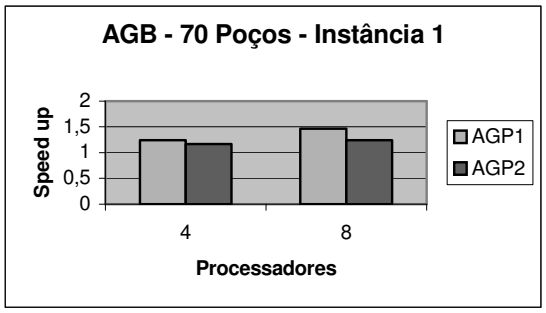

Figura 9.11: Comparativo de *Speed up* entre versões Paralelas do AGB

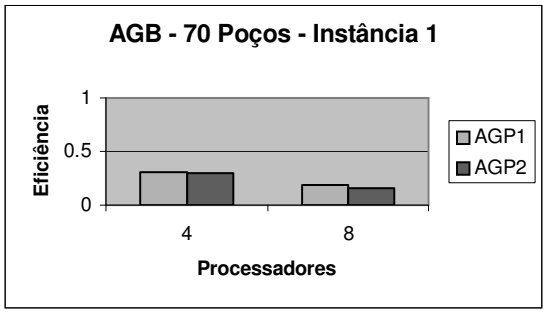

Figura 9.12: Comparativo de Eficiência entre versões Paralelas do AGB

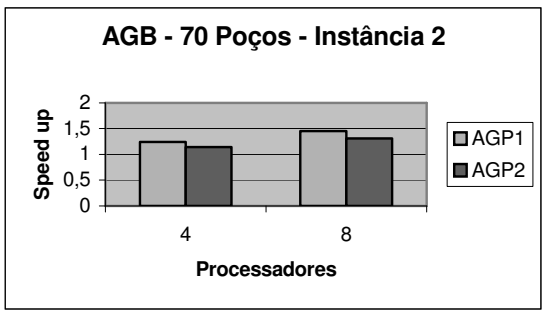

Figura 9.13: Comparativo de *Speed up* entre versões Paralelas do AGB

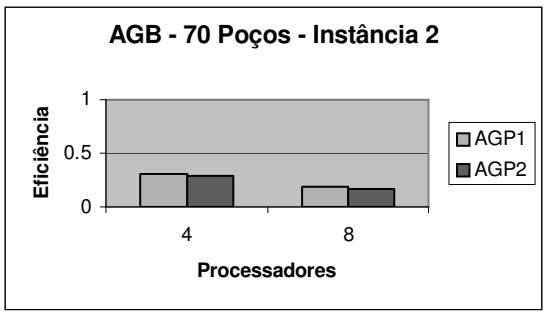

Figura 9.14: Comparativo de Eficiência entre versões Paralelas do AGB

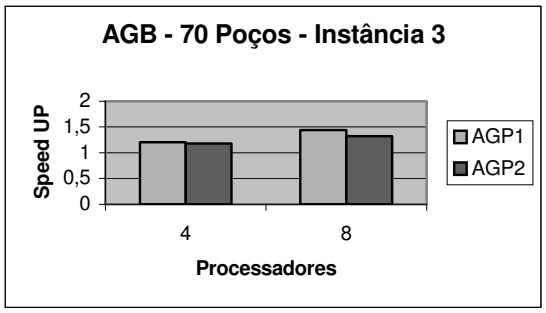

Figura 9.15: Comparativo de *Speed up* entre versões Paralelas do AGB

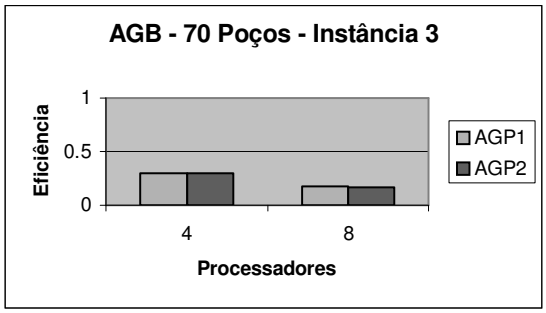

Figura 9.16: Comparativo de Eficiência entre versões Paralelas do AGB

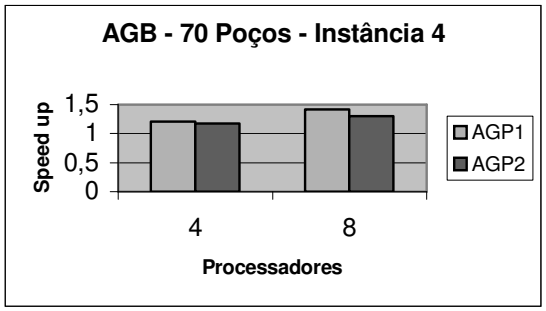

Figura 9.17: Comparativo de *Speed up* entre versões Paralelas do AGB

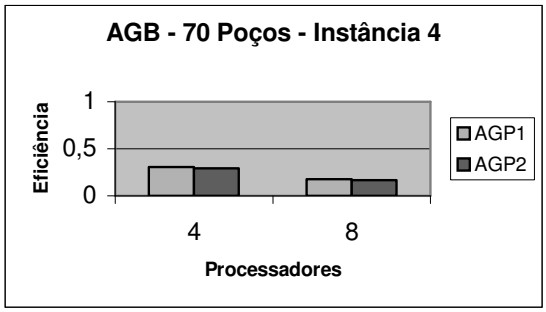

Figura 9.18: Comparativo de Eficiência entre versões Paralelas do AGB

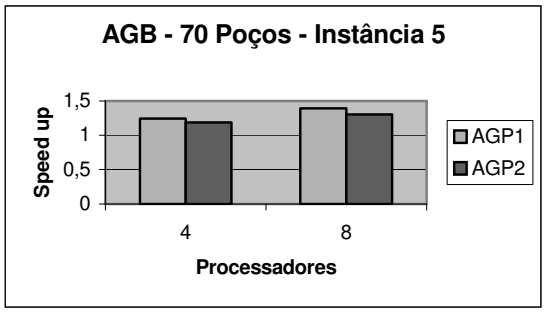

Figura 9.19: Comparativo de *Speed up* entre versões Paralelas do AGB

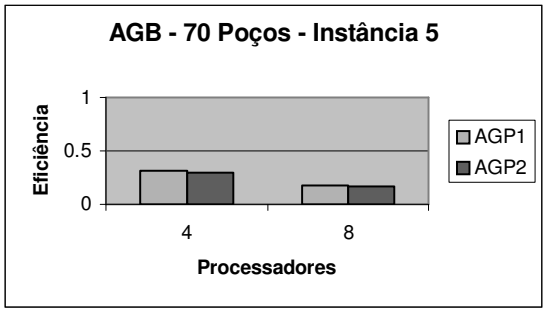

Figura 9.20: Comparativo de Eficiência entre versões Paralelas do AGB

Speed up **Eficiência**

100 Poços

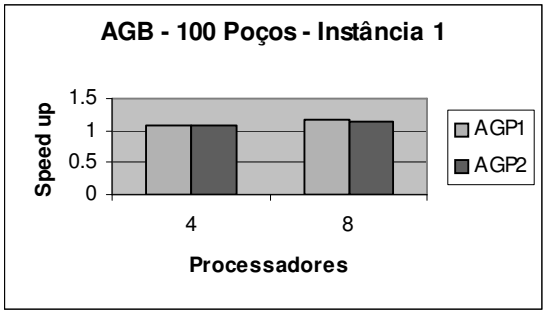

Figura 9.21: Comparativo de *Speed up* entre versões Paralelas do AGB

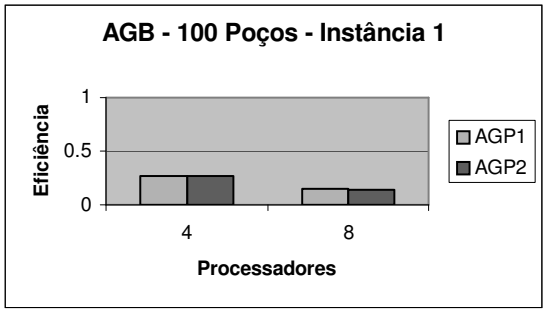

Figura 9.22: Comparativo de Eficiência entre versões Paralelas do AGB

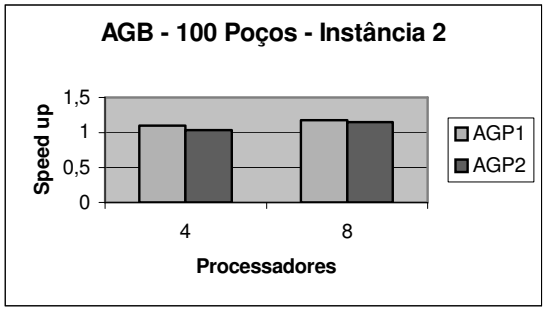

Figura 9.23: Comparativo de *Speed up* entre versões Paralelas do AGB

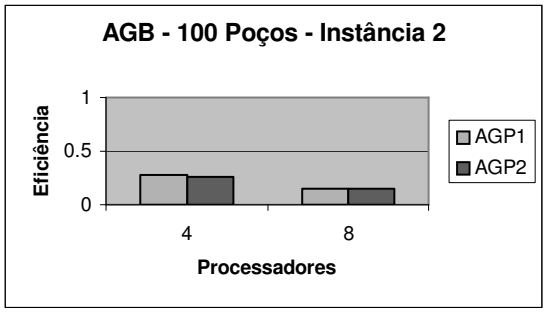

Figura 9.24: Comparativo de Eficiência entre versões Paralelas do AGB

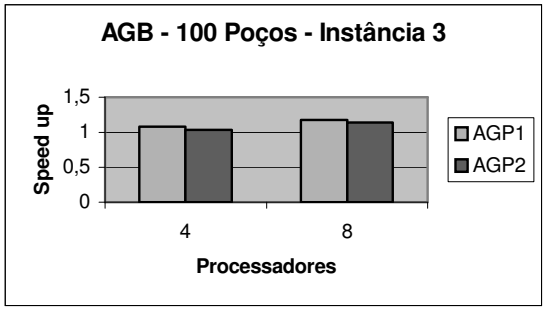

Figura 9.25: Comparativo de *Speed up* entre versões Paralelas do AGB

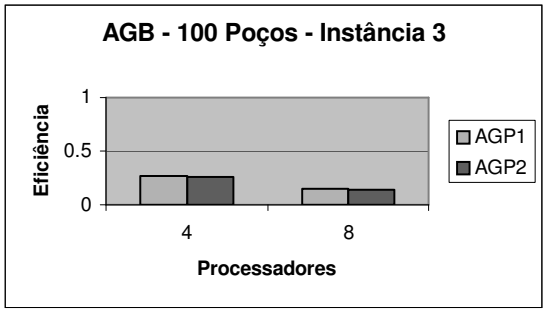

Figura 9.26: Comparativo de Eficiência entre versões Paralelas do AGB

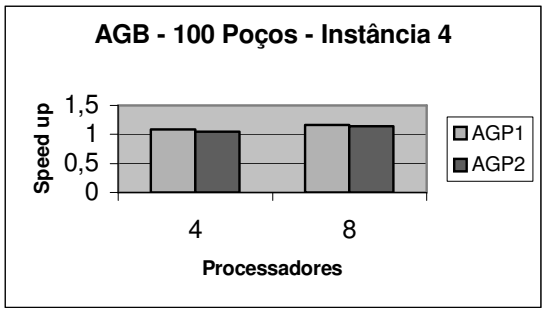

Figura 9.27: Comparativo de *Speed up* entre versões Paralelas do AGB

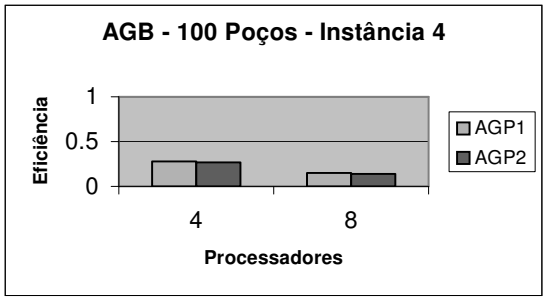

Figura 9.28: Comparativo de Eficiência entre versões Paralelas do AGB

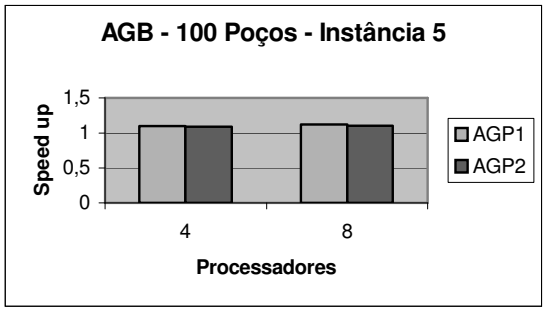

Figura 9.29: Comparativo de *Speed up* entre versões Paralelas do AGB

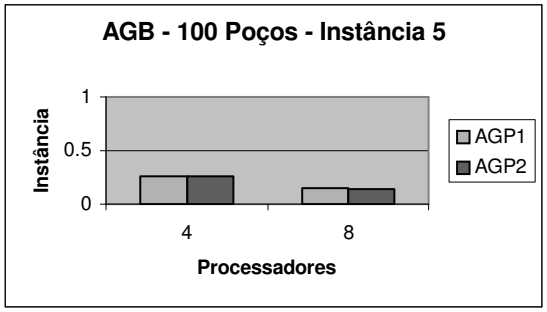

Figura 9.30: Comparativo de Eficiência entre versões Paralelas do AGB

Speed up **Eficiência**

120 Poços

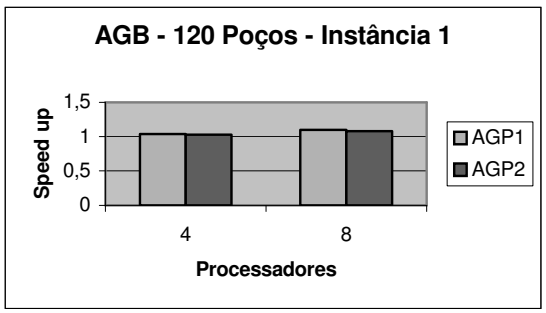

Figura 9.31: Comparativo de *Speed up* entre versões Paralelas do AGB

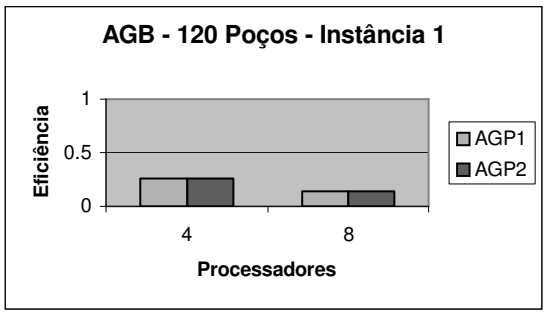

Figura 9.32: Comparativo de Eficiência entre versões Paralelas do AGB

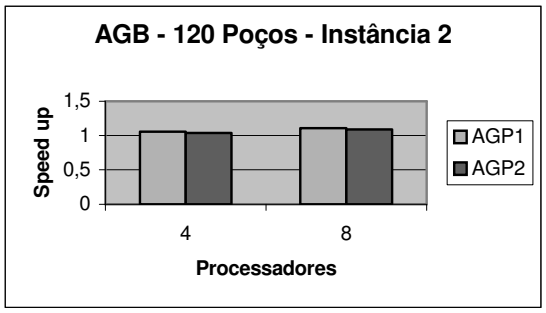

Figura 9.33: Comparativo de *Speed up* entre versões Paralelas do AGB

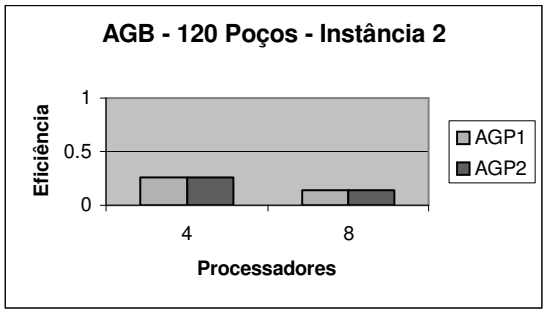

Figura 9.34: Comparativo de Eficiência entre versões Paralelas do AGB

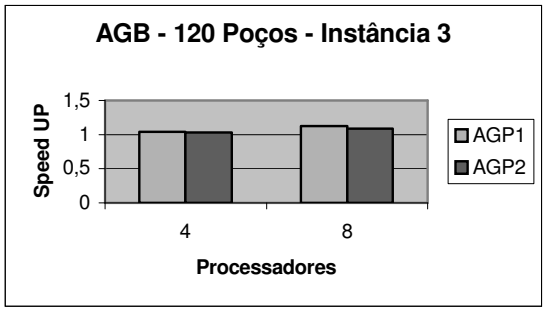

Figura 9.35: Comparativo de *Speed up* entre versões Paralelas do AGB

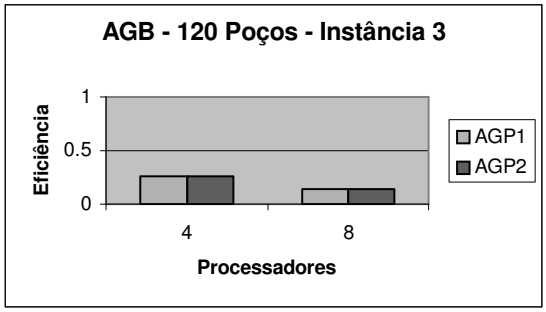

Figura 9.36: Comparativo de Eficiência entre versões Paralelas do AGB

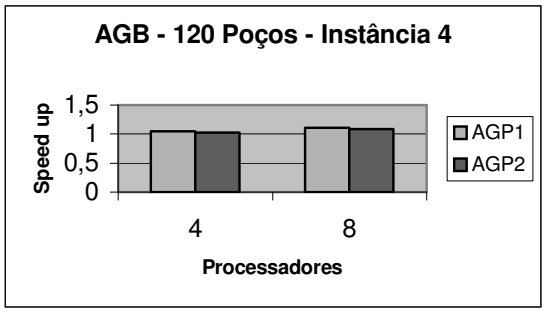

Figura 9.37: Comparativo de *Speed up* entre versões Paralelas do AGB

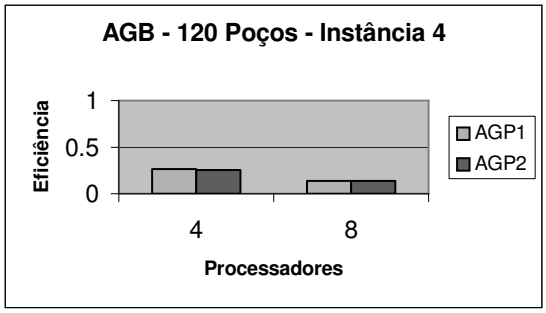

Figura 9.38: Comparativo de Eficiência entre versões Paralelas do AGB

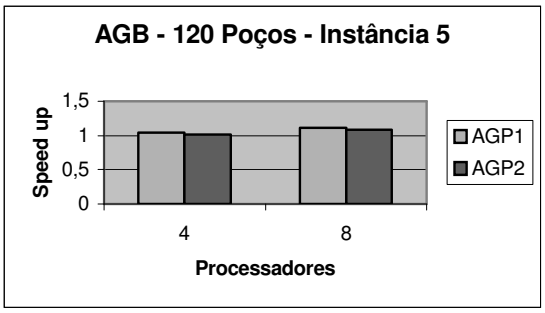

Figura 9.39: Comparativo de *Speed up* entre versões Paralelas do AGB

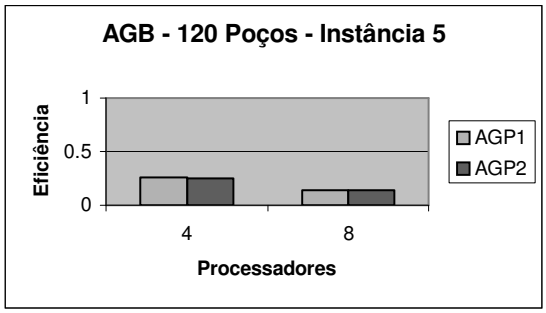

Figura 9.40: Comparativo de Eficiência entre versões Paralelas do AGB

9.2 Gráficos *Speed up* e *Eficiência* dos algoritmos AGP3 e AGP4.

As figuras à esquerda representam o *speed up*, e à direita, os gráficos de *eficiência* alcançados para cada um dos problemas executados em 4 e 8 processadores e que foram obtidos através dos algoritmos AGP3 e AGP4.

Speed up **Eficiência**

50 Poços

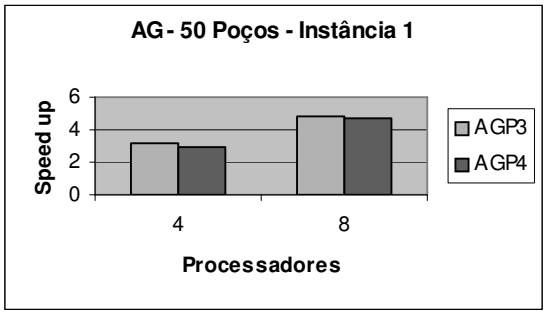

Figura 9.41: Comparativo de *Speed up* entre versões Paralelas do AG

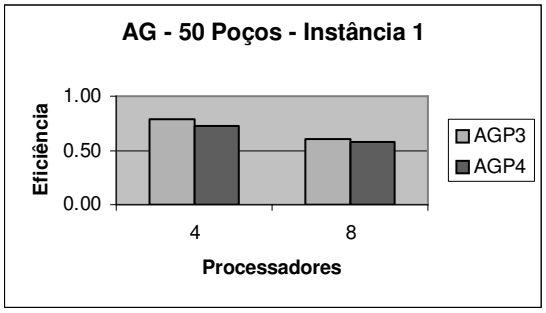

Figura 9.42: Comparativo de Eficiência entre versões Paralelas do AG

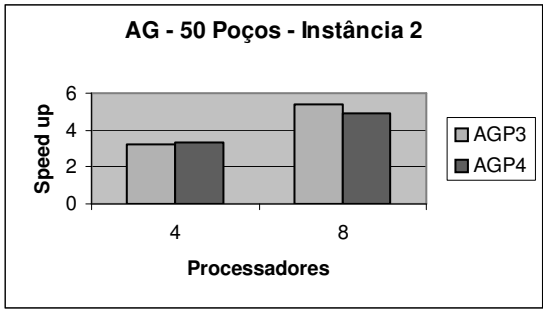

Figura 9.43: Comparativo de *Speed up* entre versões Paralelas do AG

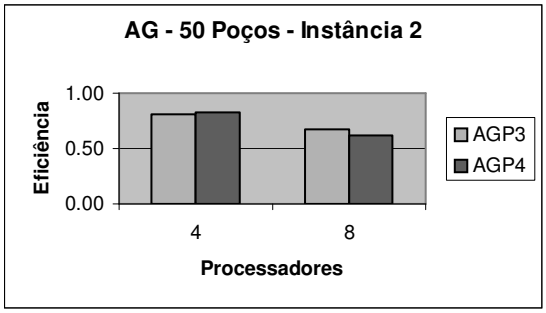

Figura 9.44: Comparativo de Eficiência entre versões Paralelas do AG

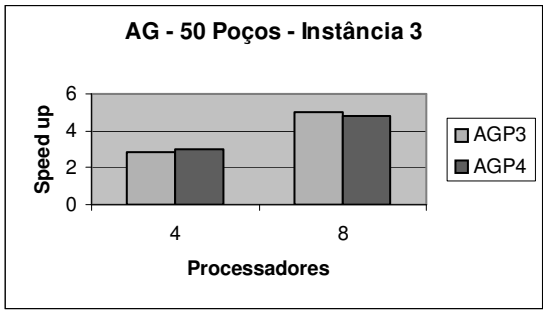

Figura 9.45: Comparativo de *Speed up* entre versões Paralelas do AG

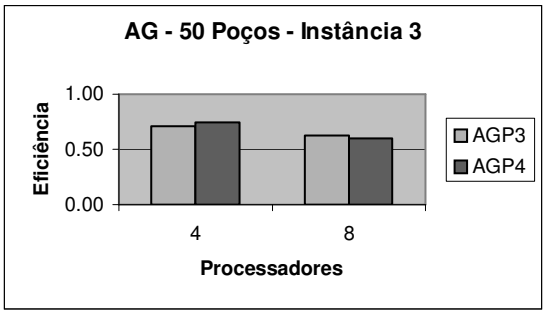

Figura 9.46: Comparativo de Eficiência entre versões Paralelas do AG

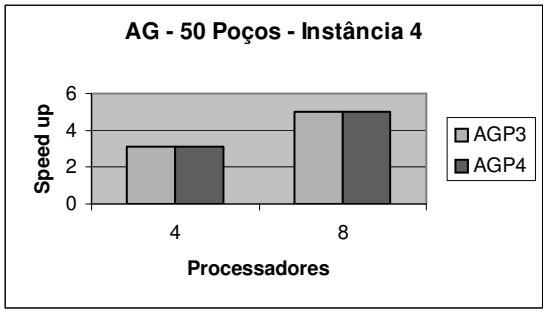

Figura 9.47: Comparativo de *Speed up* entre versões Paralelas do AG

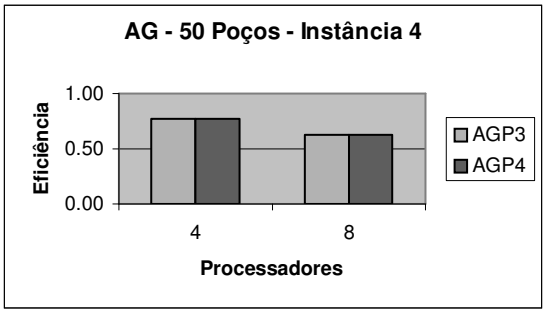

Figura 9.48: Comparativo de Eficiência entre versões Paralelas do AG

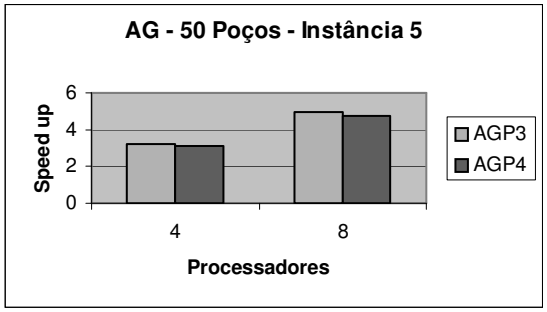

Figura 9.49: Comparativo de *Speed up* entre versões Paralelas do AG

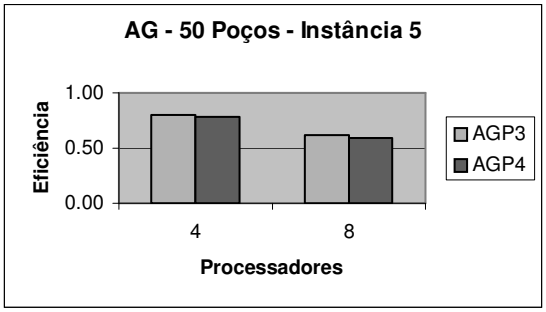

Figura 9.50: Comparativo de Eficiência entre versões Paralelas do AG

Speed up **Eficiência**

70 Poços

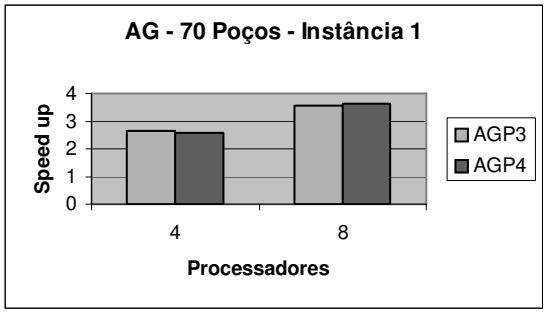

Figura 9.51: Comparativo de *Speed up* entre versões Paralelas do AG

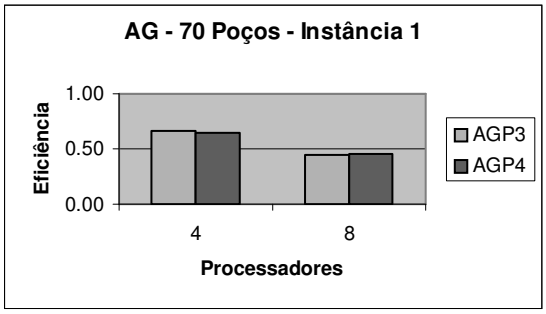

Figura 9.52: Comparativo de Eficiência entre versões Paralelas do AG

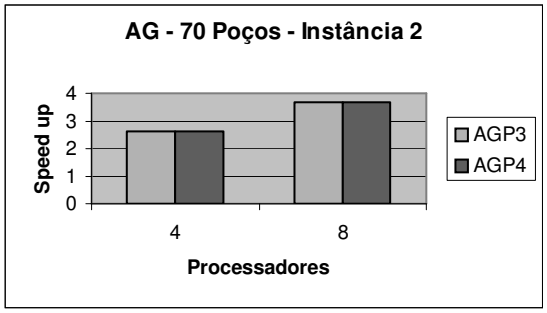

Figura 9.53: Comparativo de *Speed up* entre versões Paralelas do AG

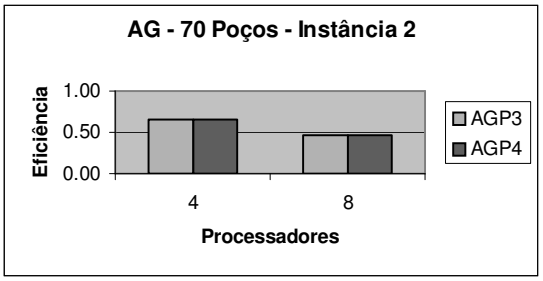

Figura 9.54: Comparativo de Eficiência entre versões Paralelas do AG

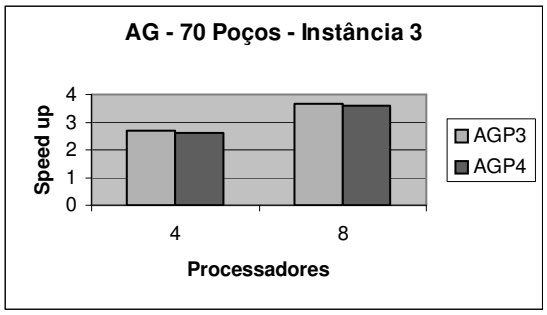

Figura 9.55: Comparativo de *Speed up* entre versões Paralelas do AG

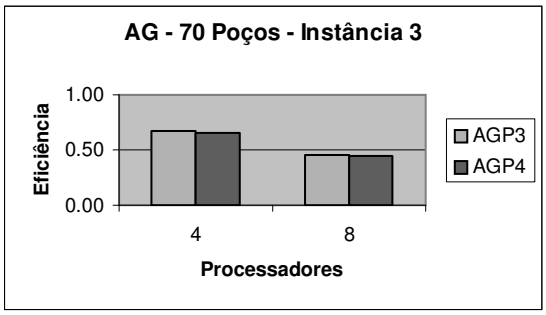

Figura 9.56: Comparativo de Eficiência entre versões Paralelas do AG

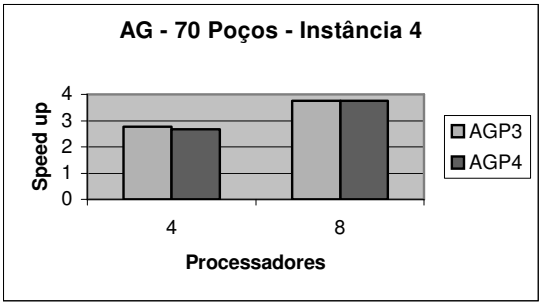

Figura 9.57: Comparativo de *Speed up* entre versões Paralelas do AG

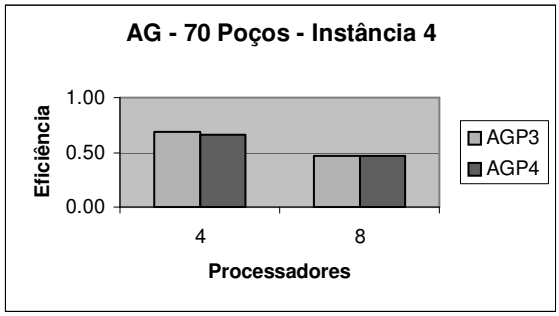

Figura 9.58: Comparativo de Eficiência entre versões Paralelas do AG

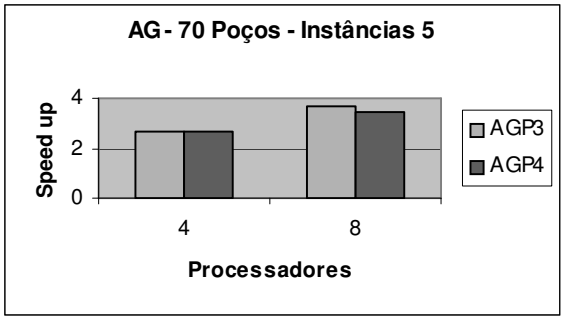

Figura 9.59: Comparativo de *Speed up* entre versões Paralelas do AG

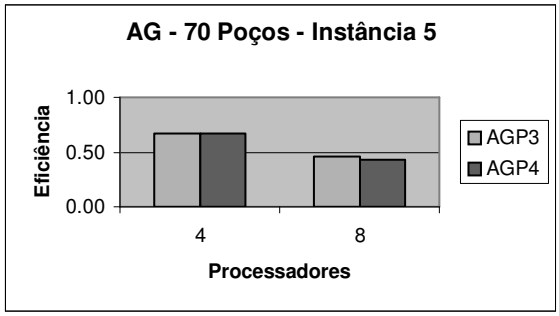

Figura 9.60: Comparativo de Eficiência entre versões Paralelas do AG

Speed up **Eficiência**

100 Poços

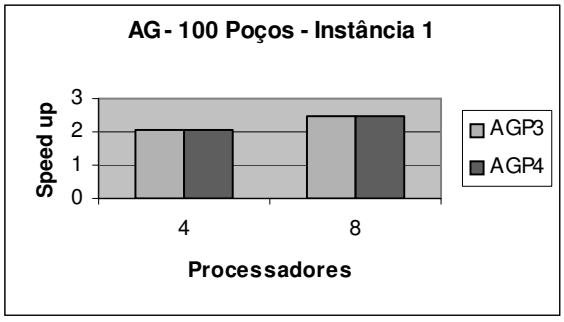

Figura 9.61: Comparativo de *Speed up* entre versões Paralelas do AG

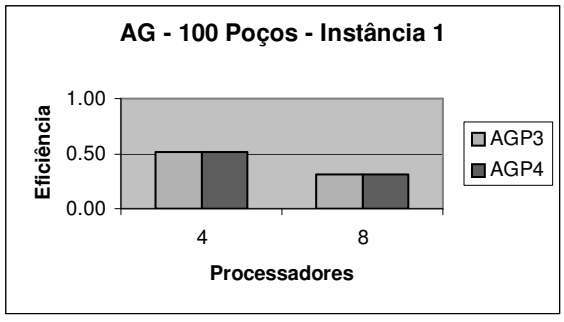

Figura 9.62: Comparativo de Eficiência entre versões Paralelas do AG

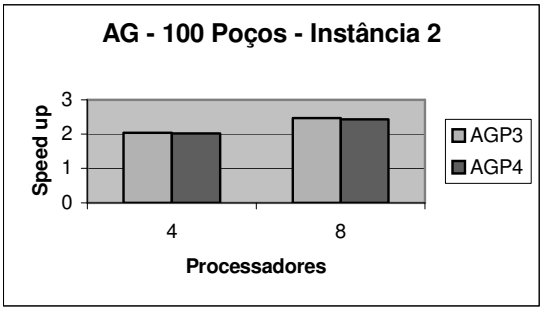

Figura 9.63: Comparativo de *Speed up* entre versões Paralelas do AG

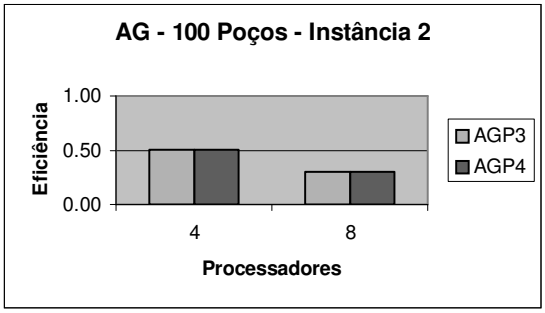

Figura 9.64: Comparativo de Eficiência entre versões Paralelas do AG

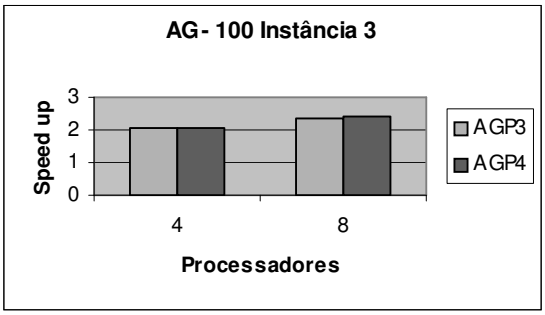

Figura 9.65: Comparativo de *Speed up* entre versões Paralelas do AG

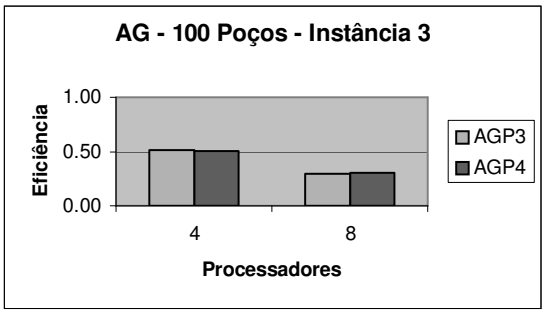

Figura 9.66: Comparativo de Eficiência entre versões Paralelas do AG

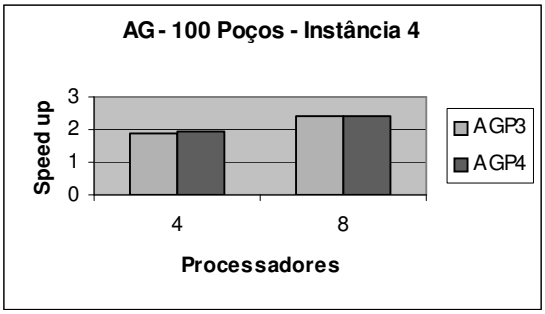

Figura 9.67: Comparativo de *Speed up* entre versões Paralelas do AG

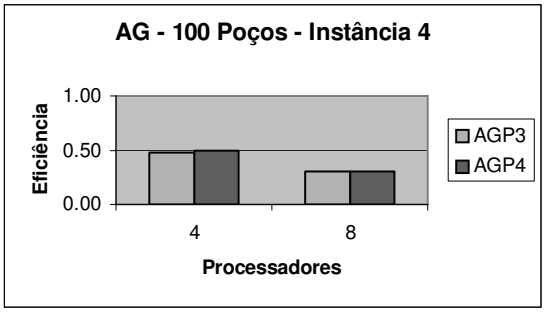

Figura 9.68: Comparativo de Eficiência entre versões Paralelas do AG

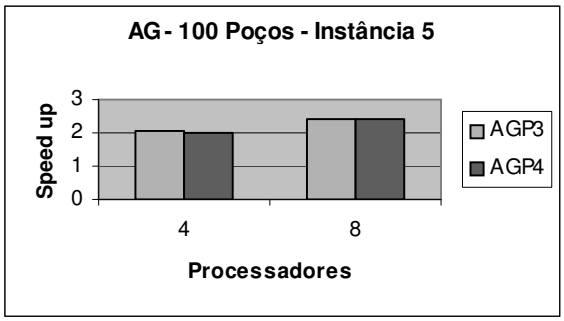

Figura 9.69: Comparativo de *Speed up* entre versões Paralelas do AG

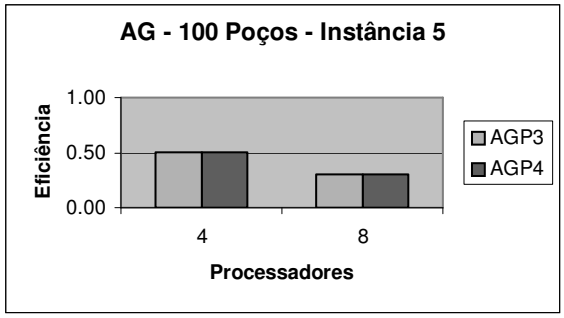

Figura 9.70: Comparativo de Eficiência entre versões Paralelas do AG

Speed up **Eficiência**

120 Poços

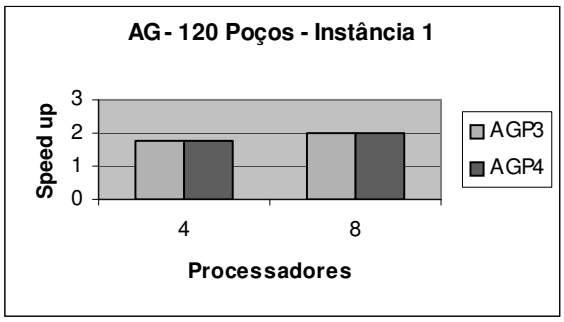

Figura 9.71: Comparativo de *Speed up* entre versões Paralelas do AG

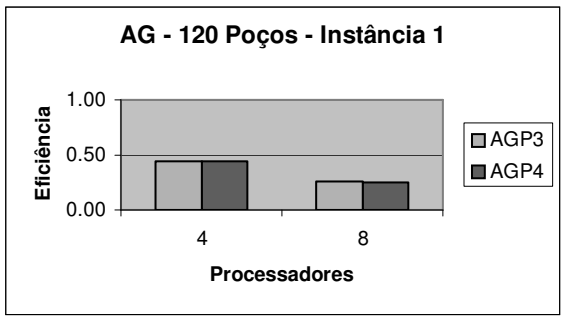

Figura 9.72: Comparativo de Eficiência entre versões Paralelas do AG

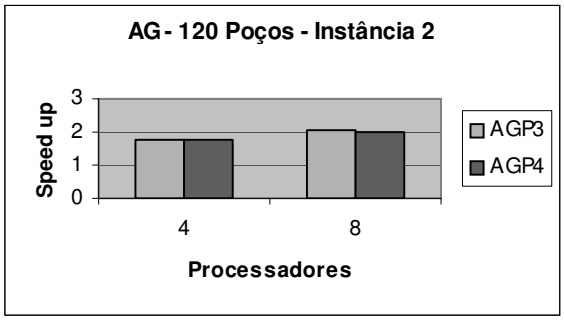

Figura 9.73: Comparativo de *Speed up* entre versões Paralelas do AG

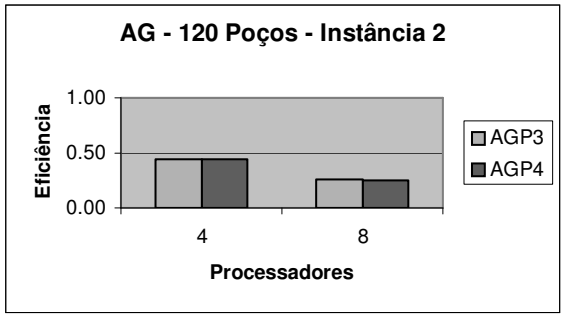

Figura 9.74: Comparativo de Eficiência entre versões Paralelas do AG

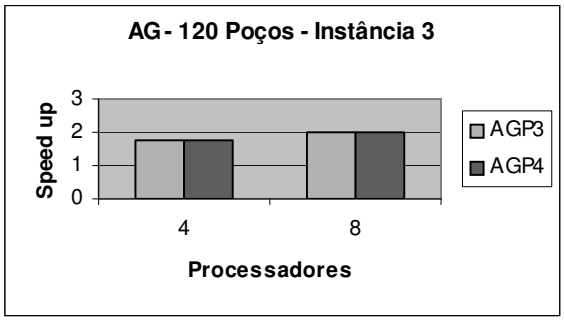

Figura 9.75: Comparativo de *Speed up* entre versões Paralelas do AG

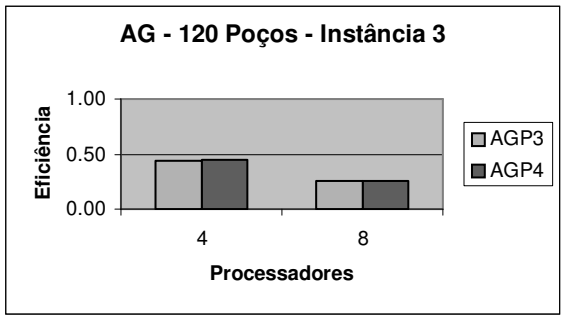

Figura 9.76: Comparativo de Eficiência entre versões Paralelas do AG

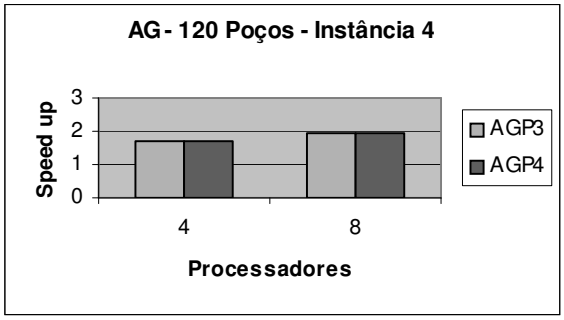

Figura 9.77: Comparativo de *Speed up* entre versões Paralelas do AG

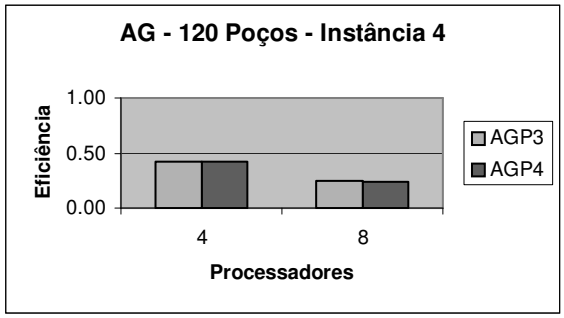

Figura 9.78: Comparativo de Eficiência entre versões Paralelas do AG

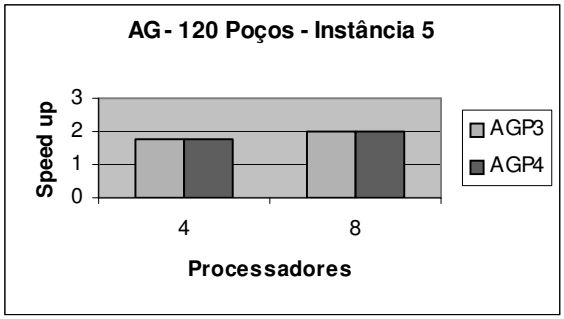

Figura 9.79: Comparativo de *Speed up* entre versões Paralelas do AG

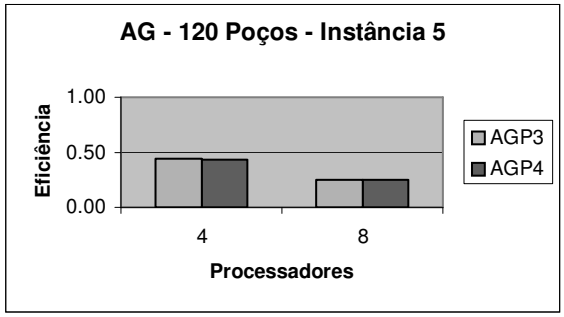

Figura 9.80: Comparativo de Eficiência entre versões Paralelas do AG

9.3 Gráficos *Speed up* e *Eficiência* dos algoritmos AGP5 e AGP6.

As figuras à esquerda representam o *speed up*, e à direita, os gráficos de *eficiência* alcançados para cada um dos problemas executados em 4, 8 e 16 processadores e que foram obtidos através dos algoritmos AGP5 e AGP6.

Speed up **Eficiência**

50Poços

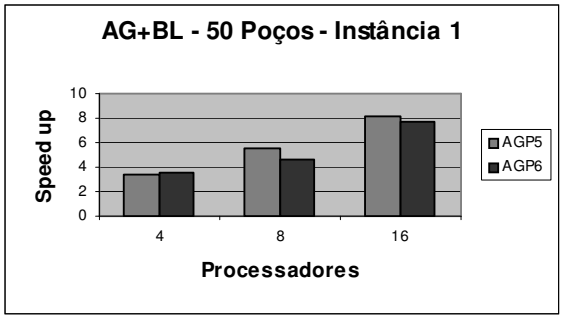

Figura 9.81: Comparativo de *Speed up* entre versões Paralelas do AG + BL

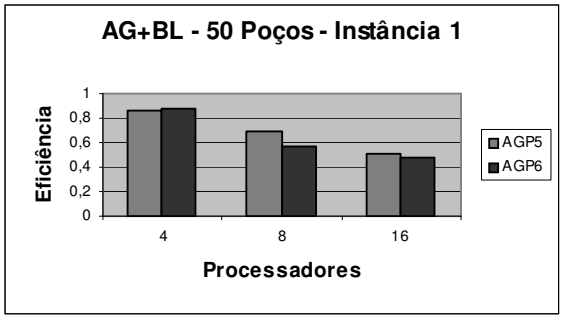

Figura 9.82: Comparativo de Eficiência entre versões Paralelas do AG + BL

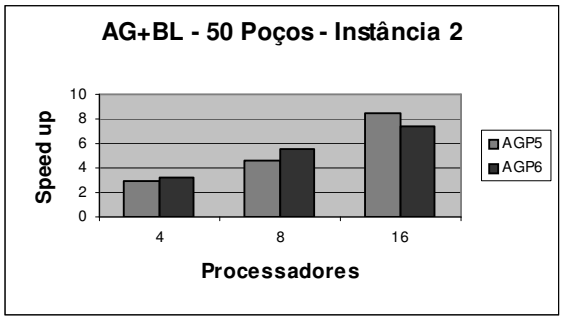

Figura 9.83: Comparativo de *Speed up* entre versões Paralelas do AG + BL

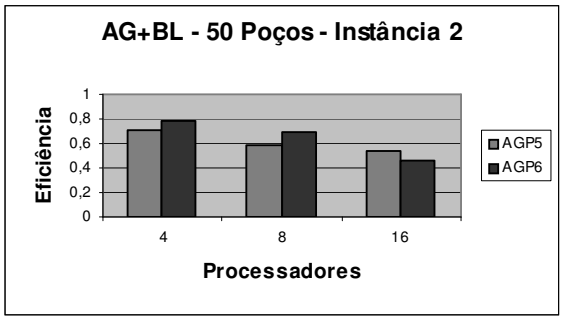

Figura 9.84: Comparativo de Eficiência entre versões Paralelas do AG + BL

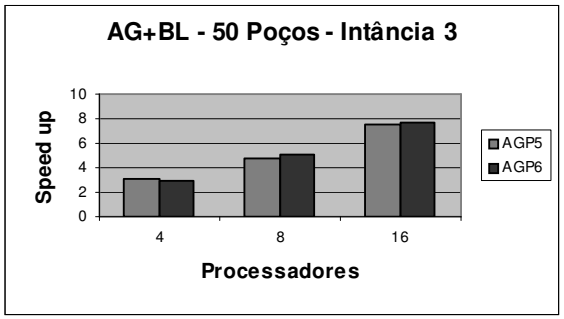

Figura 9.85: Comparativo de *Speed up* entre versões Paralelas do AG + BL

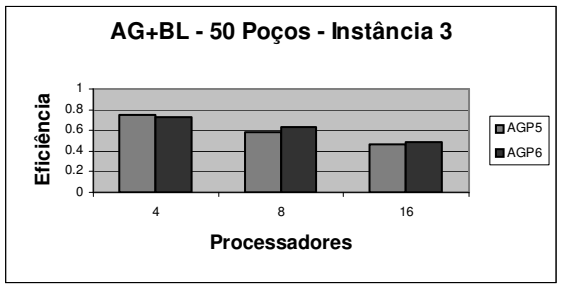

Figura 9.86: Comparativo de Eficiência entre versões Paralelas do AG + BL

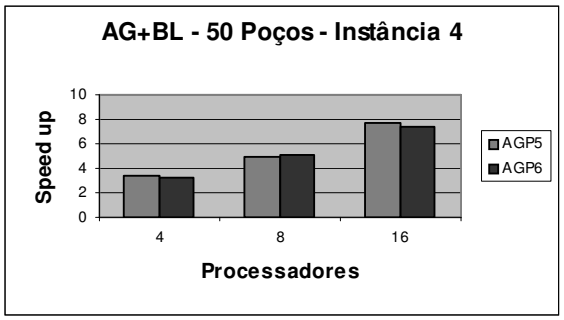

Figura 9.87: Comparativo de *Speed up* entre versões Paralelas do AG + BL

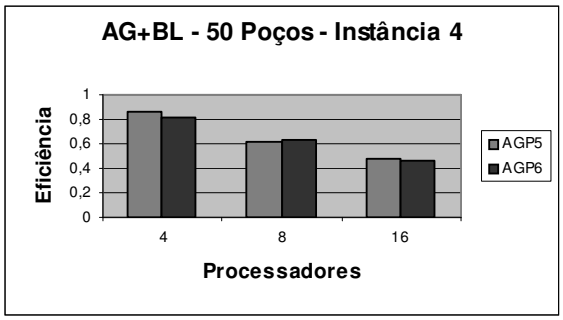

Figura 9.88: Comparativo de Eficiência entre versões Paralelas do AG + BL

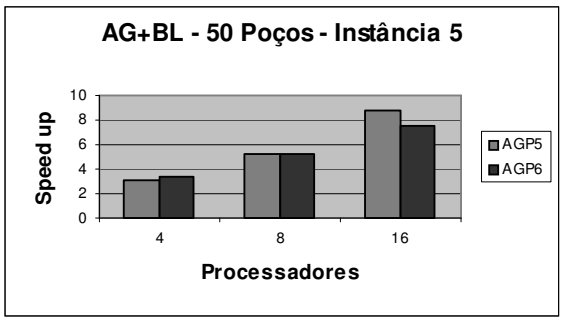

Figura 9.89: Comparativo de *Speed up* entre versões Paralelas do AG + BL

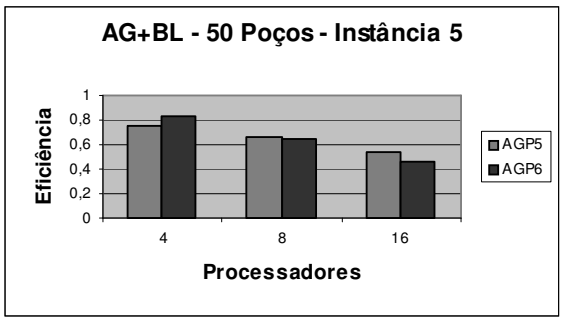

Figura 9.90: Comparativo de Eficiência entre versões Paralelas do AG + BL

Speed up **Eficiência**

70 Poços

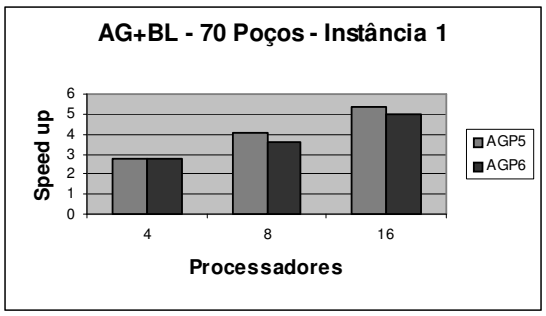

Figura 9.91: Comparativo de *Speed up* entre versões Paralelas do AG + BL

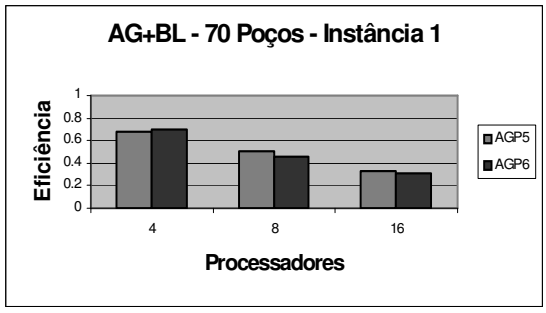

Figura 9.92: Comparativo de Eficiência entre versões Paralelas do AG + BL

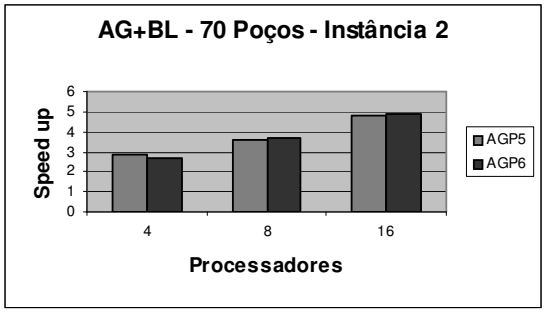

Figura 9.93: Comparativo de *Speed up* entre versões Paralelas do AG + BL

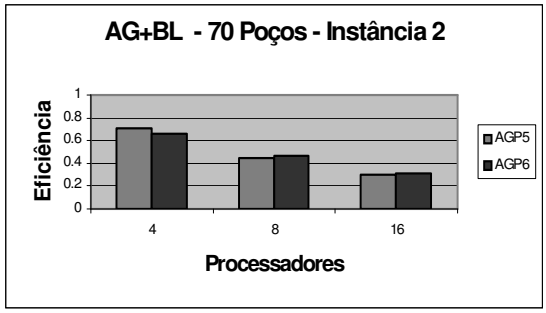

Figura 9.94: Comparativo de Eficiência entre versões Paralelas do AG + BL

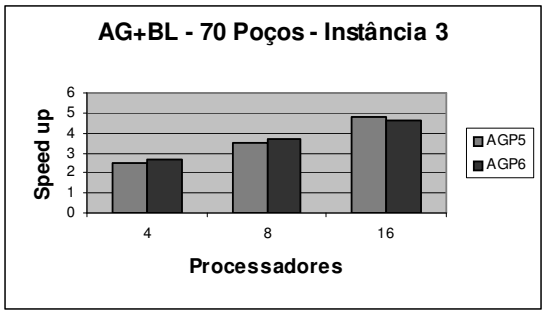

Figura 9.95: Comparativo de *Speed up* entre versões Paralelas do AG + BL

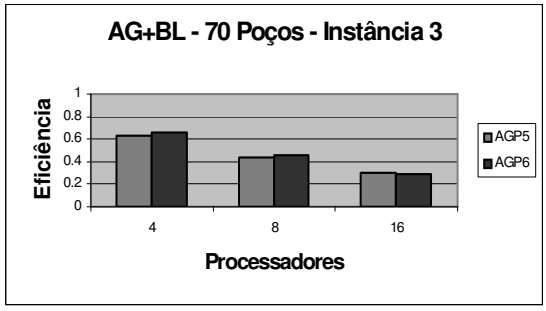

Figura 9.96: Comparativo de Eficiência entre versões Paralelas do AG + BL

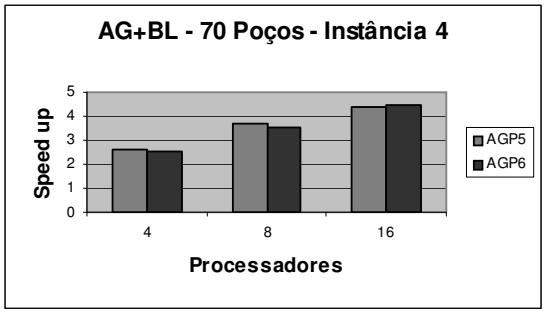

Figura 9.97: Comparativo de *Speed up* entre versões Paralelas do AG + BL

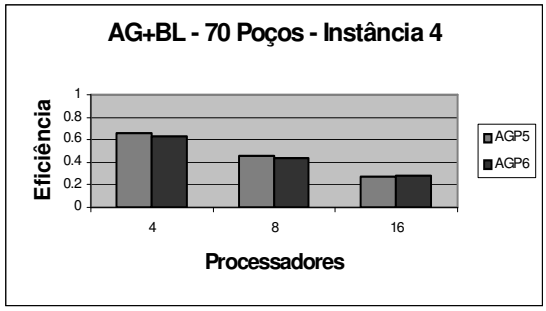

Figura 9.98: Comparativo de Eficiência entre versões Paralelas do AG + BL

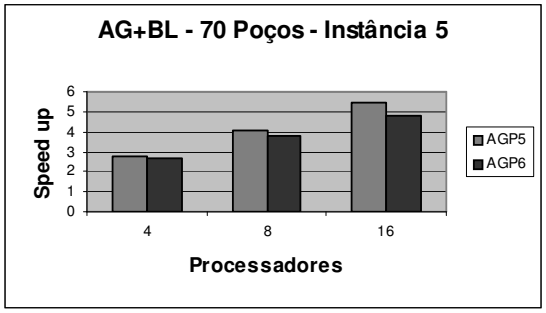

Figura 9.99: Comparativo de *Speed up* entre versões Paralelas do AG + BL

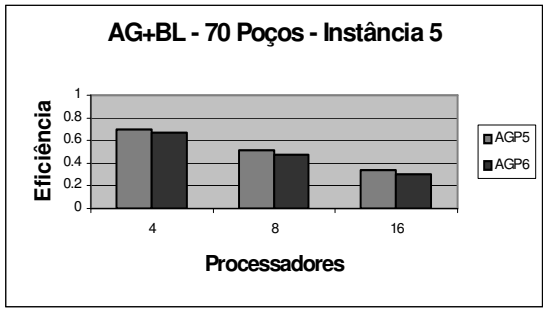

Figura 9.100: Comparativo de Eficiência entre versões Paralelas do AG + BL

Speed up

Eficiência

100 Poços

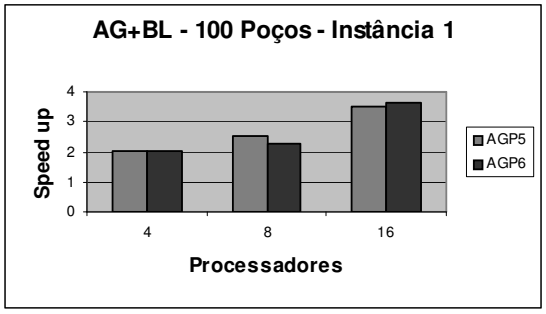

Figura 9.101: Comparativo de *Speed up* entre versões Paralelas do AG + BL

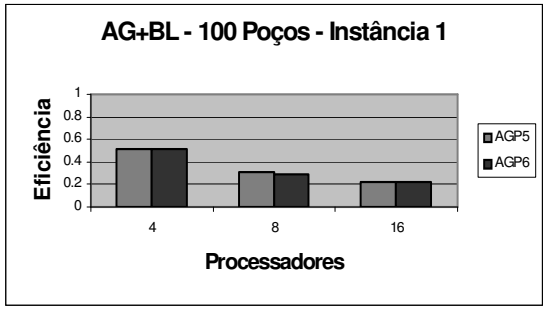

Figura 9.102: Comparativo de Eficiência entre versões Paralelas do AG + BL

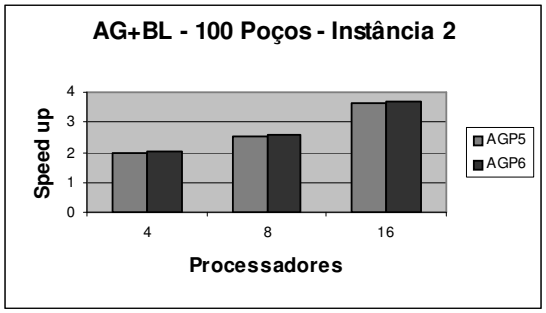

Figura 9.103: Comparativo de *Speed up* entre versões Paralelas do AG + BL

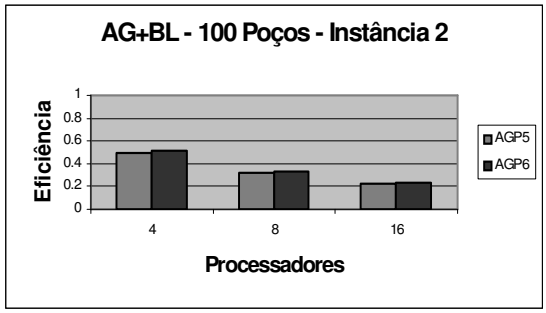

Figura 9.104: Comparativo de Eficiência entre versões Paralelas do AG + BL

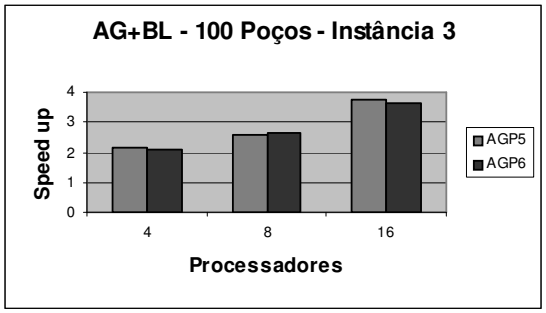

Figura 9.105: Comparativo de *Speed up* entre versões Paralelas do AG + BL

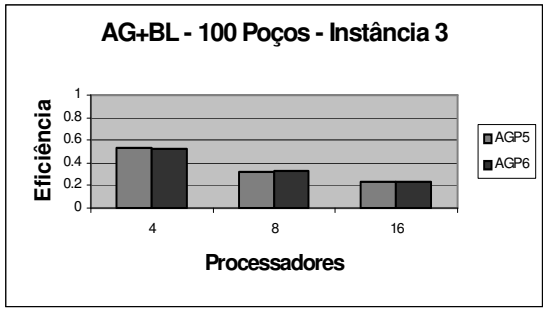

Figura 9.106: Comparativo de Eficiência entre versões Paralelas do AG + BL

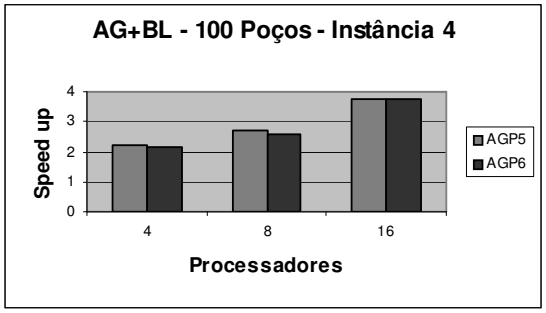

Figura 9.107: Comparativo de *Speed up* entre versões Paralelas do AG + BL

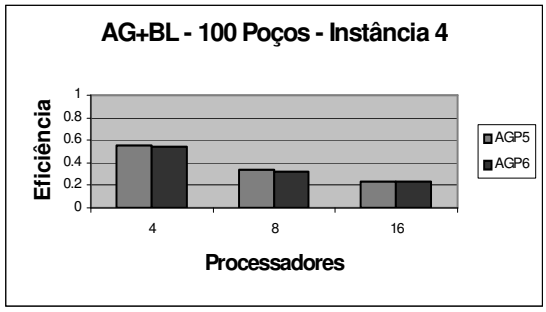

Figura 9.108: Comparativo de Eficiência entre versões Paralelas do AG + BL

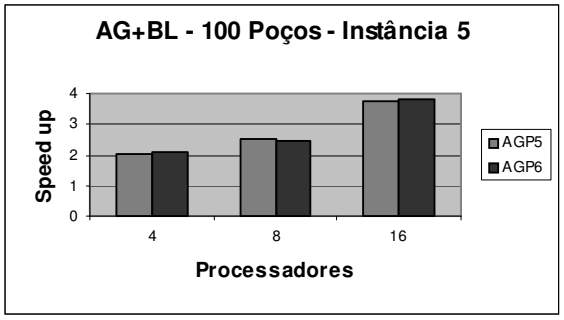

Figura 9.109: Comparativo de *Speed up* entre versões Paralelas do AG + BL

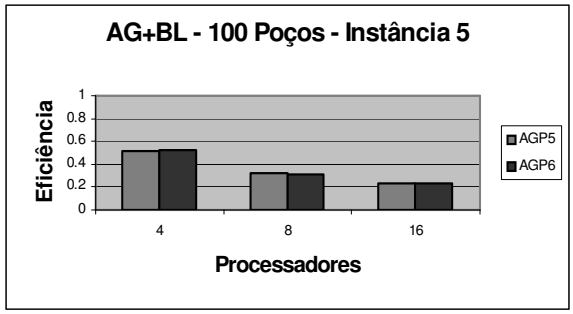

Figura 9.110: Comparativo de Eficiência entre versões Paralelas do AG + BL

Speed up **Eficiência**

120 Poços

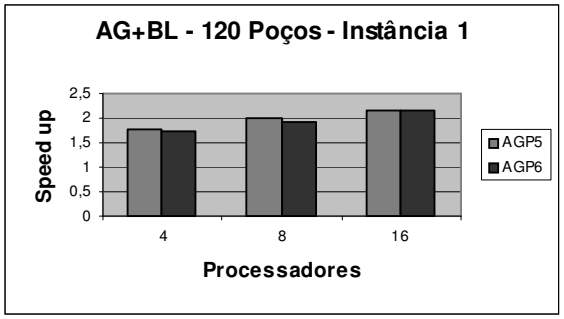

Figura 9.111: Comparativo de *Speed up* entre versões Paralelas do AG + BL

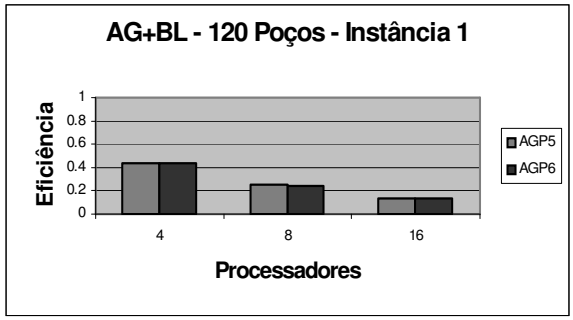

Figura 9.112: Comparativo de Eficiência entre versões Paralelas do AG + BL

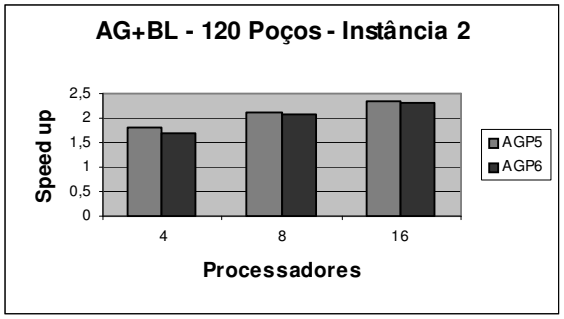

Figura 9.113: Comparativo de *Speed up* entre versões Paralelas do AG + BL

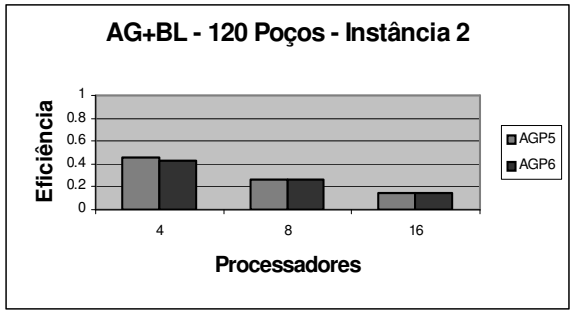

Figura 9.114: Comparativo de Eficiência entre versões Paralelas do AG + BL

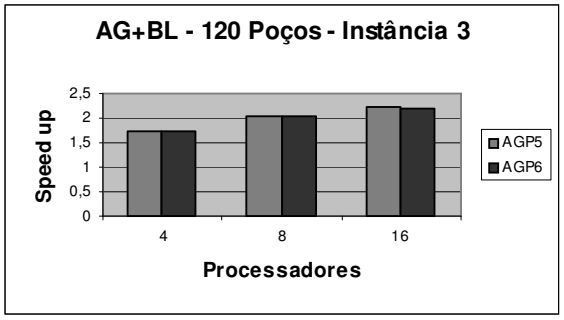

Figura 9.115: Comparativo de *Speed up* entre versões Paralelas do AG + BL

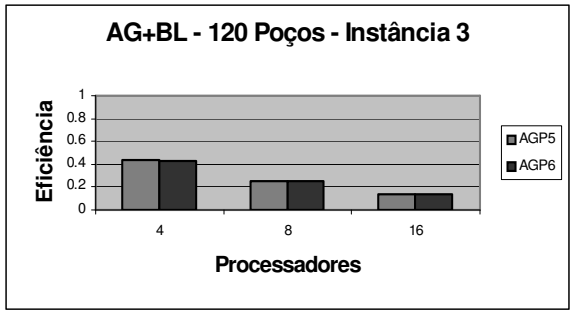

Figura 9.116: Comparativo de Eficiência entre versões Paralelas do AG + BL

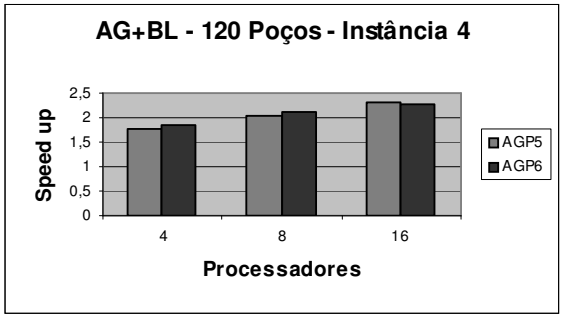

Figura 9.117 : Comparativo de *Speed up* entre versões Paralelas do AGB + BL

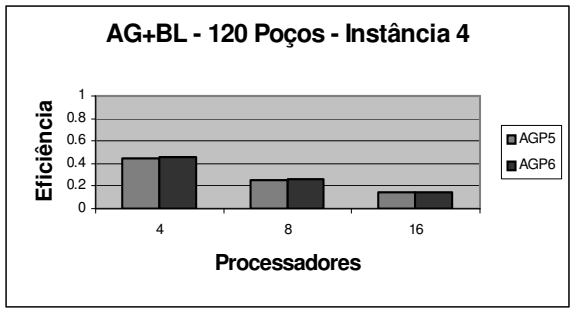

Figura 9.118: Comparativo de Eficiência entre versões Paralelas do AGB + BL

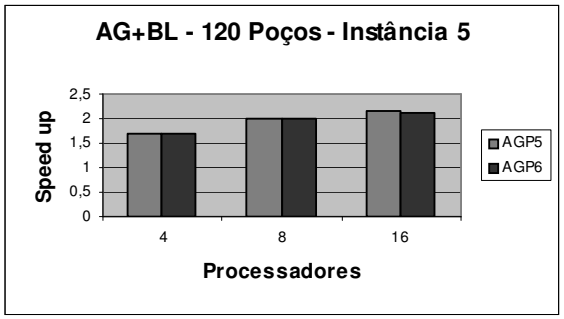

Figura 9.119: Comparativo de *Speed up* entre versões Paralelas do AG + BL

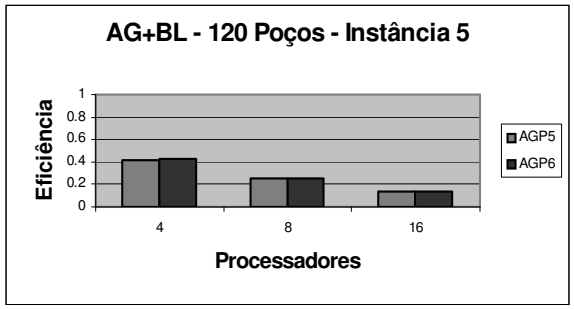

Figura 9.120: Comparativo de Eficiência entre versões Paralelas do AG + BL

9.4 Gráficos *Speed up* e *Eficiência* dos algoritmos AGP7 e AGP8.

As figuras à esquerda representam o *speed up*, e à direita, os gráficos de *eficiência* alcançados para cada um dos problemas executados em 4 e 8 processadores e que foram obtidos através dos algoritmos AGP7 e AGP8.

A seguir apresentamos gráficos comparativos de *speed up* e *eficiência* em detalhes, obtidos por esta versão.

Speed up **Eficiência**

50 Poços

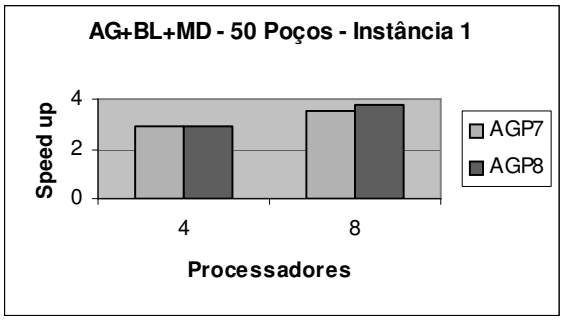

Figura 9.121: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

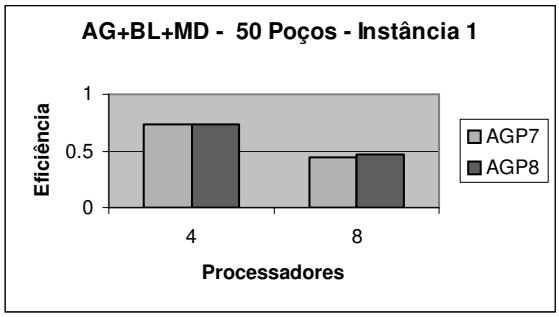

Figura 9.122: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

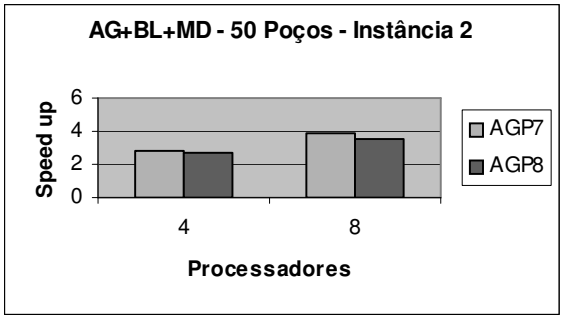

Figura 9.123: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

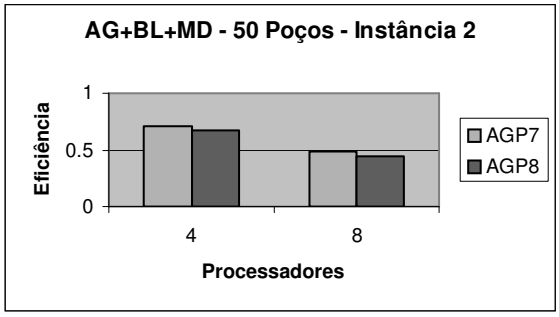

Figura 9.124: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

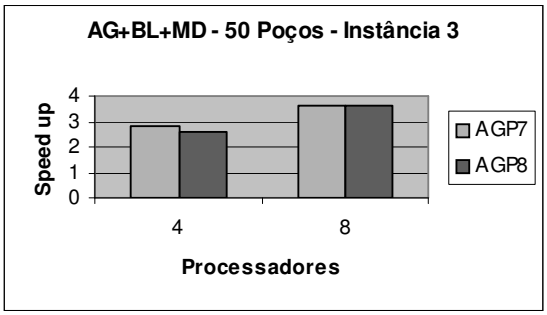

Figura 9.125: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

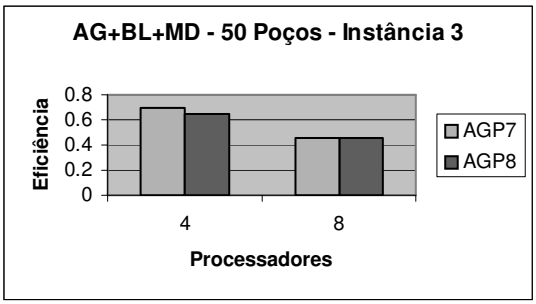

Figura 9.126: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

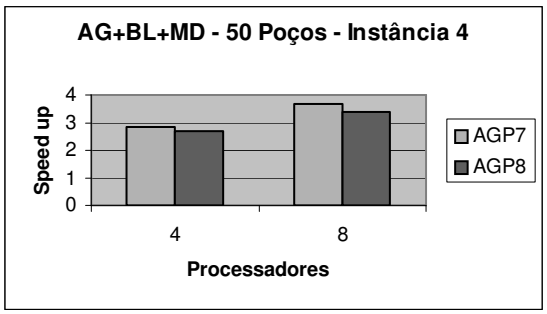

Figura 9.127: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

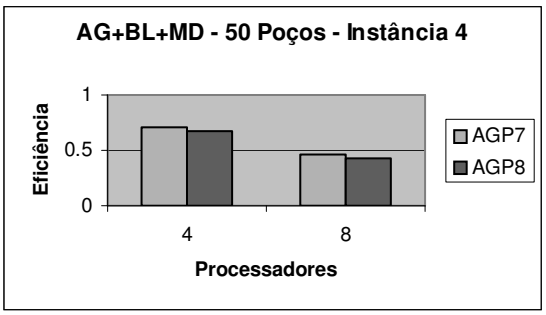

Figura 9.128: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

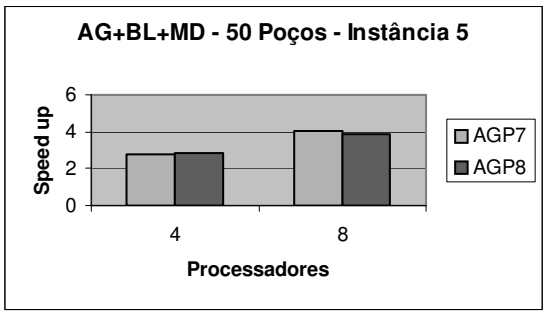

Figura 9.129: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

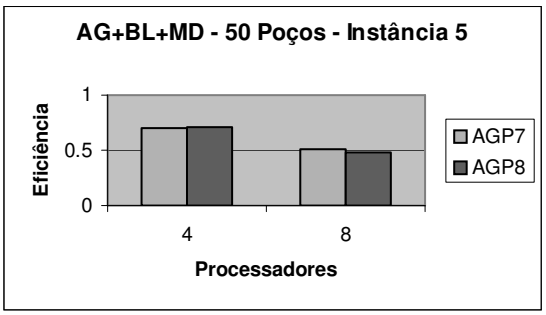

Figura 9.130: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

Speed up ***Eficiência***

70 Poços

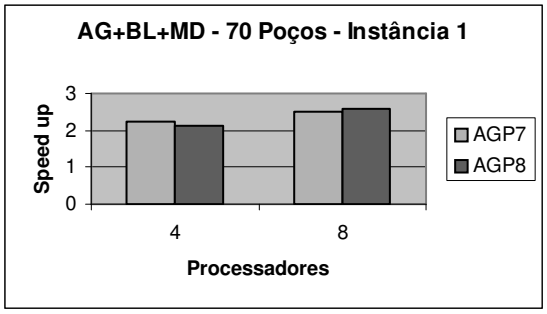

Figura 9.131: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

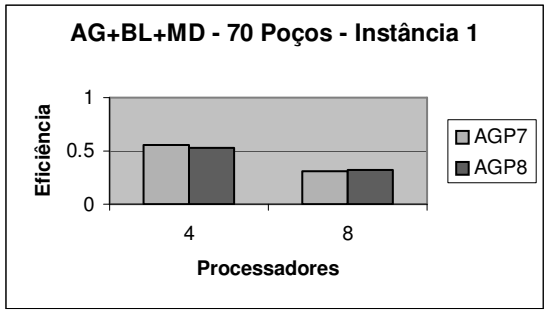

Figura 9.132: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

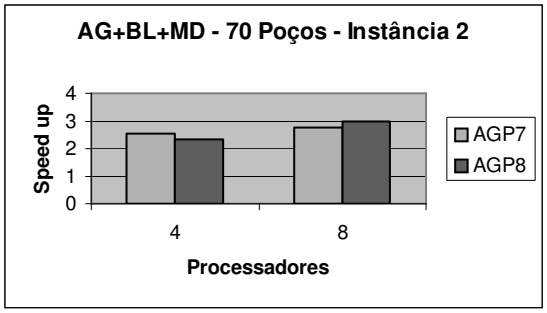

Figura 9.133: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

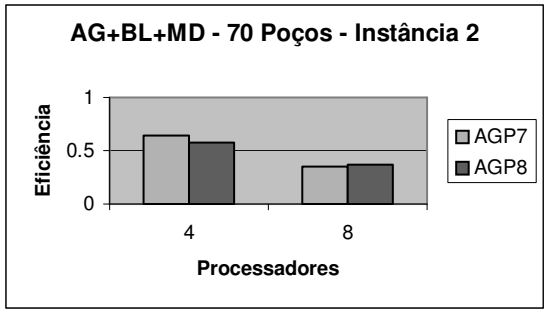

Figura 9.134: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

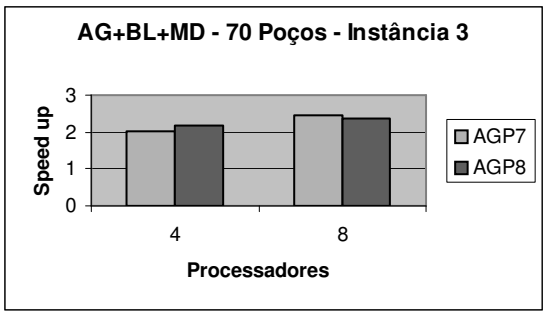

Figura 9.135: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

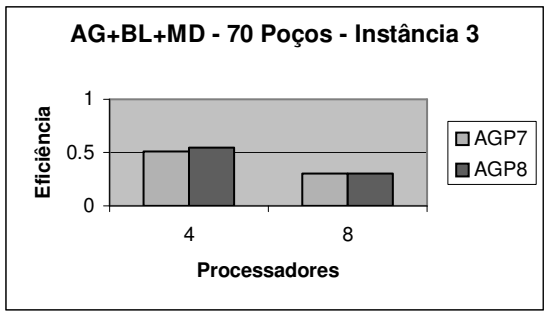

Figura 9.136: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

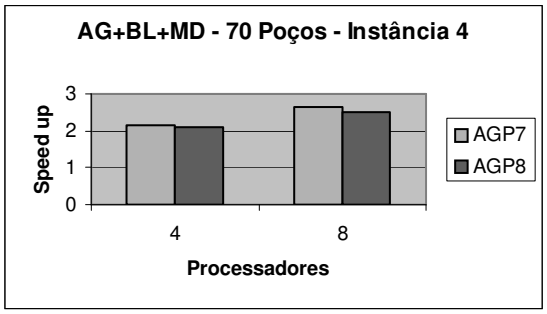

Figura 9.137: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

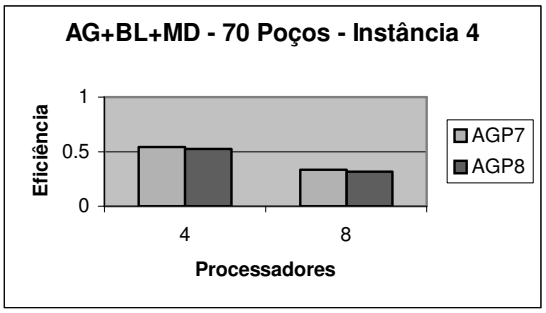

Figura 9.138: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

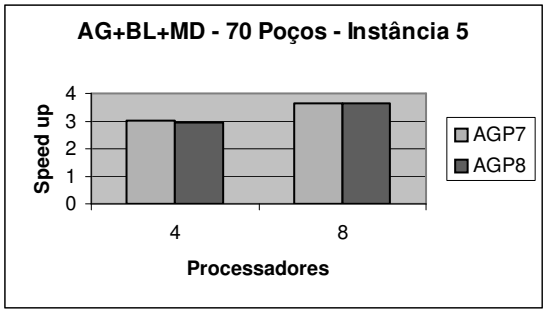

Figura 9.139: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

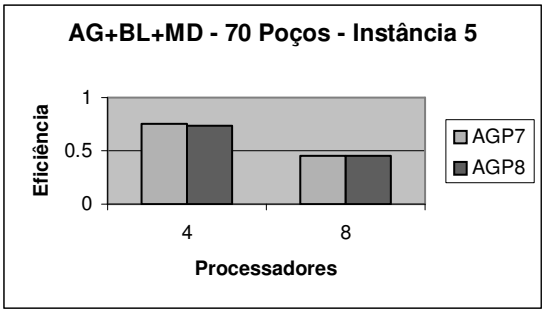

Figura 9.140: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

Speed up **Eficiência**

100 Poços

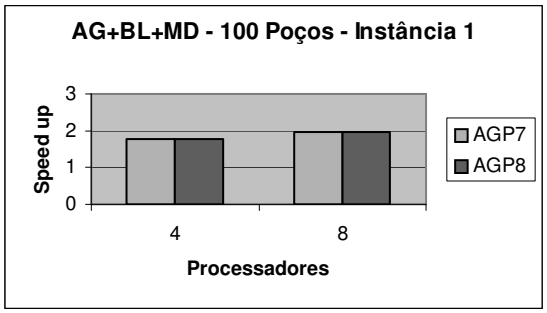

Figura 9.141: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

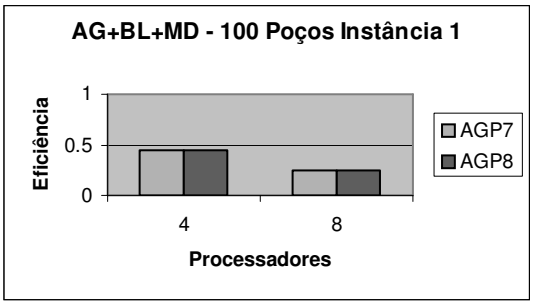

Figura 9.142: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

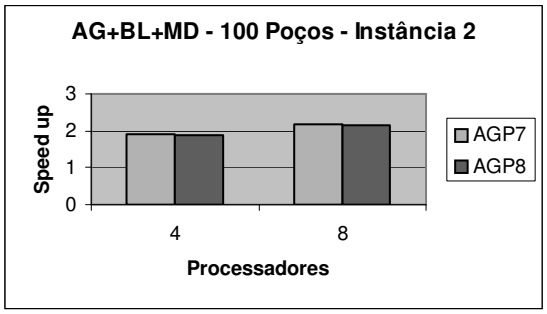

Figura 9.143: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

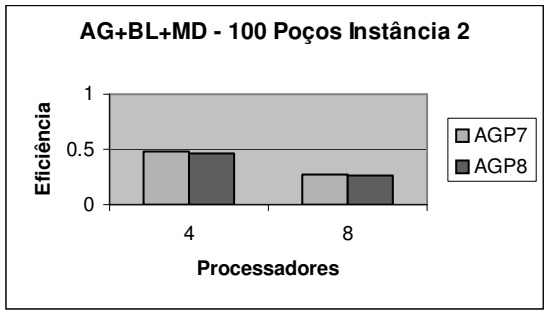

Figura 9.144: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

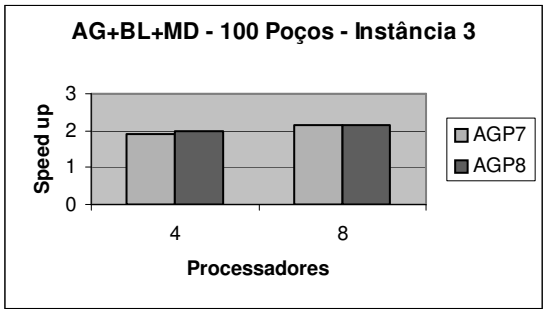

Figura 9.145: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

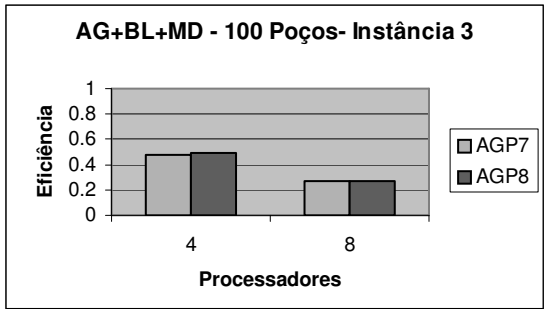

Figura 9.146: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

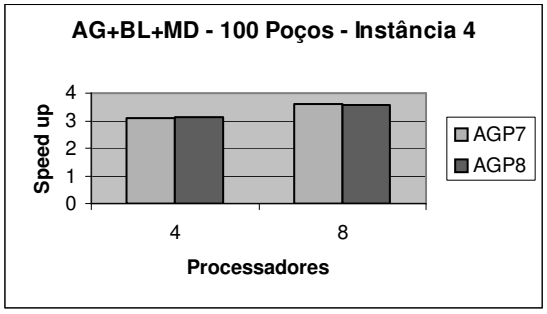

Figura 9.147: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

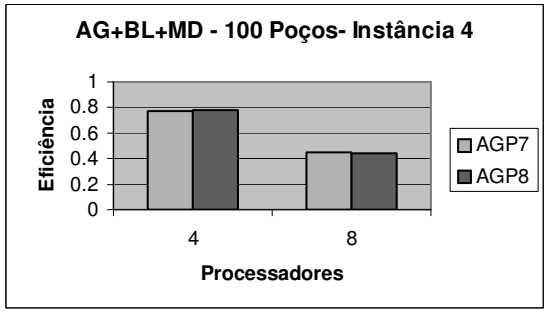

Figura 9.148: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

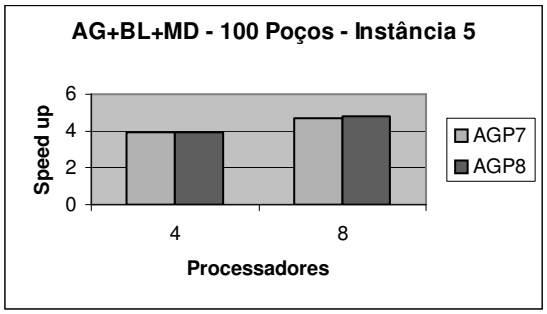

Figura 9.149: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

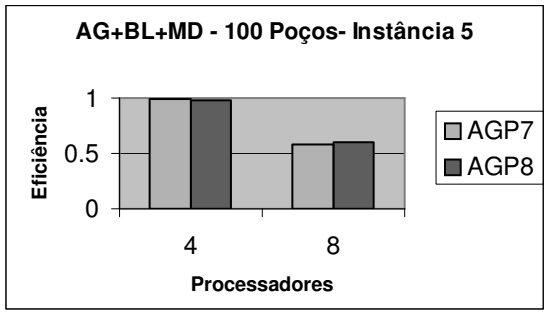

Figura 9.150: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

Speed up **Eficiência**

120 Poços

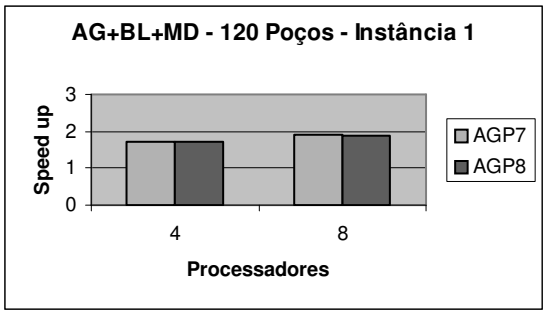

Figura 9.151: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

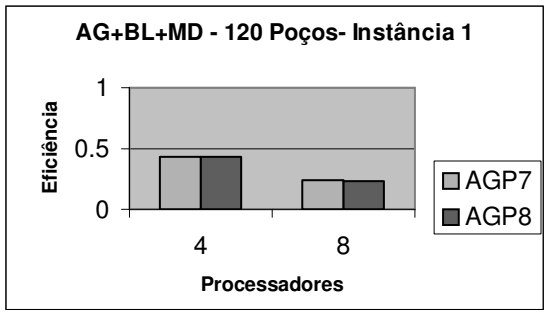

Figura 9.152: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

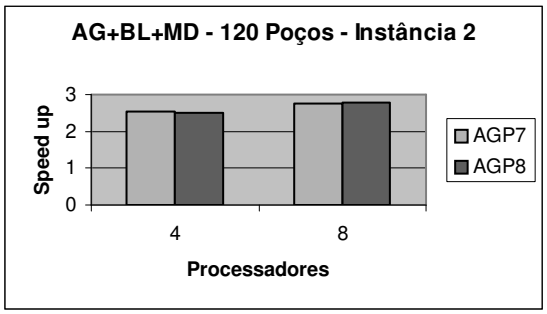

Figura 9.153: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

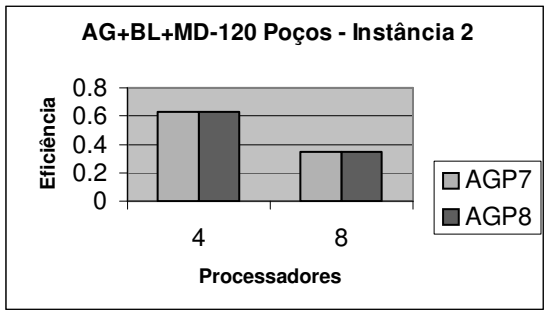

Figura 9.154: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

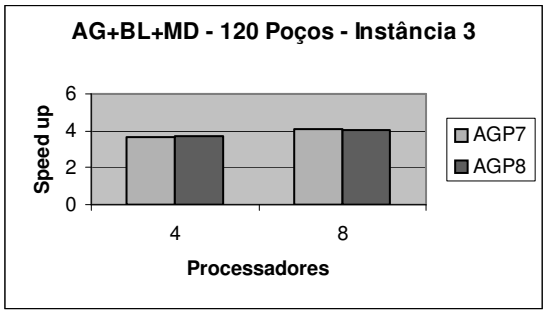

Figura 9.155: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

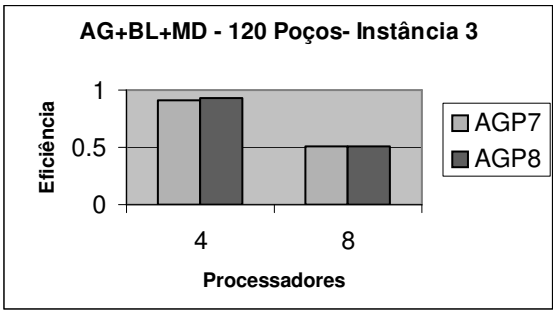

Figura 9.156: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

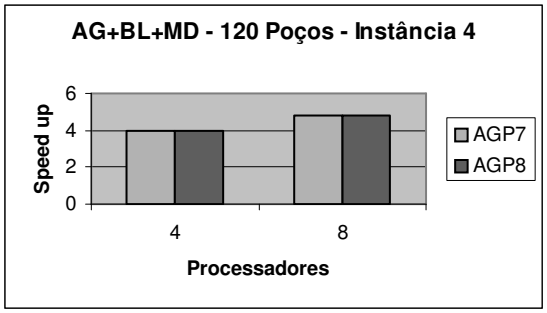

Figura 9.157: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

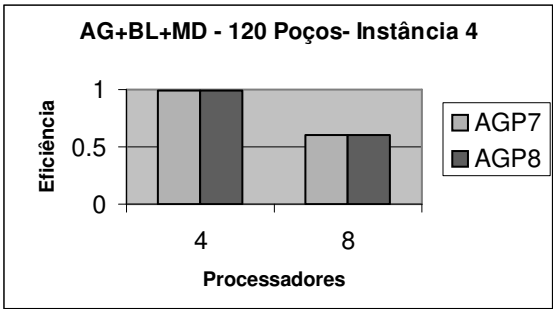

Figura 9.158: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD

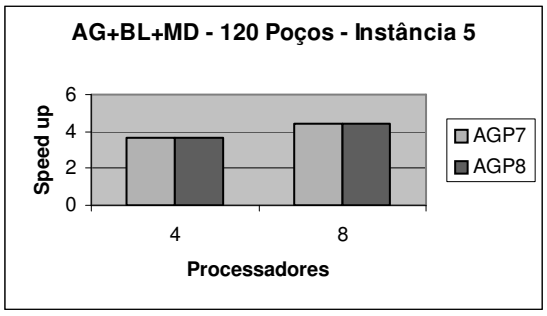

Figura 9.159: Comparativo de *Speed up* entre versões Paralelas do AG+BL+MD

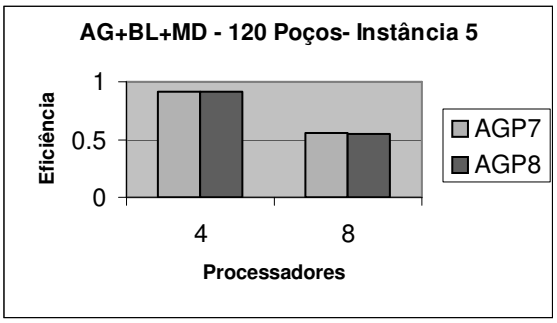

Figura 9.160: Comparativo de Eficiência entre versões Paralelas do AG+BL+MD