



# **Híbridização de Algoritmos Genéticos e Procura Tabu para o problema de *Job-Shop Scheduling***

**Aurora da Conceição João Cameirão**

**Outubro, 2008**

Orientação: Dalila B.M.M. Fontes

Co-Orientação: Carlos Soares

Tese de Mestrado em Análise de Dados e  
Sistemas de Apoio à Decisão

## Nota Biográfica

Aurora da Conceição João Cameirão nasceu em Duas Igrejas de Miranda do Douro, no dia 10 de Abril de 1975.

Concluiu a Licenciatura em Informática de Gestão, pela Universidade do Minho, em 1999.

Efectuou o seu estágio curricular na Blaupunkt Auto-Rádio Portugal, Grupo Bosch, onde tem vindo a desenvolver a sua actividade até hoje. Durante a sua permanência na Blaupunkt tem trabalhado em áreas diversas como sejam a análise e desenvolvimento de sistemas de informação, implementação de projectos de *datawarehousing*, administração de sistemas SAP, participação em projectos internacionais de implementação de sistemas SAP.

Em 2003, para complementar e diversificar os seus conhecimentos, decidiu iniciar os seus estudos de pós-graduação, em Análise de Dados e Sistemas de Apoio à Decisão.

## **Agradecimentos**

Em primeiro lugar quero agradecer aos meus orientadores, a Prof. Doutora Dalila B.M.M. Fontes e o Prof. Doutor Carlos Soares, pela colaboração, pelo tempo, pela dedicação e amizade que sempre me dedicaram.

Aos meus colegas de trabalho, pela amizade, pelos conselhos e pelo apoio.

Aos meus colegas do mestrado pelo incentivo que sempre me deram.

À minha família por me ter dado todas as condições para eu chegar até aqui.

Ao Cristóvão e à minha filha Filipa (que nasceu durante o decorrer deste trabalho) por me terem apoiado e compreendido.

## Resumo

Neste trabalho vamos abordar o problema do *job-shop* usando um algoritmo genético, dois algoritmos de procura tabu e dois algoritmos genéticos híbridos. O que pretendemos, é comprovar se usando um algoritmo híbrido podemos obter melhores resultados do que quando usamos os métodos que o compõem individualmente.

As experiências foram feitas usando 53 problemas frequentemente usados na literatura, os chamados *benchmark problems*.

Cada um dos cinco algoritmos propostos avaliará 100000 soluções, isto para que a comparação seja a mais justa possível. Com isto pretendemos avaliar a forma como cada um dos algoritmos explora o espaço de soluções e não a eficiência da sua implementação.

No decorrer deste estudo vamos analisar a evolução dos resultados de cada um dos algoritmos e efectuar a sua comparação.

## **Abstract**

In this paper we address the job shop problem using a genetic algorithm, two tabu search algorithms and two hybrid genetic algorithms. What we want to verify is if using hybrid genetic algorithms we can achieve better results than when we use the methods individually. The experiments were conducted using 53 problems often used in literature, the so-called benchmark problems.

Each one of the proposed five algorithms evaluates 100000 solutions, this in order that the comparison is as fair as possible. With this we intend to evaluate how each of the algorithms explores the space of solutions and not the efficiency of its implementation.

In this study we will examine the results evolution of each algorithm and make its comparison.

# Índice

|                                                                   |           |
|-------------------------------------------------------------------|-----------|
| <b>1. Introdução.....</b>                                         | <b>1</b>  |
| <b>1.1. Motivação .....</b>                                       | <b>1</b>  |
| <b>1.2. Objectivos.....</b>                                       | <b>2</b>  |
| <b>1.3. Estrutura da tese .....</b>                               | <b>3</b>  |
| <b>2. Planeamento de operações em job-shop scheduling.....</b>    | <b>5</b>  |
| <b>2.1. Descrição do processo job-shop .....</b>                  | <b>5</b>  |
| 2.1.1. ORR (Order Release/Review) .....                           | 8         |
| 2.1.2. Scheduling .....                                           | 8         |
| 2.1.2.1. Medidas de Avaliação.....                                | 10        |
| 2.1.3. Controlo na instalação funcional .....                     | 11        |
| <b>2.2. Abordagens existentes.....</b>                            | <b>12</b> |
| 2.2.1. Branch and Bound.....                                      | 14        |
| 2.2.2. Beam Search.....                                           | 14        |
| 2.2.3. Priority Dispatching Rules .....                           | 14        |
| 2.2.4. Shifting Bottleneck Procedure (SBP).....                   | 16        |
| 2.2.5. Local Search.....                                          | 16        |
| 2.2.6. Threshold Algorithms .....                                 | 17        |
| 2.2.7. Simulated Annealing (SA).....                              | 17        |
| 2.2.8. GRASP (Greedy Randomized Adaptative Search Procedure)..... | 18        |
| 2.2.9. Procura Tabu (PT) .....                                    | 18        |
| 2.2.10. Algoritmos Genéticos (AGs) .....                          | 19        |
| <b>3. Algoritmos Genéticos.....</b>                               | <b>20</b> |
| <b>3.1. Introdução .....</b>                                      | <b>20</b> |
| <b>3.2. Revisão da Literatura .....</b>                           | <b>21</b> |
| 3.2.1. Representação dos cromossomas.....                         | 22        |
| 3.2.2. Função de avaliação.....                                   | 23        |
| 3.2.3. Operadores de Selecção.....                                | 23        |

|             |                                                        |           |
|-------------|--------------------------------------------------------|-----------|
| 3.2.4.      | Operadores de Cruzamento .....                         | 24        |
| 3.2.5.      | Operadores de Mutação .....                            | 29        |
| 3.2.6.      | Parâmetros de controlo .....                           | 31        |
| <b>3.3.</b> | <b>Metodologia Usada.....</b>                          | <b>32</b> |
| 3.3.1.      | Concepção .....                                        | 32        |
| 3.3.2.      | Implementação .....                                    | 35        |
| <b>4.</b>   | <b><i>Procura Tabu.....</i></b>                        | <b>40</b> |
| <b>4.1.</b> | <b>Introdução .....</b>                                | <b>40</b> |
| <b>4.2.</b> | <b>Revisão da Literatura .....</b>                     | <b>41</b> |
| 4.2.1.      | Representação das soluções.....                        | 44        |
| 4.2.2.      | Obtenção da solução inicial .....                      | 44        |
| 4.2.3.      | Função de Avaliação .....                              | 44        |
| 4.2.4.      | Definição da Vizinhança .....                          | 45        |
| 4.2.5.      | Lista Tabu.....                                        | 45        |
| 4.2.6.      | Critério de Aspiração.....                             | 47        |
| 4.2.7.      | Parâmetros de Controlo .....                           | 47        |
| <b>4.3.</b> | <b>Metodologia Usada.....</b>                          | <b>48</b> |
| 4.3.1.      | Concepção .....                                        | 48        |
| 4.3.2.      | Implementação .....                                    | 49        |
| <b>5.</b>   | <b><i>Algoritmos Genéticos Híbridos .....</i></b>      | <b>52</b> |
| <b>5.1.</b> | <b>Introdução .....</b>                                | <b>52</b> |
| <b>5.2.</b> | <b>Revisão da literatura.....</b>                      | <b>52</b> |
| 5.2.1.      | Esquema Hierárquico .....                              | 53        |
| 5.2.2.      | Esquema Plano .....                                    | 54        |
| <b>5.3.</b> | <b>Metodologia usada .....</b>                         | <b>55</b> |
| 5.3.1.      | Concepção .....                                        | 55        |
| 5.3.2.      | Implementação .....                                    | 56        |
| <b>6.</b>   | <b><i>Resultados Experimentais .....</i></b>           | <b>60</b> |
| <b>6.1.</b> | <b>Descrição dos problemas usados.....</b>             | <b>60</b> |
| <b>6.2.</b> | <b>Descrição das experiências computacionais .....</b> | <b>61</b> |

|                                                                                               |            |
|-----------------------------------------------------------------------------------------------|------------|
| <b>6.3. Resultados Obtidos.....</b>                                                           | <b>63</b>  |
| 6.3.1. Análise do resultado dos algoritmos PT1 e PT2.....                                     | 66         |
| 6.3.2. Análise do resultado do algoritmo AG .....                                             | 72         |
| 6.3.3. Análise do resultado dos algoritmos AGH1 e AGH2.....                                   | 79         |
| 6.3.4. Análise comparativa do resultado obtido pelos algoritmos PT1, PT2, AG, AGH1 e AGH2 ... | 88         |
| <b>7. Conclusões .....</b>                                                                    | <b>99</b>  |
| 7.1. Objectivos.....                                                                          | 99         |
| 7.2. Resumo dos resultados.....                                                               | 100        |
| 7.3. Contribuições .....                                                                      | 101        |
| 7.4. Trabalho futuro.....                                                                     | 102        |
| <b>Bibliografia .....</b>                                                                     | <b>104</b> |
| <b>Anexos .....</b>                                                                           | <b>110</b> |
| Anexo 1 – Resultados iniciais dos parâmetros de controlo do algoritmo genético .....          | 111        |
| Anexo 2 – Codificação do algoritmo PT1 .....                                                  | 112        |
| Anexo 3 – Codificação do algoritmo PT2 .....                                                  | 115        |
| Anexo 4 – Codificação do algoritmo AG .....                                                   | 119        |
| Anexo 5 – Codificação do algoritmo AGH1 .....                                                 | 123        |
| Anexo 6 – Codificação do algoritmo AGH2 .....                                                 | 129        |
| Anexo 7 – Resultado dos tempos médios obtidos pelo algoritmo AGH1 .....                       | 137        |
| Anexo 8 - Resultados experimentais obtidos nas 10 execuções de cada algoritmo .....           | 139        |
| Anexo 9 – Resultados médios da evolução da procura .....                                      | 144        |

# 1. Introdução

## 1.1. Motivação

O *scheduling* de operações é um processo de produção muito importante em termos económicos, sendo no entanto uma tarefa muito difícil em termos computacionais.

O problema do *scheduling* de operações está entre os problemas combinatórios mais difíceis, pertencendo à classe de problemas NP-completos [Gary 1979]. Isto significa que não existem algoritmos que permitam resolver instâncias que não sejam pequenas num período de tempo aceitável.

Durante os anos 60 foi dada ênfase ao desenvolvimento de métodos de optimização que garantissem soluções óptimas através de algoritmos matemáticos sofisticados (e complicados). Estes métodos eram muito bons a resolver problemas pequenos, mas inadequados quando se tratava de problemas de média ou grande dimensão.

Dadas estas limitações, os métodos heurísticos acabaram por se tornar uma alternativa viável, pois sem considerar todo o espaço de soluções admissíveis, conseguem obter boas soluções [Jain e Meeran, 1999]. Surgem então métodos como os Algoritmos Genéticos (AG), a Procura Tabu (PT), o *Simulated Annealing* (SA), o *GRASP*, entre outros. Uma abordagem para resolver algumas das limitações destes métodos é a combinação de diferentes métodos. Estudos recentes têm mostrado que combinando algoritmos de procura local, como é o caso dos *Simulated Annealing*, a Procura Tabu ou o *GRASP*, com métodos de procura global, como os Algoritmos Genéticos, se obtêm melhores resultados do que usando um dos dois tipos de métodos individualmente.

Neste contexto, dada a importância do problema, decidimos efectuar um estudo comparativo do comportamento e desempenho de algoritmos híbridos relativamente ao uso individual dos métodos que os compõem.

## 1.2. Objectivos

Neste trabalho foi abordada uma versão simples do problema de *job-shop scheduling*, designada por  $n \times m$  [Wang e Zheng, 2002]. Nesta versão existem  $n$  *jobs* a processar em  $m$  máquinas. Assume-se que uma máquina só pode processar um *job* de cada vez, e a este processamento, de um *job* numa máquina, é designado por operação. Uma operação não pode ser interrompida após iniciar o seu processamento, e um *job* tem no máximo uma operação em cada máquina. As operações que compõem cada *job* têm de ser processadas por uma determinada ordem, que é conhecida a priori. A sequência de operações a executar em cada máquina é desconhecida, sendo que cada uma das alternativas possíveis e admissíveis constitui um programa (*schedule*). O problema consiste em encontrar um *schedule* que atribua as operações às máquinas, tendo em conta as restrições de precedência, de forma a minimizar o tempo total de processamento (*makespan*) [Gonçalves et al., 2005].

Dada a natureza combinatória do problema  $n \times m$ , o número de soluções possíveis, cresce exponencialmente com a dimensão do problema, e é dado por  $(n!)^m$ . Facilmente nos apercebemos que, à medida que aumenta o número de *jobs* e/ou o número de máquinas, se torna impraticável efectuar a enumeração exaustiva das soluções possíveis. Métodos de aproximação como os algoritmos genéticos ou a procura tabu têm vindo a ser usados em várias áreas de trabalho e investigação, nomeadamente para resolver este tipo de problemas. Apesar de não garantirem a obtenção de uma solução óptima, conduzem à obtenção de boas soluções, requerendo um tempo computacional significativamente menor comparativamente aos métodos que garantem uma solução óptima. Os algoritmos genéticos são um método muito flexível, sendo adaptáveis a muitos tipos de problemas diferentes. São capazes de explorar diferentes regiões do espaço de soluções admissíveis, tendo no entanto dificuldade em convergir para soluções óptimas locais. Por outro lado, métodos como a Procura Tabu têm dificuldades em explorar todo o espaço de soluções, mas especializam-se em encontrar ótimos locais.

Neste trabalho investigamos se a combinação destes métodos permite obter melhores resultados do que cada um dos métodos individualmente. São testadas duas variantes desta combinação, ambas do tipo Estafeta [Talbi, 2002]. A primeira é de Baixo Nível porque a Procura Tabu é usada como um operador do Algoritmo Genético, em

particular, um operador de mutação. A segunda é de Alto Nível, já que a Procura Tabu é efectuada apenas depois de terminada a execução do Algoritmo Genético.

### **1.3. Estrutura da tese**

No capítulo 2, abordamos a problemática do *job-shop scheduling*. Inicialmente começamos por fazer uma descrição do processo *job-shop* comparando-o com o processo do *flow-shop*. Ainda neste capítulo descrevemos o problema do *scheduling* de operações situando-o dentro do processo *job-shop*.

No capítulo 3, analisamos os algoritmos genéticos. Começamos por fazer uma descrição genérica do funcionamento dos AGs, passando depois para o seu enquadramento no problema em análise. Ainda neste capítulo, fazemos uma revisão da literatura para ilustrar o que tem sido feito sobre o assunto e explicar as decisões tomadas na implementação do AG. Para terminar o capítulo explicamos a sua implementação.

No capítulo 4, analisamos o método da procura tabu. Neste capítulo, à semelhança do anterior, começamos por efectuar uma descrição genérica do método da procura tabu e do seu funcionamento. De seguida fazemos uma revisão da literatura para analisar o que tem sido feito nesta área, e explicar as nossas decisões na implementação do método. Finalmente explicamos a implementação do método da procura tabu.

No capítulo 5, abordamos a temática da híbridização. Inicialmente fazemos uma descrição deste tipo de métodos, explicando o porquê do seu aparecimento. De seguida evidenciamos o que tem sido feito nesta área e a metodologia usada por nós no desenvolvimento dos algoritmos testados neste trabalho. Por fim, descrevemos com mais detalhe a implementação de cada um deles.

No capítulo 6, procedemos à análise dos resultados obtidos. Inicialmente, explicamos quais os problemas que serviram de base a este estudo experimental e para depois

descrever as experiências computacionais levadas a cabo. Por último analisamos os resultados observados.

No capítulo 7, são apresentadas as conclusões. Neste capítulo fazemos um resumo dos resultados obtidos, evidenciando as principais contribuições deste trabalho bem como algumas ideias para trabalho futuro.

## **2. Planeamento de operações em *job-shop scheduling***

### **2.1. Descrição do processo *job-shop***

Cada empresa é única na forma como faz o planeamento das suas actividades e actua. No entanto, de acordo com [Courtois et al., 1996], existem alguns critérios como sejam: as quantidades fabricadas e grau de competitividade (produção unitária, produção de pequenas séries, produção de médias séries e produção de grandes séries); organização dos fluxos de produção (produção contínua, produção descontínua, produção por projectos); o relacionamento com os clientes (venda a partir do stock, produção por encomenda, montagem por encomenda), que podem ser usados para enquadrar e classificar uma empresa. De acordo com estes critérios podemos caracterizar o modo de produção de uma empresa como *Flow-Shop* ou *Job-Shop*.

Uma empresa com um processo produtivo do tipo *flow-shop* caracteriza-se por usar um fluxo de produção contínuo (linha de produção), produzindo um ou poucos produtos diferentes, mas em grandes quantidades.

Neste tipo de produção, as máquinas estão dedicadas ao produto em fabrico, pelo que geralmente é pouco flexível quando se pretendem efectuar alterações.

Por outro lado, este processo é relativamente simples no que toca à programação e controlo das operações.

Geralmente, neste tipo de produção, podemos ver que muitas das tarefas são automatizadas, isto para poderem ter custos de produção baixos, manter um elevado nível de qualidade, ter um baixo número de produtos em curso de fabrico e proporcionar uma rápida movimentação dos produtos entre os postos de trabalho.

A maioria das empresas de pequena e média dimensão do tecido empresarial Português usa um processo produtivo do tipo “*job-shop*”, que se caracteriza por ter um processo de produção descontínuo, onde são processadas várias ordens de fabrico, podendo cada uma delas ter um *routing* diferente, ou seja, uma sequência específica e diferente de operações a realizar. Neste tipo de empresas produzem-se vários produtos diferentes,

mas em pequenas quantidades, pelo que a flexibilidade é uma característica imperativa neste tipo de processo. Aqui, as máquinas ou as instalações não estão atribuídas a um produto específico, podendo realizar um grande número de operações. Neste tipo de processo as duas decisões mais importantes são a definição da data de início de execução de uma ordem de produção<sup>i</sup>, e a sequência de processamento das suas operações. A disponibilidade dos materiais e dos recursos é também uma das preocupações a ter em conta.

De uma forma geral, podemos dizer que um processo produtivo do tipo *job-shop* opõe uma grande flexibilidade e organização complexa, a uma flexibilidade mais reduzida mas com uma organização mais simples como é o caso do *flow-shop*.

Na Figura 1 podemos ver um quadro comparativo dos processos produtivos do tipo *job-shop* e do tipo *flow-shop*.

|                                               | <b>job-shop</b>                                                                | <b>flow-shop</b>                                                                              |
|-----------------------------------------------|--------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| <b>fluxo dos produtos(routings )</b>          | Fluxos complexos, variados                                                     | Fluxo linear, único                                                                           |
| <b>bottlenecks</b>                            | variam com frequência                                                          | estacionários                                                                                 |
| <b>máquinas usadas</b>                        | flexíveis, não especializadas                                                  | especializadas (geralmente para um produto)                                                   |
| <b>dimensão dos lotes</b>                     | geralmente pequenos                                                            | geralmente grandes (produções em série)                                                       |
| <b>custos de set-up</b>                       | geralmente baixos                                                              | elevados devido à grande especialização                                                       |
| <b>especialização da mão-de-obra</b>          | elevada                                                                        | geralmente é baixa                                                                            |
| <b>custos em mão-de-obra</b>                  | elevados                                                                       | baixos                                                                                        |
| <b>stocks de matéria prima</b>                | geralmente baixos                                                              | muito variáveis                                                                               |
| <b>stocks de produtos em curso de fabrico</b> | elevados                                                                       | geralmente são baixos                                                                         |
| <b>stocks de produtos acabados</b>            | baixos                                                                         | geralmente são elevados                                                                       |
| <b>programação das operações</b>              | incerta, varia frequentemente                                                  | inflexível                                                                                    |
| <b>"problemas" para gestão de operações</b>   | previsões de procura, loading, bottlenecks                                     | melhoria da produtividade, controle do processo                                               |
| <b>prazos</b>                                 | geralmente longos                                                              | geralmente curtos                                                                             |
| <b>flexibilidade das linhas de produção</b>   | linhas de produção flexíveis                                                   | linhas de produção rígidas                                                                    |
| <b>exemplos</b>                               | restaurantes, lojas de reparação, tipografias, estabelecimentos de ensino, etc | indústria petroquímica, cimenteiras, indústria automóvel, ferramentas, electrodomésticos, etc |

Figura 1. Comparação de processos *Job-shop* / *Flow-shop*, adaptado de [Fontes e Moreira, 2004-a].

---

<sup>i</sup> A ordem de produção resulta de uma encomenda a ser satisfeita e é composta por um conjunto de operações a serem processadas

Geralmente num *job-shop* o modelo de organização mais comum é em secções homogéneas, também designadas por centros de trabalho, que agrupam as máquinas e restantes equipamentos pela semelhança que existe entre eles (ex: montagem manual, fresagem, pintura, controlo final, etc.), ou de acordo com critérios de qualidade ou capacidade. Este tipo de organização tem vantagens, pois permite que o pessoal que trabalha numa secção e é especialista numa determinada máquina, possa usar qualquer outra da mesma secção. Pelo facto das secções serem independentes do produto em fabrico, elas são muito flexíveis, podendo ser adaptadas para produzir novos produtos, sem interromper ou interferir no fluxo normal das outras secções.

No entanto, não há só vantagens neste modelo de organização, a flexibilidade paga-se com a complexidade, os fluxos são muito complexos, originando pontos de inversão e acumulação de produtos em curso de fabrico (stocks de produtos em curso de fabrico elevados). Os *job-shops* são caracterizados pela grande variedade de produtos, com *routings* e tempos de processamento variáveis [Land e Gaalman, 1997].

Um dos grandes desafios deste tipo de processo é a programação (*schedule*) e controlo na instalação funcional [Chase e Aquiliano, 1989]. Por programação entende-se o planeamento que é efectuado, a curto prazo, para escalonar operações usando recursos e alocando as instalações. O controlo das operações é efectuado para monitorizar o progresso das ordens de fabrico, efectuando alguns ajustes de forma a garantir o cumprimento do programa director.

De acordo com [Chase e Aquiliano, 1989], um sistema de programação e controlo tem de ser eficiente na:

“Atribuição de ordens, equipamento e pessoal aos centros de trabalho ou a outras localizações específicas. (Essencialmente, trata-se do planeamento de capacidades a curto prazo.)

Determinação da sequência de realização das ordens; isto é, estabelecimento de prioridades para as tarefas.

Início da execução do trabalho programado; isto é normalmente designado por lançamento de ordens.

Controlo de fabricação (ou controlo das actividades de produção) envolvendo:

Revisão do estado e controlo do progresso das ordens à medida que são executadas.

Aceleração das ordens atrasadas e críticas.

Revisão do programa à luz das alterações no estado das ordens. “

### **2.1.1. ORR (Order Release/Review)**

Num sistema produtivo do tipo *Job-shop*, geralmente, trabalha-se para satisfazer encomendas, sendo a produção efectuada em lotes. Cada uma destas encomendas é considerada um *job*, sendo que a primeira questão (fase de planeamento) a colocar deverá ser a data de início de produção da encomenda, ou seja, determinação da data de lançamento (*release date*) do *job* no *shop-floor*.

Existem alguns factores a ter em conta antes de lançar uma ordem de produção no *shop-floor*: qual é a data de entrega acordada com o cliente (*due date*), quanto tempo é que o *job* necessita para processamento, qual a prioridade do *job*, existência dos recursos necessários, carga do sistema, etc. A forma mais simples para calcular a *release date*, seria determinar o tempo de processamento necessário para cada actividade do *job* e deduzi-lo à *due date*, com isto poderiam evitar-se os stocks de produtos acabados (caso comesse cedo demais), ou atraso na entrega (caso comesse demasiado tarde). Seria simples, se houvessem poucas encomendas, ou se o fluxo (*routing*) fosse único, no entanto tratando-se de um sistema complexo, onde cada ordem de produção (*job*) pode ter várias operações, com tempos de processamento diferentes e *routings* diferentes, este processo torna-se mais complexo.

Antes de poder lançar uma nova ordem, é necessário controlar o nível de *input* e *output* do *shop-floor*, isto para saber se ele tem capacidade de receber uma nova ordem. Os mecanismos de lançamento de ordens (ORR) devem poder controlar as entradas e saídas no *shop-floor*, para determinar o melhor momento para lançar uma nova ordem. Estes mecanismos devem conseguir determinar quais as ordens que devem ser lançadas, quando devem ser lançadas e quais as condições a verificar para o seu lançamento.

### **2.1.2. Scheduling**

Por programação (*scheduling*) entende-se o planeamento que é feito a curto prazo para escalonar operações, usando recursos e alocando as instalações.

Tipicamente, este problema é descrito como um problema do tipo  $n$  jobs x  $m$  máquinas, sendo que cada um destes  $n$  jobs é composto por um conjunto de operações a serem

processadas em  $m$  máquinas. Cada uma destas operações usa uma das  $m$  máquinas durante um período fixo de tempo. Cada máquina pode processar no máximo uma operação de cada vez, e após iniciar o seu processamento, este não pode ser interrompido. As operações que compõem um *job* têm de ser processadas por uma determinada ordem (*routing*).

O problema consiste em encontrar um programa (*schedule*) que atribua as operações às máquinas, tendo em conta as restrições de precedência, e que permita minimizar o *makespan*, ou seja o tempo total para executar todas as operações [Gonçalves e tal., 2004].

[Gonçalves e tal., 2004] usam o *makespan* para avaliar os programas, no entanto podem ser usadas outras funções de avaliação, tal como veremos a seguir.

Para ilustrar esta situação podemos ver o exemplo em baixo, onde estão representados 3 *jobs*, com *routings* e tempos de processamento diferentes.

|     |   | Tempos de processamento |           |
|-----|---|-------------------------|-----------|
|     |   | Máquina 1               | Máquina 2 |
| Job | 1 | 1                       | 3         |
|     | 2 | 2                       | 1         |
|     | 3 | 2                       | 2         |

a)

|     |   | Routings  |           |
|-----|---|-----------|-----------|
|     |   | Máquina 1 | Máquina 2 |
| Job | 1 | 1         | 2         |
|     | 2 | 2         | 1         |
|     | 3 | 1         | 2         |

b)

Figura 2. Representação de 3 jobs, com as respectivas operações, tempos de processamento, máquinas e routings.

A tabela a) da Figura 2, contém o tempo de processamento requerido por cada *job* em cada máquina. A tabela b) contém as operações que compõem cada *job*, a máquina onde devem ser processadas, bem como a sequência de processamento (os *routings*). Estes dados vão ser posteriormente usados para construir o melhor *schedule*.

Se quiséssemos que estes *jobs* fossem processados pela ordem 1, 2 e depois 3 iríamos obter o seguinte *schedule*:

| Máquina 1 | J1    |    |    |    |    |    | J2 | J3 | J3 |    |    |          |  |  |
|-----------|-------|----|----|----|----|----|----|----|----|----|----|----------|--|--|
| Máquina 2 |       | J1 | J1 | J1 | J2 | J2 |    |    |    | J3 | J3 |          |  |  |
|           | $t_0$ |    |    |    |    |    |    |    |    |    |    | $t_{11}$ |  |  |

Tempo →

Figura 3. Schedule quando existe a restrição de executar os jobs pela ordem 1, 2 e 3.

Como podemos observar na Figura 3, o processamento destes 3 *jobs* estaria concluído ao fim de 11 períodos de tempo, ou seja obteríamos um *makespan* de 11. Este é o cenário mais simples e fácil de construir, no entanto existem muitos outros, geralmente mais eficientes.

Para este exemplo temos 36 programas diferentes,  $(n!)^m$  alternativas diferentes (onde  $n$  é o número de *jobs* e  $m$  é o número de máquinas).

Na Figura 4, podemos ver um programa alternativo.

|           |    |    |    |    |    |    |    |  |  |  |  |  |  |
|-----------|----|----|----|----|----|----|----|--|--|--|--|--|--|
| Máquina 1 | J3 | J3 | J2 | J1 |    |    |    |  |  |  |  |  |  |
| Máquina 2 | J2 | J2 | J3 | J3 | J1 | J1 | J1 |  |  |  |  |  |  |

Figura 4. Solução possível para o problema de *scheduling*, do exemplo dado.

Como se pode ver pela Figura 4, com este programa, consegue-se reduzir o tempo de processamento total (*makespan*), de 11 para 7 unidades de tempo, e sem haver interrupção no funcionamento das máquinas.

À medida que aumenta o número de ordens de produção e de máquinas, tornasse impraticável testar todos os cenários possíveis para determinar aquele que melhor se adequa, pelo que ao longo do tempo têm vindo a ser desenvolvidas metodologias para auxiliar a resolução deste problema.

#### 2.1.2.1. Medidas de Avaliação

A medida de avaliação mais frequentemente usada para avaliar os programas, é o *makespan* ([Park et al., 2003], [Wang e Zheng, 2002], [Soares e tal., 1995], etc), ou seja, o tempo total necessário para executar todas as operações. Existem ainda autores que combinam esta medida de avaliação com outras, como é o caso de [Tsujimura e tal., 2001], que apresenta entre outras, uma medida de avaliação dada pela combinação de três critérios: minimização dos tempos de paragem, minimização dos tempos de espera das máquinas e minimização do *makespan*.

De acordo com [Jones e Rabelo, 1998], nas medidas de avaliação mais frequentemente usadas podemos encontrar: a minimização dos atrasos totais; a minimização do número de *jobs* atrasados; a maximização da utilização do sistema/recursos; a minimização dos *stocks* de produtos em curso; o uso equilibrado de recursos e a maximização do rácio de produção.

Apesar do *makespan* ser a medida mais usada, não existe uma medida de avaliação que seja melhor para todos os problemas, esta depende muito do problema em estudo. Por exemplo, se o atraso nas entregas ao cliente é muito importante ou se existem problemas em termos de espaço de armazenamento, a medida de avaliação deve ter em conta estes factores.

### 2.1.3. Controlo na instalação funcional

O controlo na instalação funcional é efectuado para monitorizar a evolução das ordens de produção, ajustando o sistema quando necessário. Este ajuste é feito quer ao nível dos recursos necessários, quer ao nível da alteração de prioridades de processamento de determinadas tarefas, como de identificação e resolução de pontos de estrangulamento (*bottlenecks*). É muito importante controlar o nível de entradas e saídas do sistema, para evitar pontos de estrangulamento, o nível do trabalho planeado não deve exceder o nível do trabalho concluído.

Na Figura 5 podemos ter uma ideia geral do tipo de tarefas a desenvolver no controlo de fabrico.

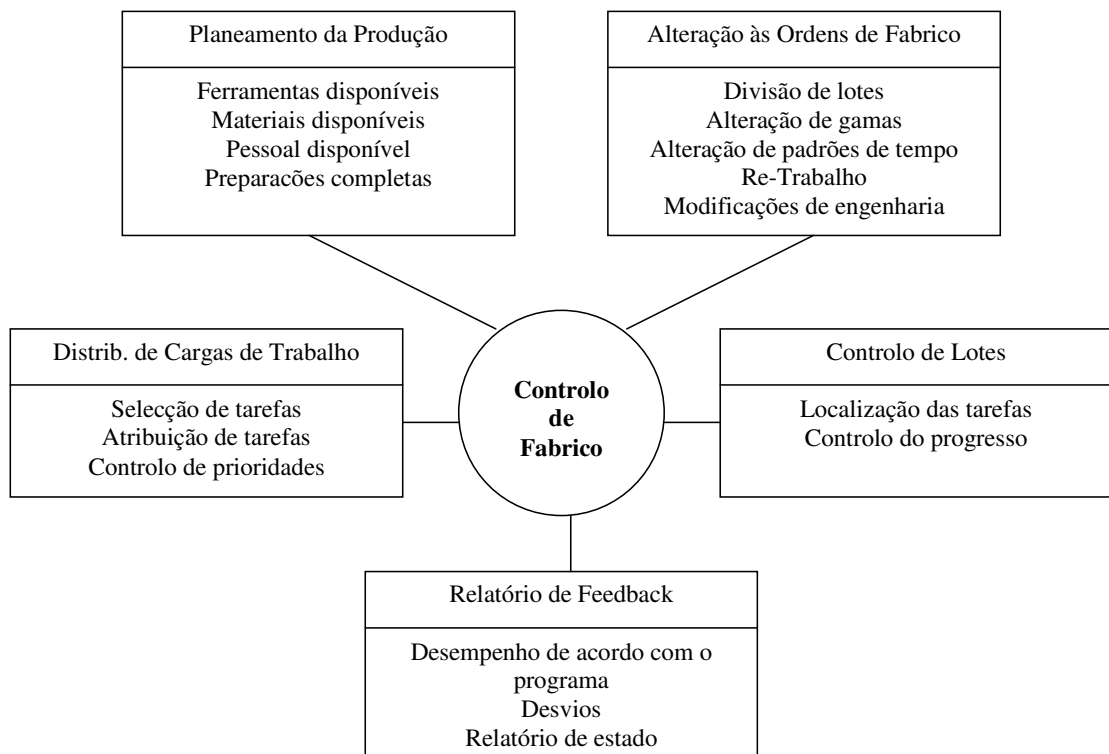


Figura 5. Tarefas a desenvolver no controlo de Fabrico, adaptado de [Chase e Aquilano, 1989].

De acordo com [Chase e Aquilano, 1989], para o bom funcionamento da instalação funcional temos de controlar 5 variáveis muito importantes. Por um lado actuar ao nível do planeamento de produção de forma a garantir a disponibilidade dos recursos, em termos de pessoas, máquinas, matérias e processos.

É necessário controlar eventuais alterações às ordens de fabrico, saber se houve alterações ao produto para evitar re-trabalhos, entre outros.

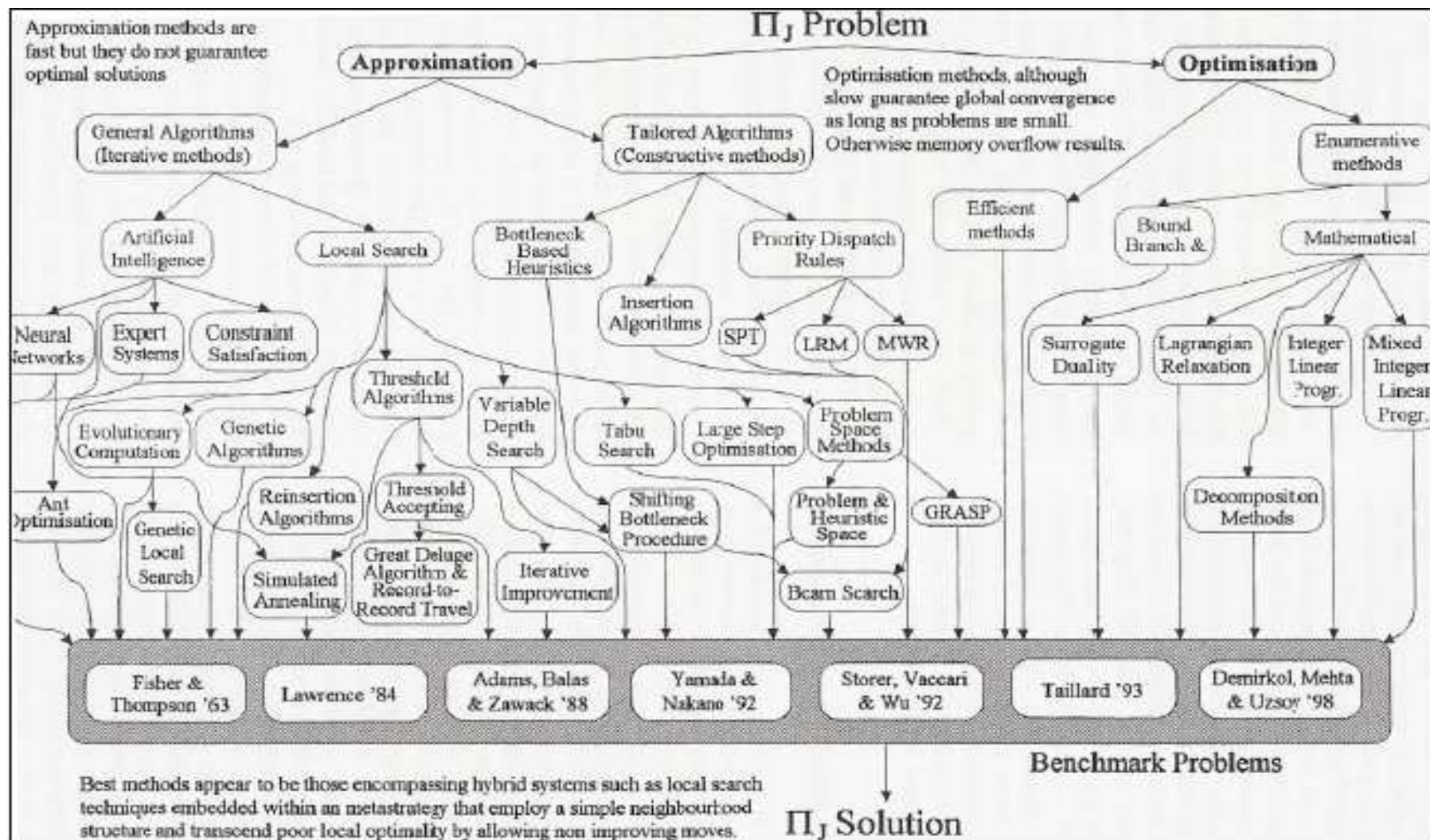
A distribuição da carga de trabalho é outra variável muito importante a ter em conta. É necessário efectuar uma selecção e alocação eficaz das tarefas e definir prioridades para tentar evitar os pontos de estrangulamento.

O controlo dos lotes é outra variável muito importante, é necessário ter um controlo permanente do progresso das ordens de fabrico, saber a todo o momento qual a tarefa que está a ser processada e em que máquina.

Por fim, e não menos importante, aparece o relatório de feedback. É importante que os decisores tenham informação adequada sobre o estado das ordens de fabrico, eventuais desvios e desempenho do *shop-floor* para que possam agir de acordo com a situação, de forma sustentada.

## **2.2. Abordagens existentes**

Ao longo dos tempos, muitos métodos foram surgindo para o tratamento do problema de *scheduling* de operações, a Figura 6 resume as mais importantes.



1      Figura 6. Métodos usados para abordar o problema de *scheduling* (retirado de [Jain e Meeran, 1999]).

Os métodos de optimização garantem soluções óptimas, no entanto, só são aplicáveis a problemas pequenos dada a necessidade de elevados tempos de processamento. Devido às suas limitações, os métodos aproximados tornaram-se uma alternativa viável, pois apesar de nem sempre conseguirem encontrar as soluções óptimas, são mais eficientes em termos de tempo computacional.

Seguidamente descrevem-se de forma resumida alguns dos métodos que foram surgindo.

### **2.2.1. Branch and Bound**

Este método vai construindo uma árvore com soluções admissíveis. Na tentativa de não representar todo o espaço de procura, usa procedimentos e regras que permitem remover partes dessa árvore.

Durante muito tempo este foi o método mais popular para resolução de problemas de *scheduling*. Apesar de ser adequado quando se trata de problemas com  $n < 250$ , o tempo computacional requerido é muito grande [Jain e Meeran, 1999].

### **2.2.2. Beam Search**

O método *Beam Search* incorpora heurísticas na procura que eliminam em cada nível da árvore, parte das soluções possíveis. A procura é feita numa determinada direcção, sem que haja retrocesso (*backtracking*) [Jain e Meeran, 1999]. Neste caso já não há garantia de encontrar a solução óptima.

### **2.2.3. Priority Dispatching Rules**

Este é o método mais popular e simples de implementar, consiste em atribuir prioridades às operações que se encontram à espera, escolhendo a que tiver maior prioridade para ser processada.

[Baker, 1974] apresenta algumas regras de atribuição de prioridades de execução dos *jobs*. Estas dividem-se essencialmente em regras locais ou globais, e em estáticas ou dinâmicas. Uma regra é considerada local, se na atribuição de prioridades usa somente informação da máquina na qual a operação está em fila de espera, e é considerada global

se utiliza informação de outras máquinas para além daquela onde se formou a fila de espera.

Nas regras Estáticas, a prioridade atribuída não se altera ao longo do tempo, enquanto nas dinâmicas ela pode ser alterada.

Como exemplos de regras podemos encontrar:

*SPT (Shortest Processing Time)*: atribui maior prioridade às operações que requerem o menor tempo de processamento;

*LWKR (Least Work Remaining)*: atribui maior prioridade às operações cujo *job* associado precisa do menor tempo até terminar o processamento;

*AWINQ (Anticipated Work in Next Queue)*: é dada maior prioridade às operações em espera, cuja operação subsequente tem o menor tempo de espera na fila (*queue*);

*FOFO (First Off First On)*: é dada maior prioridade à operação que pode ser completada primeiro (mesmo que ela ainda não tenha chegado à fila, a máquina fica à espera);

*FASFS (First Arrival at the Shop First Served)*: é dada maior prioridade aos *jobs* que chegam primeiro;

*TWORK (Total Work)*: é dada maior prioridade aos *jobs* com o menor tempo total de processamento;

*EDD (Earliest Due Date)*: é dada maior prioridade ao *job* com a data de entrega mais próxima;

*FCFS (First Come First Served)*: dá maior prioridade às operações que chegam primeiro à fila (*pré-shop pool*);

*MST (Minimum Slack Time)*: dá maior prioridade aos *jobs* cuja folga (*slack time*) é menor (folga é a diferença entre a data de entrega e a data de conclusão da ordem se não houver atraso);

*OPNDD (Operation Due Date)*: é dada maior prioridade à operação em espera com a menor data de entrega operacional (data de entrega operacional é a diferença entre a data de entrega e a data de chegada ao sistema, a dividir pelo número de operações que constituem o *job*);

*S/OPN (Slack per Operation)*: é dada maior prioridade às operações em espera, cujo respectivo *job* tem o menor rácio entre a folga e as operações ainda não processadas;

*TSPT (Truncated SPT)*: atribui maior prioridade às operações em espera que requerem um menor tempo de processamento.

Algumas das regras de processamento estáticas (ex: SPT, LWKR e OPNDD) podem ser consideradas dinâmicas se algumas das operações do *job* vão adquirindo prioridades diferentes.

Estas regras são muito fáceis de implementar e necessitam de pouco tempo de computação, no entanto nem sempre conseguem bons resultados.

#### **2.2.4. Shifting Bottleneck Procedure (SBP)**

Este método tem quatro etapas: identificação de sub-problemas, selecção de pontos de estrangulamento (*bottleneck*), solução de sub-problemas, e re-otimização de programas. A estratégia actual envolve o relaxamento do problema em  $m$  problemas com uma máquina, resolvendo um sub-problema de cada vez, obtêm-se  $m$  soluções (uma por cada máquina). Pela comparação de todas as soluções, cria-se um ranking das máquinas, e a que tiver o pior resultado é considerada a máquina “*bottleneck*” pelo que é aquela que é planeada primeiro. O SBP faz a distribuição das operações pela máquina *bottleneck*, enquanto fixa as máquinas já planeadas e ignora as restantes. Após esta operação, o SBP faz a re-otimização das máquinas já anteriormente planeadas.

Os problemas deste método são a sua dificuldade em fazer a re-otimização e a geração de soluções não admissíveis [Jain e Meeran, 1999].

#### **2.2.5. Local Search**

Estes métodos têm, por base, a ideia de que na vizinhança de uma boa solução, pode existir uma solução ainda melhor. Portanto, vão-se fazendo pequenas alterações nas boas soluções, na tentativa de encontrar melhores soluções. Os métodos de procura local exploram um espaço de procura, definido pela vizinhança de uma determinada solução. Através duma função de custos, é possível avaliar o conjunto de soluções e escolher a nova solução que servirá de referência na iteração seguinte. Este processo repete-se até que seja satisfeita uma determinada condição de paragem [Bael e Rijckaert, 2002].

### **2.2.6. Threshold Algorithms**

De acordo com [Jain e Meeran, 1999], este é o método iterativo mais popular, em que se escolhem soluções por comparação com as soluções vizinhas. Se a diferença em termos de custos, entre a solução corrente e o seu vizinho estiver abaixo de um determinado valor (*threshold*), então escolhe-se essa solução para continuar a procura.

### **2.2.7. Simulated Annealing (SA)**

Este é um método de procura local “aleatório”, introduzido como uma analogia entre a simulação do arrefecimento de sólidos e a resolução de problemas de optimização combinatoria. Com o objectivo de simular o comportamento de um sólido evoluindo para o equilíbrio térmico a uma dada temperatura, [Metropolis et al., 1953] investigaram o método de Monte Carlo. Este método gera uma sequência de estados que gradualmente vão atingindo um equilíbrio térmico. O novo estado é obtido a partir do estado actual, que é descrito pelo posicionamento das partículas do sólido, provocando-se de forma aleatória uma pequena perturbação numa partícula seleccionada aleatoriamente. Seguidamente, é determinada a diferença entre o novo estado e o actual. Caso a diferença seja negativa, o novo estado passa a ser o actual. Caso contrário, a probabilidade do novo estado ser aceite é dada pela função:  $\exp(-\Delta E/Kb T)$ . Esta regra de aceitação é chamada de “critério de Metropolis” e pode ser usado na resolução de problemas de optimização combinatoria.

Em problemas de optimização, os vários estados do sólido são as várias soluções possíveis, a temperatura é denominada de parâmetro de controlo (diminuído a cada execução), a constante de Boltzmann  $Kb$  não é utilizada e a energia do sistema é substituída pela função objectivo.

Apesar de SA conseguir bons resultados, este método ainda precisa de muito tempo de computação. Como resultado, em pesquisas recentes *simulated annealing* tem sido combinado com outros métodos, de forma a obter bons resultados diminuindo o tempo de processamento [Jain e Meeran, 1999].

### **2.2.8. GRASP (Greedy Randomized Adaptive Search Procedure)**

GRASP, é um método iterativo, onde cada iteração GRASP é composta por duas fases: construção e procura local. A fase de construção permite identificar soluções possíveis, cuja vizinhança é depois explorada na fase da procura local. A melhor solução encontrada no conjunto de todas as iterações é a solução final para o problema.

Relativamente ao problema de *scheduling*, na fase de construção, em cada iteração o GRASP criar uma solução (*schedule*) admissível. No entanto, como nada garante que essa é a solução ótima local, na segunda fase deste método, vai ser efectuada uma procura na vizinhança para detectar se existe uma solução melhor. Nesta última fase, se for encontrada uma solução vizinha melhor do que a definida na fase de construção, fica-se com a solução vizinha, e passa-se a outra iteração do método. No final escolhe-se a melhor solução encontrada no conjunto de todas as iterações deste método [Binato et al., 2002]. O algoritmo GRASP implementado por [Binato et al., 2002], nos 66 problemas onde foi aplicado conseguiu obter soluções ótimas ou quase ótimas para todos os problemas, no entanto tem o problema do tempo de computação ainda ser significativo.

### **2.2.9. Procura Tabu (PT)**

O método de Procura Tabu é também um método de procura local baseado no conceito de vizinhança. A ideia base deste método é explorar o espaço de soluções admissíveis através de uma sequência de movimentos. O movimento de uma solução para outra solução é feito avaliando todos os candidatos e escolhendo o melhor disponível. Alguns movimentos são classificados como tabu (proibidos) por conduzirem a ótimos locais ou a soluções já antes visitadas. Esses movimentos são guardados naquilo a que se chamou a Lista Tabu, que é no fundo o historial de movimentos já efectuados durante a pesquisa. Com esta lista de movimentos Tabu, consegue-se guiar a procura para uma região do espaço ainda não explorada [Jones e Rabelo, 1998].

Este método geralmente consegue bons resultados, no entanto o tempo de computação requerido depende muito da complexidade do problema [Mahmood e Hyasat, 2002].

### **2.2.10. Algoritmos Genéticos (AGs)**

Os Algoritmos Genéticos surgiram à mais de 30 anos como uma aproximação à biologia evolutiva. Este é um método iterativo, em que a partir de um conjunto de soluções iniciais se vão executando operações, inspiradas na evolução biológica, que permitem melhorar essas soluções.

Vários autores têm proposto algoritmos genéticos para a resolução de problemas de *scheduling*, apesar de estes não garantirem soluções óptimas. Os AGs permitem abordar uma gama muito grande de problemas diferentes e são bons a explorar todo o espaço de procura. No próximo capítulo são estudados com mais detalhe e rigor.

## 3. Algoritmos Genéticos

### 3.1. Introdução

O termo Algoritmo Genético (AG) foi pela primeira vez usado por John Holland, tendo recentemente alguns autores usado também os termos Computação Evolutiva e ou Algoritmos Evolutivos [Glover e Kochenberger, 2003]. Algoritmos Genéticos surgiram há mais de 30 anos como uma aproximação à biologia evolutiva. Na natureza, uma população de indivíduos reproduz-se originando “melhores” populações de acordo com um processo natural de selecção [Nearchou, 2004]. De forma similar, um algoritmo genético é executado durante  $m$  gerações, e em cada geração, através de operações de selecção, cruzamento e mutação, vão sendo introduzidas alterações que permitem melhorar as novas gerações. Por geração entende-se uma iteração completa do algoritmo genético que permitiu gerar uma nova população. A população é o conjunto de indivíduos existentes numa determinada geração, e que representam as soluções candidatas numa iteração do problema. Cada indivíduo representa uma solução possível, e em cada geração, cada um destes indivíduos é avaliado usando uma função de avaliação também denominada de função de *fitness*. De acordo com essa avaliação irão ser seleccionados os indivíduos que darão origem à nova população.

Assim, para implementar um algoritmo genético é necessário definir: a representação dos cromossomas<sup>ii</sup>, a forma de iniciar a população, a função de *fitness* que permitirá avaliar cada indivíduo, os operadores de selecção, cruzamento e mutação a usar e o valor de alguns parâmetros do AG como sejam o tamanho da população, o número de gerações e outros relacionados com os operadores [Nearchou, 2004].

Na Figura 7 podemos ver a estrutura básica de funcionamento de um algoritmo genético.

---

<sup>ii</sup>cromossoma - conjunto de bits que permitem codificar cada indivíduo da população. Esta codificação pode ser através de números binários, inteiros, etc.

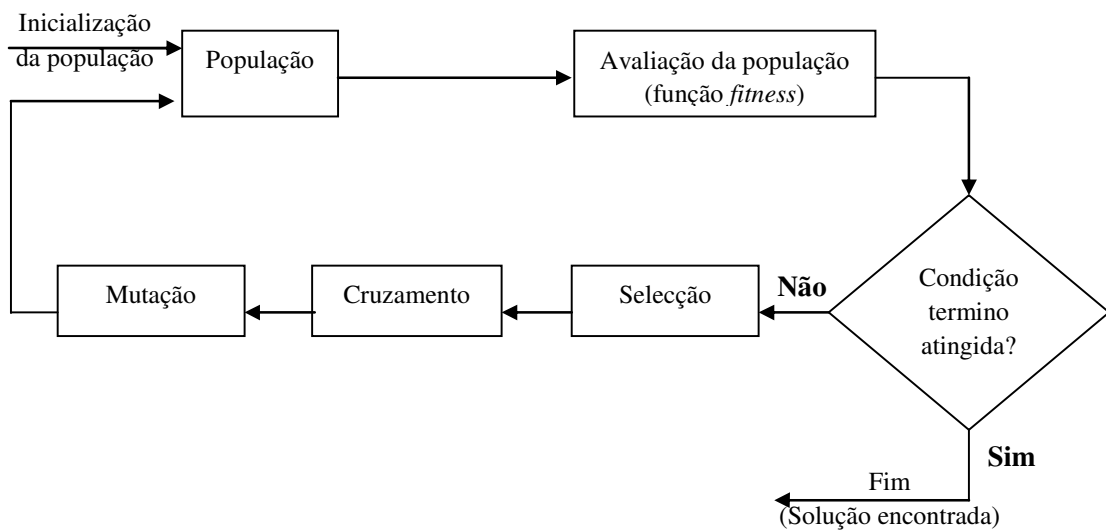


Figura 7. Representação de uma iteração de um algoritmo genético simples.

Após criar a população, é necessário avaliar cada um dos seus indivíduos. De acordo com o problema e objectivo a atingir deve ser definida a função que permita fazer esta avaliação.

Os operadores de selecção, permitem seleccionar os indivíduos de uma determinada população de acordo com um determinado critério (função de *fitness*). Os operadores de cruzamento permitem cruzar informação genética de dois indivíduos pai dando origem a novos indivíduos. Os operadores de mutação permitem introduzir nova informação genética na população.

### 3.2. Revisão da Literatura

Os algoritmos genéticos têm vindo a ser usados em várias áreas de trabalho e investigação, nomeadamente para resolver problemas relacionados com a problemática do *Job-Shop*. Tipicamente estuda-se uma versão simples deste problema que se caracteriza por ter  $n$  jobs a processar em  $m$  máquinas. Na resolução deste problema assume-se que uma máquina só pode processar um *job* de cada vez, e o processamento de um *job* numa máquina é chamado operação. Uma operação depois de começar a ser processada não pode ser interrompida, sendo que um *job* é composto, no máximo, por  $m$  operações. A sequência de processamento das operações de um *job* é chamada de *machine sequence* e é conhecida à priori bem como os respectivos tempos de

processamento. A sequência de operações a executar em cada máquina é desconhecida, sendo o conjunto de alternativas possíveis denominadas de representações simbólicas. Cada uma destas representações que seja admissível é chamada de programa ou *schedule*.

Num problema do tipo  $n \times m$  ( $n$  jobs a processar em  $m$  máquinas) existem  $(n!)^m$  soluções possíveis, pelo que facilmente nos apercebemos que à medida que aumentam o número de *jobs* e de máquinas, torna-se impraticável testar todos os cenários possíveis. Por outro lado, sabemos que os algoritmos genéticos têm a capacidade de identificar soluções possíveis, e de geração em geração ir melhorando essas soluções. Os algoritmos genéticos apesar de não garantirem a solução ótima garantem boas soluções.

Para auxiliar o desenvolvimento do nosso algoritmo, foi feita uma pesquisa na literatura para determinar o que é mais usado e o que tem apresentado melhores resultados.

### 3.2.1. Representação dos cromossomas

Alguns autores sugerem um cromossoma constituído por uma *string* de números, tal como podemos ver na Figura 8.

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 2 | 1 | 3 | 1 | 2 | 3 | 3 |
|---|---|---|---|---|---|---|---|---|

Figura 8. Representação do cromossoma para um problema do tipo 3x3.

Cada gene contém informação sobre o *job*/operação que vai ser processado.

Pelo cromossoma representado, podemos dizer que inicialmente será processada a primeira operação do *job* 1, seguidamente a 1ª operação do *job* 2, depois segue a 2ª operação do *job* 2, depois vem a 2ª operação do *job* 1, e por aí fora.

Esta parece ser a representação mais usada na literatura por ser simples de implementar e garantir que após operações de cruzamento e mutação se continuam a obter soluções válidas. Esta representação foi a escolhida por [Tsujimura et al., 2001], [Wang e Zheng, 2002], [Park et al., 2003], entre outros.

[Ombuki e Ventresca, 2004] sugerem a representação ilustrada na Figura 9.

|         |         |         |         |         |         |         |          |         |
|---------|---------|---------|---------|---------|---------|---------|----------|---------|
| (1,3,1) | (2,2,8) | (3,2,3) | (1,1,3) | (2,3,5) | (3,1,8) | (1,2,6) | (2,1,10) | (3,3,1) |
|---------|---------|---------|---------|---------|---------|---------|----------|---------|

Figura 9. Representação do cromossoma, para um problema do tipo 3x3.

|       |       |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| (1,3) | (2,2) | (3,2) | (1,1) | (2,3) | (3,1) | (1,2) | (2,1) | (3,3) |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|

Figura 10. Representação simplificada do cromossoma, para um problema do tipo 3x3.

Cada gene que compõe o cromossoma tem informação de qual o *job*, máquina e tempo de processamento. Na versão simplificada do cromossoma, o tempo de processamento não faz parte da informação contida no gene. Esta representação apesar de ser mais completa é mais difícil de implementar.

### 3.2.2. Função de avaliação

A função de avaliação mais usada na literatura é a minimização do tempo total necessário para o processamento de todas as actividades (*Makespan*). Esta função foi por exemplo usada por [Bierwith e Mattfeld, 1999], [Wang e Zheng, 2002], [Nearchou, 2004], [Park et al., 2003], [Ombuki e Ventresca, 2004] e [Gonçalves et al., 2005].

[Tsujimura et al., 2001] apresentam 3 algoritmos. No primeiro usam uma função dada pelo tempo total de processamento de todas as operações, no segundo combinam o tempo total de processamento com o tempo de ocupação das máquinas, e no terceiro combinam o tempo total de processamento com o tempo de ocupação das máquinas e o tempo total de paragem das máquinas.

[Mattfeld e Bierwirth, 2004] usaram uma função diferente, resultado da combinação entre o tempo total de processamento de todas as operações, mais a diferença entre o prazo de entrega e a data efectiva de fim de processamento e a informação de se e quantos *jobs* acabaram de processar com atraso.

### 3.2.3. Operadores de Selecção

Operadores de selecção, são usados para seleccionar os indivíduos de uma geração que passam para a geração seguinte. A seguir apresentamos os mais usados na literatura.

♦ *Proportional Selection ou Roulette Wheel* – com este operador a probabilidade de um indivíduo ser seleccionado é proporcional ao seu valor de *fitness*. Se

imaginarmos uma roleta, em que cada fatia representa um indivíduo, e a área da fatia é proporcional ao valor da sua função de *fitness*, então a probabilidade de um indivíduo ser seleccionado é tanto maior quanto melhor for o seu valor de *fitness*. Este método dá oportunidade aos piores indivíduos para que também eles possam ser seleccionados. Este método foi usado por exemplo por [Tsujiura et al., 2001] e [Ombuki e Ventresca, 2004].

♦ *Tournament Selection* - através deste operador é seleccionado um grupo de indivíduos, da população, de forma aleatória. Os indivíduos seleccionados são depois comparados entre si, através da sua função de *fitness*, e só o que tiver maior valor é que passa para a geração seguinte. Geralmente seleccionam-se grupos de 2 indivíduos (*Binary Tournament*). Este algoritmo foi usado por [Soares et al., 1995], [Nearchou, 2004], [Park et al., 2003], entre outros.

♦ *Scaling in Tournament Selection* - este operador é semelhante ao anterior, diferencia-se por introduzir um mecanismo de aceitação (*threshold*), na selecção. Cada vez que se selecciona um grupo de indivíduos, é gerado um valor  $\theta$  compreendido entre 0 e 1, e se o valor  $\theta$  for menor que  $\tau$  (*threshold*), então selecciona-se o que tiver maior valor de *fitness*, caso contrário selecciona-se o outro.  $\tau$  é um parâmetro definido no início, e normalmente está compreendido entre 0.75 e 1.

♦ *Elitist Selection* – através deste operador começam por se ordenar os indivíduos da população de acordo com um determinado critério (normalmente é usado o valor da função de *fitness*). Depois, é usada uma regra para seleccionar os indivíduos. Esta regra pode, por exemplo, ser a de seleccionar os 10% melhores indivíduos da população. Este operador foi usado por [Ikeuchi et al., 1998], [Wang e Zheng, 2002], [Gonçalves et al., 2005], entre outros.

### 3.2.4. Operadores de Cruzamento

Os operadores de cruzamento permitem trocar informação genética entre dois indivíduos da população. A seguir apresentamos os mais usados na literatura.

♦ *OX (Order Crossover)* - através deste operador é seleccionado um intervalo de genes de um dos cromossomas pai, que são posteriormente copiados para as mesmas posições do cromossoma filho. Os restantes genes do filho são copiados a partir do segundo pai, preservando a ordem. Podemos encontrar este operador em [Soares et al., 1995].

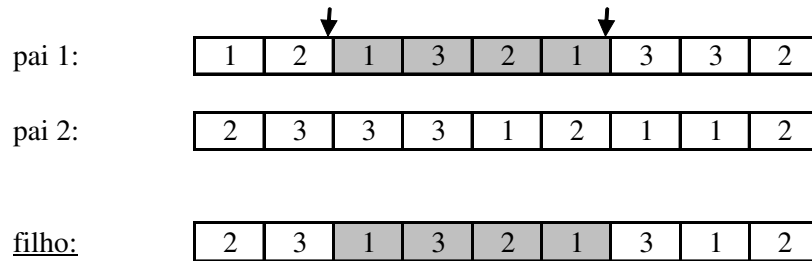


Figura 11. Esquema de funcionamento do operador OX.

♦ *One-point crossover* – com este operador seleccionam-se dois cromossomas pai e aleatoriamente é definido um ponto de corte no primeiro cromossoma. Seguidamente copia-se a primeira parte deste cromossoma pai para o cromossoma filho e os restantes genes são copiados do segundo pai, preservando a ordem e consistência do filho. Este operador foi usado por exemplo por [Nearchou, 2004]. Ver esquema de funcionamento na Figura 12.

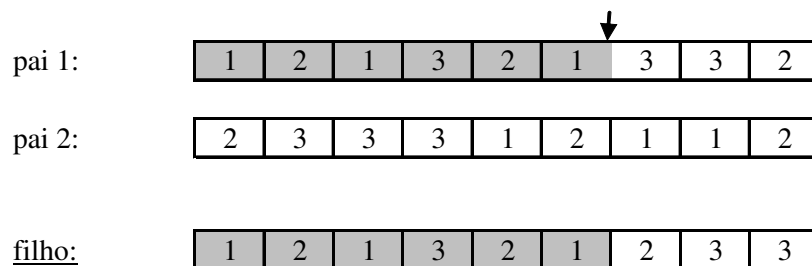


Figura 12. Esquema de funcionamento do operador One-Point Crossover.

♦ *Two-point crossover* – este operador é semelhante ao operador *order crossover*. Seleccionam-se dois cromossomas pai, e aleatoriamente são definidos dois pontos de corte no primeiro deles. Este operador tem duas formas, na primeira copia-se a primeira e terceira parte do primeiro cromossoma pai para o filho e a restante informação vem do segundo pai. Na segunda forma, copia-se a segunda parte do primeiro cromossoma pai para o filho e a restante informação vem do segundo cromossoma pai. Em ambas as formas quando se copiam os genes do segundo

cromossoma pai, mantém-se a ordem e consistência da solução. Este operador foi usado por exemplo por [Nearchou, 2004]. Ver esquema representativo na Figura 13.

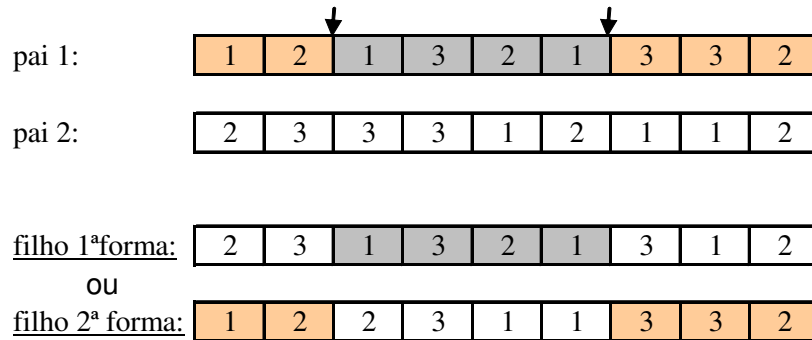


Figura 13. Esquema de funcionamento do operador *Two-Point Crossover*.

♦ *Three-point crossover* – este operador é semelhante aos anteriores, a diferença é que aqui são definidos três pontos de corte. Também aqui há duas formas, na primeira é copiada a primeira e terceira parte do primeiro cromossoma pai sendo a restante informação copiada do segundo cromossoma pai, e na segunda forma é copiada a segunda e quarta parte do primeiro cromossoma pai e a restante informação genética vem do segundo cromossoma. Também aqui é respeitada a ordem e consistência dos cromossomas filhos. Operador usado por exemplo por [Nearchou, 2004].

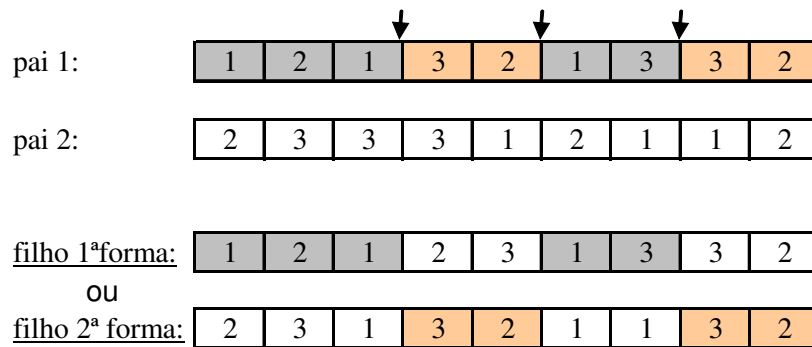


Figura 14. Esquema de funcionamento do operador *Three-Point Crossover*.

♦ *Position Based Crossover também chamado de UOX* – através deste operador são seleccionados dois cromossomas pai, e do primeiro pai são seleccionados aleatoriamente alguns genes. Posteriormente esses genes são copiados para as mesmas posições do cromossoma filho. A restante informação cromossoma filho vem do segundo pai, preservando a ordem e a consistência. Este operador foi usado por exemplo por [Nearchou, 2004].

|               |   |   |   |   |   |   |   |   |   |
|---------------|---|---|---|---|---|---|---|---|---|
| pai 1:        | 1 | 2 | 1 | 3 | 2 | 1 | 3 | 3 | 2 |
| pai 2:        | 2 | 3 | 3 | 3 | 1 | 2 | 1 | 1 | 2 |
| <u>filho:</u> | 1 | 2 | 1 | 3 | 2 | 1 | 3 | 3 | 2 |

Figura 15. Esquema de funcionamento do operador *position based crossover*.

♦ *Precedence Preservative Crossover (PPX)* – com este operador seleccionam-se dois cromossomas pai, e é criado um “cromossoma” máscara, contendo em cada gene o valor 1 ou 2 (gerados aleatoriamente). Seguidamente cria-se o cromossoma filho, que vai herdar informação do primeiro pai ou do segundo de acordo com a informação contida na máscara. Se o gene da máscara tiver o valor 1, então vai-se buscar o valor do gene ao primeiro pai, caso contrário vai-se buscar ao segundo pai. Também aqui se preserva a ordem dos genes nos pais, mas mantendo a consistência do filho. Ver esquema de funcionamento na Figura 16.

|               |   |   |   |   |   |   |   |   |   |
|---------------|---|---|---|---|---|---|---|---|---|
| pai 1:        | 1 | 2 | 1 | 3 | 2 | 1 | 3 | 3 | 2 |
| pai 2:        | 2 | 3 | 3 | 3 | 1 | 2 | 1 | 1 | 2 |
| mascara:      | 1 | 2 | 2 | 1 | 1 | 2 | 1 | 2 | 1 |
| <u>filho:</u> | 1 | 2 | 3 | 2 | 1 | 3 | 3 | 1 | 2 |

Figura 16. Esquema de funcionamento do operador *precedence preservative crossover*.

Este operador foi usado por exemplo por [Bierwirth e Mattfeld, 1999] e [Mattfeld e Bierwirth, 2004].

♦ *TaskCrossover* – este operador é uma variante do UOX, usado por [Ombuki e Ventresca, 2004], em que para garantir a consistência dos novo cromossomas filhos, quando se selecciona um gene do pai, copiam-se os genes que contêm operações do *job* seleccionado, para o cromossoma filho. Os restantes genes são copiados do outro pai.

♦ [Wang e Wu, 2004] propõem um operador de cruzamento em que se seleccionam aleatoriamente dois cromossomas pai, e os genes que forem comuns (com

o mesmo valor) são copiados para o cromossoma filho. Posteriormente procede-se como no *one-point crossover*, tal como se pode ver na Figura 17.

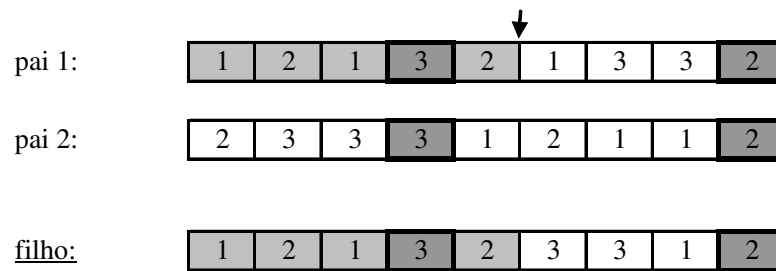


Figura 17. Esquema de funcionamento do operador usado por [Wang e Wu, 2004].

♦ [Wang e Zheng, 2002] propõem o operador a que chamam “*special crossover operation*”, cujo funcionamento é muito semelhante ao operador *precedence preservative crossover*. Neste operador seleccionam-se dois cromossomas pai, e criam-se dois subconjuntos A1 e A2 cada um com pelo menos um elemento, a partir de um conjunto A (constituído por  $n \times m$  elementos). Depois criam-se dois filhos, sendo que o primeiro herda os genes do primeiro pai que pertencem ao subconjunto A1, e os genes do segundo pai que pertencem ao subconjunto A2. O segundo filho herda os genes contidos em A2 que pertencem ao primeiro pai e os genes identificados em A1 que pertencem ao segundo pai. Ver esquema representativo na Figura 18.

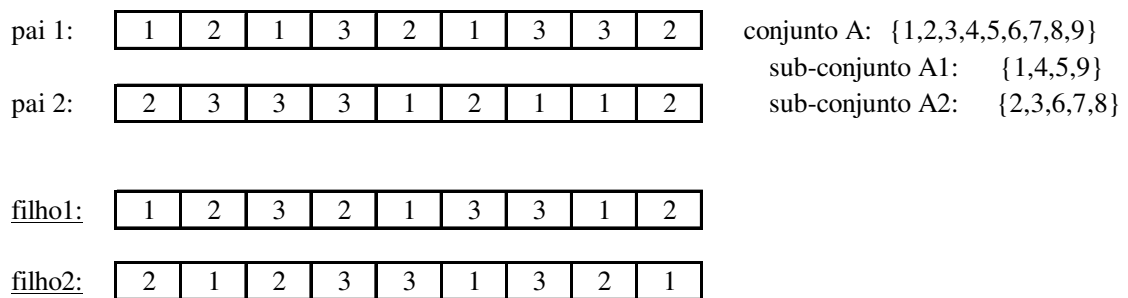


Figura 18. Esquema de funcionamento do operador usado por [Wang e Zheng, 2002].

♦ *Partial Schedule Crossover* – este operador é implementado em duas etapas. Inicialmente são seleccionados dois cromossomas pai, e aleatoriamente definem-se dois pontos de corte, copia-se a primeira parte de um pai, a segunda do outro e a terceira do primeiro. Seguidamente, para garantir a consistência dos cromossomas filhos, vão comparar-se os genes que trocaram de posição, para detectar possíveis genes em falta ou em excesso. Como se pode ver pela Figura 19, o gene a mais do primeiro filho troca

de posição com o gene em falta do segundo filho. Este operador foi usado por exemplo por [Tsujimura et al., 2001].

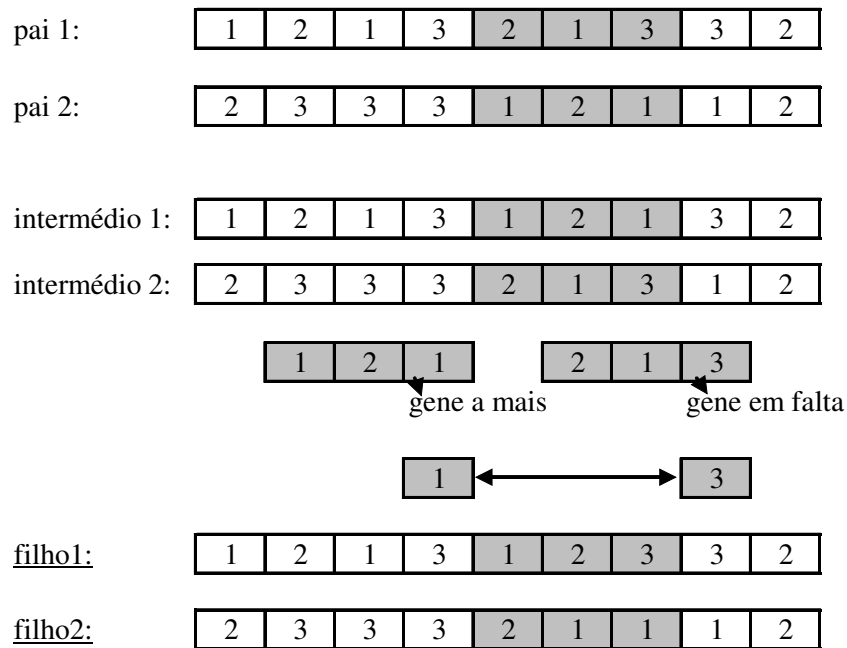


Figura 19. Esquema de funcionamento do operador *Partial Schedule Crossover*.

### 3.2.5. Operadores de Mutação

Os operadores de mutação permitem introduzir informação genética nova nos indivíduos que constituem a população, sendo que alguns dos operadores que podemos encontrar na literatura são os descritos a seguir.

♦ *Shift Mutation* (também chamado de *PBM - Position-Based Mutation*) - através deste operador é seleccionado aleatoriamente um gene do cromossoma, que depois é eliminado, e inserido noutra posição também ela seleccionada aleatoriamente. Este operador foi usado por exemplo por [Ikeuchi et al., 1998], [Bierwirth e Mattfeld, 1999], [Wang e Zheng, 2002], [Nearchou, 2004] e [Mattfeld e Bierwirth, 2004].

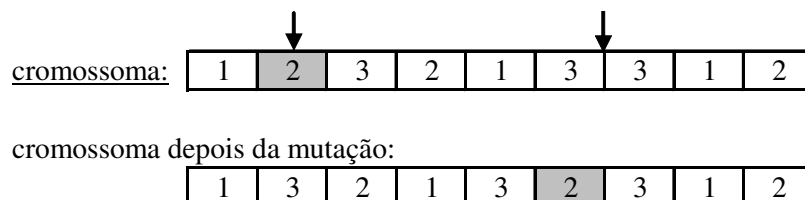


Figura 20. Esquema de funcionamento do operador *Shift Mutation*.

♦ *Random Exchange Mutation (também chamado de swap)* – através deste operador são seleccionados aleatoriamente dois genes do cromossoma, e trocam-se os seus valores. Este operador foi usado por [Ikeuchi et al., 1998], [Wang e Zheng, 2002], [Nearchou, 2004], entre outros.

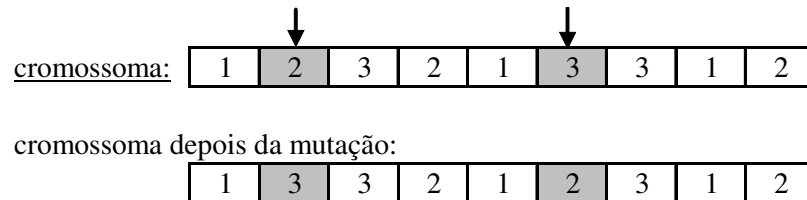


Figura 21. Esquema de funcionamento do operador *Random Exchange Mutation*.

♦ *Adjacent Exchange Mutation* – com este operador são seleccionados aleatoriamente dois genes adjacentes de um cromossoma, e os *jobs* contidos nesses genes são trocados. Operador usado por exemplo por [Nearchou, 2004] e [Wang e Wu, 2004].

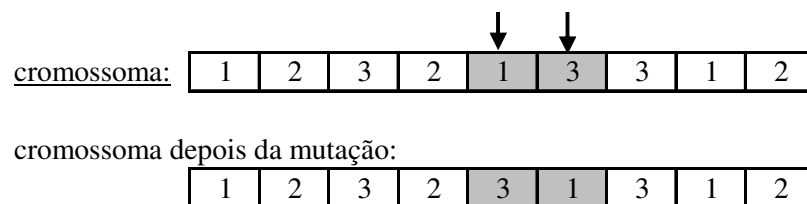


Figura 22. Esquema de funcionamento do operador *Adjacent Exchange Mutation*.

♦ *Displacement Mutation* – através deste operador é seleccionada uma sequência de genes e movem-se para uma posição seleccionada aleatoriamente. Usado por exemplo por [Nearchou, 2004].

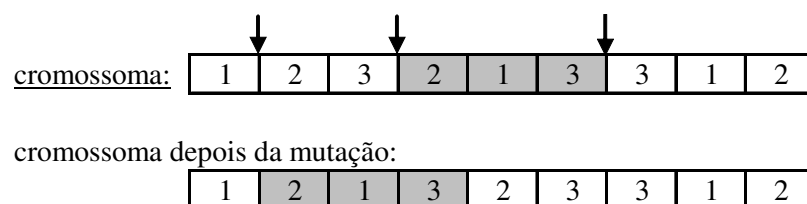


Figura 23. Esquema de funcionamento do operador *Displacement Mutation*.

♦ *Inversion Mutation* – este operador selecciona aleatoriamente duas posições do cromossoma, e inverte o valor dos genes no intervalo seleccionado. Este operador foi usado por exemplo por [Wang e Zheng, 2002] e [Nearchou, 2004].

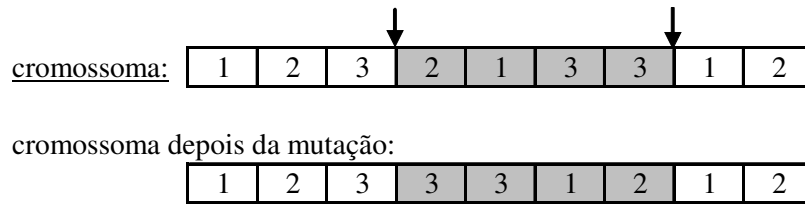


Figura 24. Esquema de funcionamento do operador *Inversion Mutation*.

♦ *Inversion/Displacement Mutation* – este operador é um misto dos dois operadores anteriores. Selecciona-se um intervalo de genes e invertem-se os seus valores e posteriormente movem-se para uma posição seleccionada aleatoriamente. Este operador foi usado por [Nearchou, 2004].

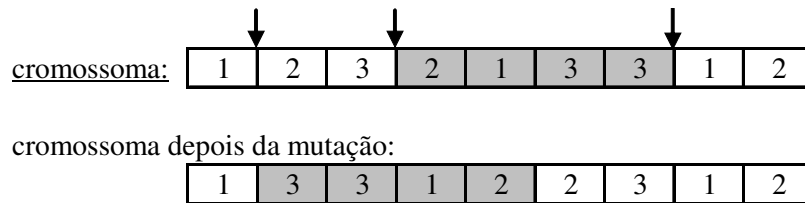


Figura 25. Esquema de funcionamento do operador *Inversion/Displacement Mutation*.

♦ [Ombuki e Ventresca, 2004], usaram um operador de mutação a que chamaram *mutation operation*. Através deste operador seleccionam-se aleatoriamente uma máquina e dois *jobs* consecutivos dessa máquina, e trocam-se as suas posições.

### 3.2.6. Parâmetros de controlo

[Bierwirth e Mattfeld, 1999] usaram uma população de 100 indivíduos, e uma probabilidade de mutação e cruzamento de 0.2 e 0.8 respectivamente.

[Liaw, 2000] usou uma população de 121 indivíduos, e 150 gerações. A probabilidade de cruzamento usada foi de 0.8 e a probabilidade de mutação de 0.2.

[Nearchou, 2004] usou uma população de 30 indivíduos, e 10000 gerações. Se depois de 2000 gerações consecutivas não se verificar nenhuma melhoria na solução então o algoritmo pára a sua execução. A probabilidade de cruzamento usada foi de 1 e a de mutação foi de 0.8. Esta probabilidade de mutação diminui em cada iteração para  $Pm = \theta Pm$  (com  $\theta = 0.99$ ). Quando a diversidade da população for muito baixa então a  $Pm$  (probabilidade de mutação) volta a ser de 0.8.

[Ombuki e Ventresca, 2004] correram o algoritmo durante 550 gerações, com uma população de 200 indivíduos, e uma probabilidade de mutação e cruzamento de 0.15 e 0.95, respectivamente.

[Gonçalves et al., 2004] executaram o seu algoritmo durante 400 gerações e usaram uma população de 2 vezes o número de operações. A probabilidade de cruzamento usada foi de 0.7.

[Azimi, 2005] no AG usou populações de 10, 50, 100 e 200 indivíduos. As probabilidades de mutação usadas foram de 0.02, 0.1 e 0.5. As probabilidades de cruzamento foram de 0.5, 0.8 e 1. De acordo com o autor, os melhores resultados foram obtidos usando uma população de 100 indivíduos com probabilidade de cruzamento de 0.8 e probabilidade de mutação de 0.02.

[Watanabe et al., 2005] usaram uma população de 400 indivíduos. Em vez do parâmetro gerações usaram o número de soluções visitadas e que foi de 3000000. As probabilidades de mutação e cruzamento foram de 0.2 e 0.8, respectivamente.

[Park et al., 2003] usaram populações de 200 indivíduos, e probabilidades de cruzamento de 0.6, 0.7 e 0.8. A probabilidade de mutação usada foi de 0.1.

### **3.3. Metodologia Usada**

Esta secção foi dividida em duas partes, na primeira analisamos os operadores e parâmetros de controlo usados explicando a razão das escolhas. Na segunda parte explicamos a implementação do algoritmo genético.

#### **3.3.1. Concepção**

Optamos por implementar estes algoritmos em R, por ter sido objecto de estudo na parte lectiva deste mestrado.

Os casos de estudo que serviram de base para este trabalho foram os denominados *benchmark problems*. A escolha destes problemas prende-se com o facto deles serem comumente usados na literatura.

Na implementação do Algoritmo Genético tentou-se que fosse o mais parecido possível com os algoritmos genéticos que são propostos pelos variados autores.

Depois de efectuar uma análise detalhada a trabalhos publicados sobre o assunto, verificamos que a maior parte dos autores usam como função de avaliação das soluções encontradas, a minimização do tempo necessário para processar todas as operações (*makespan*). Daí a decisão de usarmos esta função de avaliação.

Relativamente à representação dos cromossomas, optamos por usar a representação de [Wang et Zheng, 2002], [Park et al., 2003], entre outros. Esta representação além de ser frequentemente usada, é simples de implementar, e fácil de manusear na aplicação dos operadores.

Como operador de selecção, foi aplicado o *Elitist Selection*. A regra usada foi a de seleccionar os melhores 10% de indivíduos de uma geração para passar para a geração seguinte. Desta forma garantimos que as melhores soluções de uma geração não se perdem. Além deste operador foi também usado o *Tournament Selection*, mais precisamente o *Binary Tournament*. Através deste operador são seleccionados dois indivíduos da população, mas só o melhor deles é que vai ser incluído na população da geração seguinte. Este operador foi usado por ser muito referenciado na literatura.

Para cruzar informação genética dos cromossomas, foi usado o operador *Two-Point Crossover*. De acordo com [Nearchou, 2004], das experiências que fez com vários operadores este foi o que obteve melhores resultados médios. Com este operador são gerados aleatoriamente dois pontos de corte num dos cromossomas pai, e copiados os genes entre os dois pontos de corte para as mesmas posições do cromossoma filho. Os restantes genes do filho são preenchidos com informação dos genes do segundo pai, respeitando a ordem e a consistência do cromossoma filho.

Para inserir nova informação genética nos cromossomas, foi usado o operador *Shift Mutation*, por ser um dos mais referenciados. Este operador escolhe aleatoriamente um gene e uma posição, depois elimina o gene e insere-o na posição escolhida.

A parte mais complicada foi escolher os parâmetros de controlo a usar. A variedade de parâmetros de controlo que são usados num AG, e as várias combinações possíveis faz com que encontrar a combinação perfeita seja quase impossível [Nearchou, 2004]. Assim, decidimos fazer alguns testes iniciais com algumas variantes de parâmetros, para nos ajudar na escolha. Nestes testes foram usadas populações de 50, 100 e 200 indivíduos, probabilidades de cruzamento de 1, 0.8 e 0.7, e probabilidades de mutação

de 0.01, 0.1 e 0.2. Detalhes dos resultados obtidos nestes testes podem ser consultados no anexo 1.

Depois de analisar os resultados obtidos e os valores usados pelos diversos autores, decidimos executar o nosso algoritmo com os 4 conjuntos de parâmetros apresentados no quadro da Figura 26.

|         | Nº Soluções Testadas | Nº Repetições | popSize AG | Pm AG | Pc AG |
|---------|----------------------|---------------|------------|-------|-------|
| Param 1 | 100000               | 10            | 50         | 0.01  | 1     |
| Param 2 | 100000               | 10            | 100        | 0.01  | 1     |
| Param 3 | 100000               | 10            | 50         | 0.1   | 1     |
| Param 4 | 100000               | 10            | 100        | 0.1   | 1     |

Figura 26. Número de soluções testadas, número de repetições do AG, tamanho da população, probabilidade de mutação e cruzamento usados

A maior parte dos autores usam probabilidades de mutação muito baixas e probabilidades de cruzamento muito próximas de 1. Por isso após efectuarmos alguns testes decidimos usar valores semelhantes. Para o cruzamento usamos uma probabilidade de 1, pois deu bons resultados nos primeiros testes que efectuamos e também tem sido usada por alguns autores, como sejam [Nearchou, 2004] e [Azimi, 2005]. Por outro lado ao usarmos uma probabilidade de cruzamento de 1, estamos a eliminar um parâmetro, o que pode eliminar alguma da aleatoriedade que daí poderia advir.

As probabilidades de mutação usadas foram definidas com base nos testes iniciais efectuados e nos valores referenciados na literatura.

Para a escolha da dimensão da população baseamo-nos nos nossos testes iniciais, nos trabalhos de outros autores e na opinião de [Reeves, 1999]. De acordo com este autor um dos factores que mais influência a performance de um AG é a dimensão da população. O autor menciona como valores de referência populações de 30 indivíduos, de  $n$  ou  $2n$ . Por isso decidimos testar valores intermédios entre os que ele sugere e os que são usados na literatura. [Azimi, 2005] dos testes que fez com populações de 10, 50, 100 e 200 indivíduos, diz que os melhores resultados foram obtidos com populações de 100.

Geralmente um AG em vez do número de soluções testadas usa o número de gerações como parâmetros, no entanto como nós pretendemos dar as mesmas oportunidades a

todos os algoritmos decidimos usar um parâmetro que também fosse possível de implementar na Procura Tabu.

### 3.3.2. Implementação

Este algoritmo é composto por várias funções, a que vou chamar blocos. O bloco principal descreve a estrutura de funcionamento genérica do AG. Este bloco recebe como parâmetros de entrada os parâmetros de controlo do AG: número de soluções a testar, tamanho da população, probabilidade de cruzamento e de mutação e retorna a população encontrada na última geração.

A estrutura base de funcionamento deste bloco é a descrita no quadro em baixo.

|                                                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------------------|
| - Ler tabela de precedências das operações dos <i>jobs</i>                                                                       |
| - Ler tempos de processamento das operações nas máquinas                                                                         |
| - Bloco: Gera população inicial                                                                                                  |
| - Bloco: Avalia cromossomas da população gerada                                                                                  |
| - Até atingir maxSolucoes=100000 fazer:                                                                                          |
| - Aplica Operador de Selecção: Selecciona os melhores $x=10\%$ indivíduos da população corrente e passa-os para a nova população |
| - Bloco: Aplica Operador de Cruzamento                                                                                           |
| - Bloco: Aplica Operador de Mutação                                                                                              |
| - Bloco: Avalia cromossomas da população gerada                                                                                  |
| - Retorna a melhor solução e o respectivo valor de <i>fitness</i>                                                                |

Figura 27. Algoritmo de funcionamento do bloco principal do AG.

O primeiro passo a dar neste algoritmo é ler as tabelas de tempos e precedências do problema corrente. A tabela de tempos contém o tempo requerido para processar cada operação na respectiva máquina.

|   |   | Máquinas |   |   |
|---|---|----------|---|---|
| J | 1 | 3        | 1 | 2 |
| o | 2 | 2        | 3 | 1 |
| b | 3 | 2        | 1 | 3 |
| s |   |          |   |   |

|   |   | Tempos |   |    |
|---|---|--------|---|----|
| J | 1 | 1      | 3 | 6  |
| o | 2 | 8      | 5 | 10 |
| b | 3 | 3      | 8 | 1  |
| s |   |        |   |    |

Figura 28. Tabela de precedências e tempos de processamento para um problema do tipo 3x3.

A tabela de precedências contém a descrição de todos os *jobs*, ou seja a ordem de processamento das suas operações nas respectivas máquinas. Assim, pela primeira tabela sabemos que o *job* 1 vai processar a primeira operação na máquina 3, a segunda operação na máquina 1 e a terceira na máquina 2. Pela segunda tabela sabemos que cada uma das operações que constituem o *job* 1 vão demorar, respectivamente, 1, 3 e 6 períodos de tempo.

O bloco gera população inicial, permite iniciar a população, sendo esta inicialização feita de forma aleatória.

O bloco avalia cromossomas da população gerada, vai avaliar cada um dos indivíduos que constituem a população corrente. Para efectuar esta avaliação, foram usadas as três estruturas de apoio descritas na figura em baixo.

| <b>job.fim</b> |       | <b>mach.free</b> |           |           |           | <b>operacao.job</b> |          |
|----------------|-------|------------------|-----------|-----------|-----------|---------------------|----------|
|                | Tempo |                  | maquina 1 | maquina 2 | maquina 3 |                     | operacão |
| Job 1          | 0     | tempo            | 0         | 0         | 0         | Job 1               | 0        |
| Job 2          | 0     |                  |           |           |           | Job 2               | 0        |
| Job 3          | 0     |                  |           |           |           | Job 3               | 0        |

Figura 29. Estruturas de apoio usadas na função de avaliação para cada indivíduo da população.

A estrutura *job.fim* contém um registo para cada *job* do problema, e indica o período de tempo em que cada um deles termina de processar a operação corrente. A estrutura *mach.free* contém um registo para cada máquina do problema, e indica o período de tempo em que cada máquina vai terminar de processar a última operação programada. A estrutura *operacao.job* contém um registo para cada *job* do problema, e indica a última operação programada para cada um deles.

Para cada indivíduo da população, a função de avaliação vai percorrer o cromossoma e para cada gene, de acordo com o valor aí contido, vai actualizando as três estruturas de apoio.

$$\left[ \begin{array}{cccccccccc} 1 & 1 & 3 & 3 & 2 & 2 & 2 & 1 & 3 \end{array} \right]$$

Figura 30. Exemplo de um cromossoma para um problema do tipo 3x3.

Se dermos como exemplo o cromossoma da Figura 30, correspondente a um problema do tipo 3x3, e cujas tabelas de precedências e tempos são as da Figura 28, teríamos:

- a primeira operação a executar seria a primeira operação do *job* 1, assim as três estruturas de apoio ficariam com os seguintes valores:

| <b>job.fim</b> |       | <b>mach.free</b> |           |           |           | <b>operacao.job</b> |          |
|----------------|-------|------------------|-----------|-----------|-----------|---------------------|----------|
|                | Tempo |                  | maquina 1 | maquina 2 | maquina 3 |                     | Operacao |
| Job 1          | 1     | tempo            | 0         | 0         | 1         | Job 1               | 1        |
| Job 2          | 0     |                  |           |           |           | Job 2               | 0        |
| Job 3          | 0     |                  |           |           |           | Job 3               | 0        |

Figura 31. Valor das três estruturas de apoio após programar a primeira operação do *schedule*.

Como a primeira operação do *job* 1 é processada na máquina 3 e precisa de um período tempo, é actualizada a estrutura *job.fim*, para o *job* 1 com o valor 1, e a estrutura *mach.free* para a máquina 3 com o valor 1. Além disso, é necessário actualizar a estrutura *operacao.job*, para o *job* 1 com o valor 1, que indica que já foi programada a primeira operação.

- o segundo gene do cromossoma tem o valor 1, pelo que indica que iremos processar uma operação do *job* 1, e como a estrutura *operacao.job* para o *job* 1 tem o valor 1, significa que a próxima operação a executar é a 2. A segunda operação do *job* 1 é processada na máquina 1 e demora 3 períodos de tempo, pelos que as três estruturas ficariam:

| <b>job.fim</b> |       | <b>mach.free</b> |           |           |           | <b>operacao.job</b> |          |
|----------------|-------|------------------|-----------|-----------|-----------|---------------------|----------|
|                | Tempo |                  | maquina 1 | maquina 2 | maquina 3 |                     | Operacao |
| Job 1          | 4     | tempo            | 4         | 0         | 1         | Job 1               | 2        |
| Job 2          | 0     |                  |           |           |           | Job 2               | 0        |
| Job 3          | 0     |                  |           |           |           | Job 3               | 0        |

Figura 32. Valor das três estruturas de apoio após programar a segunda operação do *schedule*.

- como o terceiro gene contém o valor 3, significa que vamos processar uma operação do *job* 3. Pela estrutura *operacao.job*, para o *job* 3, verificamos que a próxima operação a programar é a primeira com um tempo de processamento de 3 períodos de tempo, na máquina 2. Actualizando as três estruturas ficaríamos com:

| <b>job.fim</b> |       | <b>mach.free</b> |           |           |           | <b>operacao.job</b> |          |
|----------------|-------|------------------|-----------|-----------|-----------|---------------------|----------|
|                | Tempo |                  | maquina 1 | maquina 2 | maquina 3 |                     | Operacao |
| Job 1          | 4     | tempo            | 4         | 3         | 1         | Job 1               | 2        |
| Job 2          | 0     |                  |           |           |           | Job 2               | 0        |
| Job 3          | 3     |                  |           |           |           | Job 3               | 1        |

Figura 33. Valor das três estruturas de apoio após programar a terceira operação do schedule.

Para cada operação, a função vai verificar o que fica livre por último, se o *job* correspondente ou se a máquina onde deve ser feito o processamento, e com o pior valor vai actualizar os tempos da máquina e do *job*. Este mecanismo permite respeitar as restrições do problema: cada operação usa uma das *m* máquinas durante um período fixo de tempo, e cada máquina pode processar no máximo uma operação de cada vez, e após iniciar o seu processamento, este não pode ser interrompido.

Este processo repete-se para todos os genes que constituem o cromossoma. Após processar o último gene do cromossoma obteríamos as estruturas com os seguintes valores:

| job.fim |       | mach.free |           |           |           | operacao.job |          |
|---------|-------|-----------|-----------|-----------|-----------|--------------|----------|
|         | Tempo |           | maquina 1 | maquina 2 | maquina 3 |              | Operacao |
| Job 1   | 17    | tempo     | 26        | 17        | 17        | Job 1        | 3        |
| Job 2   | 26    |           |           |           |           | Job 2        | 3        |
| Job 3   | 17    |           |           |           |           | Job 3        | 3        |

Figura 34. Valor das três estruturas de apoio após programar todas as operações do Schedule.

No final, a função de avaliação verifica qual o *job* que termina o processamento das suas operações por último. Esse tempo corresponde ao tempo *makespan* do indivíduo e irá representar o seu valor.

O bloco aplica operador de cruzamento, além de cruzar indivíduos da população, também irá aplicar um operador de selecção para seleccionar os pais. Este recebe como parâmetros de entrada a população da geração anterior, a nova população e a *fitness* da população corrente. No final, retorna a nova população.

Nesta função começam por se seleccionar dois pais através do método *binary tournament*. Seleccionam-se dois indivíduos e escolhe-se o melhor deles para pai. Aos dois pais resultantes do processo de selecção, é-lhes aplicado o operador de cruzamento originando dois indivíduos filhos que serão adicionados à nova população. O operador de cruzamento implementado foi o *two-point crossover*. Para implementar este operador, são definidos dois pontos de corte num pai. Copiam-se os genes contidos entre os dois pontos de corte para as mesmas posições de um cromossoma filho, e depois para preencher os genes vazios do filho, percorre-se o cromossoma do segundo

pai, e vai-se copiando o valor contido nos seus genes para o filho, respeitando a ordem e consistência.

No bloco aplica operador de mutação, para cada filho que é gerado, de acordo com uma dada probabilidade  $P_m$ , vai ser aplicado o operador de mutação *shift mutation*. Este bloco recebe como parâmetros de entrada a nova população e a probabilidade de mutação, e retorna a nova população já com a mutação introduzida. Para implementar o operador *shift mutation*, são gerados dois valores aleatórios e diferentes, entre 1 e  $n$ , com  $n$  igual ao número total de operações a processar. Um dos valores corresponde a um gene, e o outro a uma posição. Posteriormente elimina-se do cromossoma o gene escolhido e insere-se na posição gerada.

No desenvolvimento do algoritmo genético foram usados dois ficheiros de apoio. No primeiro registaram-se todas as soluções avaliadas e no segundo em cada geração do AG foi registada a melhor solução da população e respectivo valor de *fitness*. Desta forma se for necessário terminar o processamento mais cedo podemos saber qual a melhor solução encontrada até ao momento.

No anexo 2 podemos consultar a implementação, em R, deste algoritmo.

## 4. Procura Tabu

### 4.1. Introdução

Este é um método de procura local baseado no conceito de vizinhança. A Procura Tabu (PT) parte de uma solução e através de movimentos sucessivos vai-se deslocando para uma solução na sua vizinhança. [Colorni et al., 1998]. Este método pretende explorar o espaço de soluções admissíveis através de uma sequência de movimentos. O movimento de uma solução para outra é feito avaliando as soluções vizinhas e escolhendo a melhor disponível. Alguns movimentos são classificados como tabu (proibidos) por conduzirem a óptimos locais ou a soluções já antes visitadas. Esses movimentos são guardados numa estrutura chamada lista tabu. A lista tabu é o historial de movimentos já efectuados durante a pesquisa, e com a qual se pretende guiar a procura para uma região do espaço ainda não explorada [Jones e Rabelo, 1998].

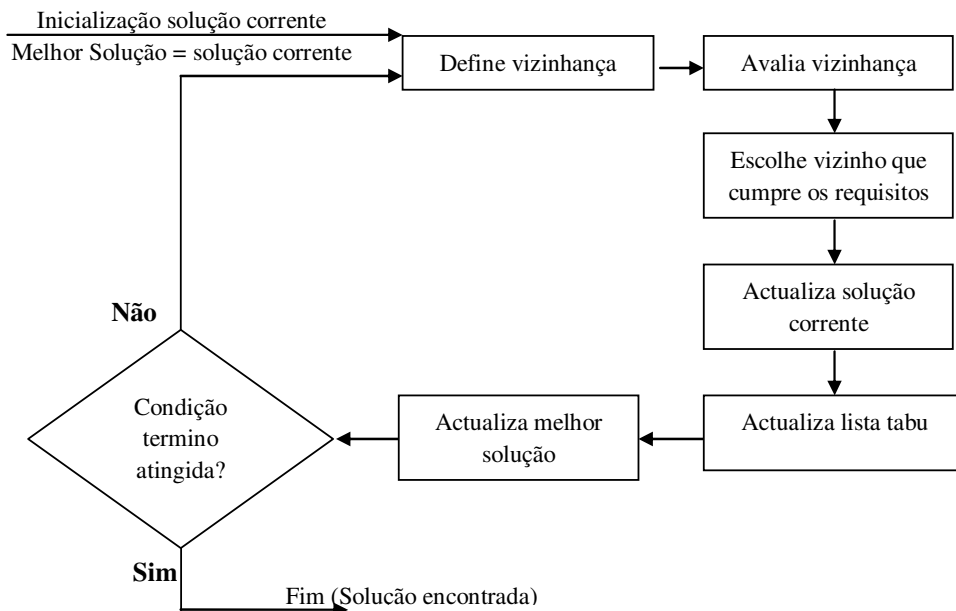


Figura 35. Esquema de funcionamento do método da Procura Tabu (PT).

O método da PT começa com uma solução inicial que guarda como solução corrente e como melhor encontrada até então. Os vizinhos da solução corrente são depois

encontrados através de uma estrutura de vizinhança, sendo estas as soluções candidatas para a próxima iteração do algoritmo. Essas soluções são avaliadas por uma função de avaliação (*fitness*), e a melhor solução que não seja tabu ou que o sendo satisfaça o critério de aspiração, é seleccionada como solução corrente. O critério de aspiração é usado para ultrapassar as restrições da lista tabu impedindo o método de parar cedo demais. A solução corrente é comparada com a melhor encontrada até então e se for melhor é guardada como a melhor encontrada até então. O movimento que levou a esta solução é guardado na lista tabu. Este processo repete-se até que seja satisfeito o critério de paragem. O critério de paragem pode ser definido, por exemplo, como o número máximo de soluções a testar, o número máximo de iterações e executar, entre outros. Na Procura Tabu a qualidade da solução inicial influencia a solução final de tal forma que quanto melhor for a solução inicial melhor será a solução final encontrada [Geyik e Cedimoglu, 2004].

## **4.2. Revisão da Literatura**

A PT é um método de aproximação que segue uma abordagem iterativa e que tem permitido obter bons resultados quando aplicado a problemas de *job-shop scheduling*. Este é um método simples que de forma inteligente guia a procura para longe de soluções que aparentam duplicar ou repetir soluções previamente visitadas [Jain et Meeran, 1999]. Geralmente conseguem-se bons resultados mas o tempo de computação requerido depende muito da complexidade do problema [Mahmood e Hyasat, 2002]. Na Procura Tabu o problema do *job-shop* é frequentemente representado através de um grafo disjuntivo. Na Figura 37 podemos ver a representação do grafo disjuntivo para o problema do tipo 4x3 descrito na Figura 36 e retirado de [Geyik e Cedimoglu, 2004].

| Jobs                          | Processos |    |    |
|-------------------------------|-----------|----|----|
|                               | 1         | 2  | 3  |
| (a) Caminhos de Processamento |           |    |    |
| 1                             | m1        | m2 | m3 |
| 2                             | m2        | m1 | m3 |
| 3                             | m3        | m2 | m1 |
| 4                             | m2        | m3 | m1 |
| (b) Tempos de Processamento   |           |    |    |
| 1                             | 4         | 3  | 2  |
| 2                             | 1         | 4  | 4  |
| 3                             | 3         | 2  | 2  |
| 4                             | 3         | 3  | 1  |

Figura 36. (a) Caminhos de Processamento e (b) Tempos de Processamento de um problema do tipo 4 jobs x 3 máquinas [Geyik e Cedimoglu, 2004].

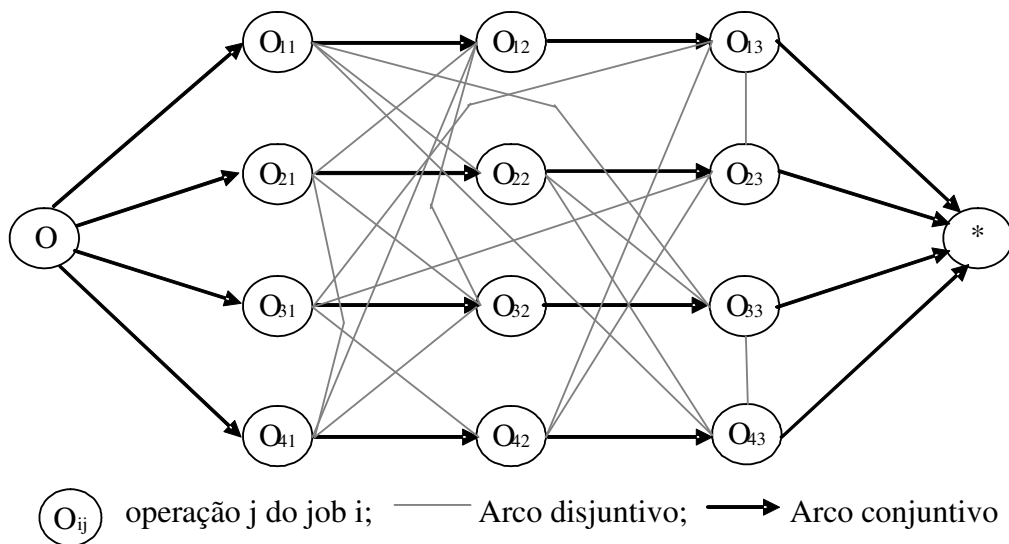


Figura 37. Representação do grafo disjuntivo para o problema 4x3 da figura 2 [Geyik e Cedimoglu, 2004].

Nesta representação os arcos conjuntivos satisfazem as restrições de precedência das operações de um *job*, enquanto os arcos disjuntivos satisfazem as restrições de capacidade das máquinas. Os arcos disjuntivos são unidireccionais.

Para identificar as soluções possíveis, começa por se fixar a direcção de cada arco disjuntivo, originando soluções parciais. Para que estas soluções sejam consideradas soluções possíveis é necessário que depois de fixados os arcos disjuntivos o grafo disjuntivo resultante seja acíclico.



#### 4.2.1. Representação das soluções

Na codificação das soluções candidatas, a Procura Tabu pode usar o mesmo tipo de estrutura que os Algoritmos Genéticos (AGs). [Ponnambalam et al., 2000] propõem um algoritmo de procura tabu onde usam uma estrutura, para codificação das soluções, semelhante à usada por [Park et al., 2003], [Wang e Zheng, 2002], [Tsuji-mura et al., 2001] em AGs. A representação das soluções é feita através de um *array* de posições. Cada uma destas posições contém informação sobre o *job* que vai ser processado. Como se pode ver na Figura 40, o processamento inicia-se com a primeira operação do *job* 3, seguidamente processa-se a primeira operação do *job* 2, depois segue-se a segunda operação do *job* 2, depois a primeira operação do *job* 1, e por aí fora.

$$\left[ \begin{array}{cccccccc} 3 & 2 & 2 & 1 & 1 & 2 & 3 & 1 & 3 \end{array} \right]$$

Figura 40. Representação das soluções para um problema do tipo 3x3.

#### 4.2.2. Obtenção da solução inicial

A solução inicial pode ser obtida por aplicação de uma das regras de atribuição de prioridades, por métodos aleatórios, entre outros.

No caso de [Ponnambalam et al., 2000] e [Zhang et al., 2007], os autores optaram por gerar a solução inicial de forma aleatória. Já [Geyik e Cedimoglu, 2004] optaram pela aplicação da regra de atribuição de prioridade SPT. Esta regra atribui maior prioridade às operações que demoram menos tempo a ser processadas, isto é, sequencia as operações por ordem inversa do tempo de processamento. [Srich et al., 2004] também usam regras de atribuição de prioridades para definir a solução inicial.

#### 4.2.3. Função de Avaliação

As funções de avaliação usadas pela PT são comuns aos outros métodos utilizados para resolver o problema de *scheduling* de operações. As mais frequentes são o tempo total requerido para processar todas as operações que compõem os *jobs* do problema (*makespan*), tempo médio para completar uma ordem e tempo médio de atraso [Geyik e Cedimoglu, 2004].

Por exemplo, [Ponnambalam et al., 2000], [Nowicki e Smutnicki, 2005] e [Zhang et al., 2007], usaram o *makespan* como função de avaliação enquanto [Schrigh et al., 2004] usaram o tempo de atraso em relação às datas de entrega.

#### 4.2.4. Definição da Vizinhança

Na construção da vizinhança, assume-se que cada solução  $\mu \in X$  ( $X$  igual ao universo de soluções possíveis) tem associado um conjunto de vizinhos,  $N(\mu) \subset X$ , chamado vizinhança de  $\mu$ . Cada solução  $\mu' \in N(\mu)$  pode ser obtida directamente de  $\mu$ , através de uma operação chamada movimento [Reeves, 1993]. Isto significa que cada solução vizinha é encontrada a partir de um pequeno “movimento” efectuado noutra solução [Geyik e Cedimoglu, 2004].

Estes movimentos podem ser efectuados através de métodos como *adjacente pairwise interchange*, em que se seleccionam dois elementos adjacentes, da solução corrente, e se trocam de posição. A utilização deste método permite criar uma vizinhança constituída por  $n-1$  vizinhos. Ver por exemplo [Ponnambalam et al., 2000].

Outro método também frequentemente usado para obter soluções vizinhas é o *pairwise exchange* (ou *Swap*), tal como descrito por [Reeves, 1993]. Através deste método cada operação  $i$  da solução corrente vai trocar de posição com as  $n-i$  operações seguintes, o que para um problema do tipo 3x3 iria resultar numa estrutura com 36 vizinhos.

Para [Schrigh et al., 2004] e [Nowicki e Smutnicki, 2005] a vizinhança de uma solução é encontrada trocando as posições das operações adjacentes, mas apenas para a máquina onde se verifica o caminho crítico.

#### 4.2.5. Lista Tabu

Cada vez que uma solução vizinha passa a ser solução corrente, o movimento que levou a essa solução é registado na tabela de movimentos tabu. Esse movimento vai ser tabu durante um número de iterações prédefinido.

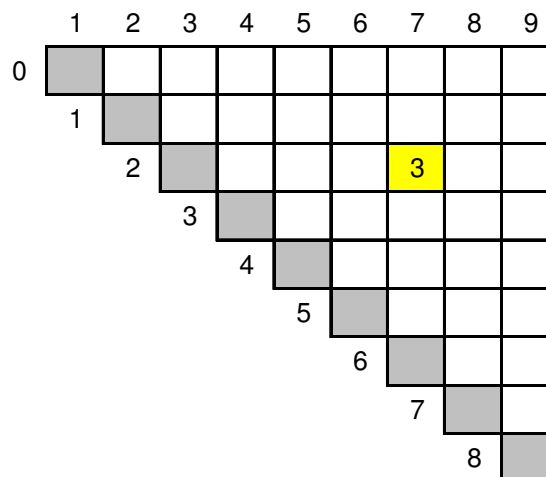


Figura 41. Exemplo de estrutura da lista tabu para um problema do tipo 3x3 [Ponnambalam et al., 2000].

[Ponnambalam et al., 2000] definiram uma lista tabu semelhante à representada na Figura 41. Aqui, cada movimento é considerado tabu durante  $\sqrt{n}$  iterações (com  $n$  igual ao produto do número de *jobs* pelo número de máquinas). Se o vizinho escolhido for aquele que resulta do movimento (2,7) (troca da operação existente na posição 2 com a operação existente na posição 7), a solução resultante deste movimento passa a ser considerada solução corrente na iteração seguinte do algoritmo. Depois, teríamos de actualizar a lista tabu de acordo com a figura em cima. Na célula correspondente às coordenadas (2,7) é registado o número de iterações em que este movimento é proibido (que para este exemplo é  $\sqrt{9} = 3$ ). Em cada iteração do algoritmo esse valor vai ser diminuído em uma unidade.

Associado a cada movimento temos sempre o valor do movimento que indica o ganho ou a perda em relação ao valor da solução corrente. O melhor movimento que não seja tabu ou que satisfaça o critério de aspiração é seleccionado para solução corrente, isto de acordo com a vizinhança definida, com o valor de cada movimento e com a lista tabu. O uso de uma estrutura para guardar a lista de movimentos tabu tenta impedir o método de convergir para soluções óptimas locais, ou de entrar em ciclo por considerar soluções já anteriormente visitadas.

[Geyik e Cedimoglu, 2004] usaram uma estrutura diferente em que a lista é um *array* de posições, de tamanho fixo. Sempre que um movimento passa a ser considerado tabu ele é inserido no início da lista de movimentos tabu e os movimentos que já faziam parte da

lista descem uma posição. O número de iterações que cada movimento é considerado tabu corresponde ao tamanho da lista.

#### **4.2.6. Critério de Aspiração**

O critério de aspiração serve para ultrapassar as restrições da lista tabu. Se todos os movimentos que originam a vizinhança de uma solução são considerados tabu, então podemos usar um critério de aspiração para decidir qual o vizinho a escolher para solução corrente.

[Ponnambalam et al., 2000] utilizaram, sucessivamente quando necessário, dois critérios de aspiração: o movimento que originaria uma solução melhor do que a corrente e o movimento menos tabu. De forma semelhante, em [Mahmood e Hyasat, 2002], [Geyik e Cedimoglu, 2004] e [Srich et al., 2004], quando o movimento leva a uma solução melhor do que a melhor encontrada até então, esse movimento é seleccionado, mesmo que ele seja tabu e até mesmo o mais tabu.

#### **4.2.7. Parâmetros de Controlo**

Para definir quando é que o algoritmo de Procura Tabu vai terminar o processo de busca é necessário definir um parâmetro de controlo, para usar como critério de paragem.

[Ponnambalam et al., 2000] definiram como condição de paragem do algoritmo o número máximo de iterações, que eles especificaram como 100.

[Mahmood e Hyasat, 2002] estipularam que o algoritmo iria efectuar entre 1000 e 20000 movimentos (sendo que o número de movimentos depende da dimensão do problema).

[Geyik e Cedimoglu, 2004] definiram como critério de paragem o número de iterações consecutivas sem melhoria da solução, que estipularam ser 2000 desde que ou pudessem gerar vizinhos ou a solução encontrada fosse possível.

[Srich et al., 2004] definiram como condição de paragem da procura tabu a execução de 250 iterações do algoritmo.

### **4.3. Metodologia Usada**

Nesta secção começamos por analisar as componentes de um método de procura tabu e os parâmetros de controlo usados, explicando a razão das escolhas. Seguidamente explicamos a implementação dos algoritmos que implementam o método da procura tabu.

#### **4.3.1. Concepção**

Para implementar os algoritmos de Procura Tabu, foi usado como referência o trabalho desenvolvido por [Ponnambalam et al., 2000]. A nossa decisão baseou-se no facto deste trabalho usar estruturas semelhantes à de um AG e, de acordo com os autores, o algoritmo apresentar bons resultados.

Os dois algoritmos de Procura Tabu que implementados diferem apenas na solução inicial. No algoritmo PT1 esta solução é gerada aleatoriamente tal como em [Ponnambalam et al., 2000], enquanto no PT2 a solução inicial usada é a melhor de 100 soluções geradas aleatoriamente. O PT2 surge porque de acordo com alguns autores a qualidade da solução final encontrada pela PT depende da qualidade da solução inicial.

Para representar as soluções do problema foi usada uma estrutura semelhante à de [Ponnambalam et al., 2000], e que já tinha sido usada no AG. Cada solução do problema é representada através de um *array* de posições. Cada uma destas posições contém informação sobre o *job* que vai ser processado.

Como função de avaliação foi usada a minimização do tempo total necessário para processar todas as operações dos *jobs* (*makespan*). Esta função de avaliação será usada em todos os algoritmos a desenvolver.

Tal como em [Ponnambalam et al., 2000] na construção da estrutura de vizinhança, das soluções correntes, usamos o operador *adjacente pairwise interchange*. Através deste operador para cada possível solução do problema vão-se trocar os valores de todas as suas posições adjacentes. Assim, para cada solução vamos construir uma vizinhança formada por  $n-1$  vizinhos.

A lista tabu guarda os movimentos que originaram as soluções correntes, impedindo desta forma que o algoritmo entre em ciclo, visitando soluções já antes visitadas e

permitindo assim diversificar o espaço de procura. O número de iterações em que estes movimentos vão ser considerados tabu, depende da dimensão do problema. Também aqui foi usado o mesmo critério que [Ponnambalam et al., 2000], número de iterações tabu= $\sqrt{n}$  (com  $n$  igual a *jobs* x máquinas).

Foram usados dois critérios de aspiração: o de escolher o movimento que originaria uma solução melhor do que a corrente e o de escolher o movimento com menor número de iterações tabu, isto nos casos em que todos os vizinhos sejam originados a partir de movimentos tabu.

### **4.3.2. Implementação**

Os algoritmos PT1 e PT2 são compostos por duas funções principais a que vou chamar blocos. O bloco principal é aquele que descreve a estrutura de funcionamento genérica dos nossos algoritmos de Procura Tabu. Este bloco recebe como parâmetros de entrada o número máximo de soluções a testar, a tabela de tempos e a tabela de precedências. Ele retorna a melhor solução encontrada.

A estrutura base de funcionamento deste bloco é a descrita no quadro em baixo.

- Gera solução corrente
- Bloco: Avalia solução
- Atribui solução corrente à melhor encontrada
- Ler tabela de precedências das operações dos *jobs*
- Ler tabela de tempos de processamento das operações nas máquinas
  - Até atingir  $\text{maxSolucoes}=100000$  fazer:
    - Gera  $n-1$  vizinhos da solução corrente
    - Bloco: Avalia solução (avalia os vizinhos)
    - Actualiza lista\_tabu com o movimento tabu
    - Escolhe melhor vizinho não tabu ou não existindo aquele que respeita critério de aspiração
    - Actualiza solução corrente com melhor vizinho
    - Se a solução corrente é melhor que a melhor encontrada  
Então actualiza melhor encontrada
    - Actualiza lista\_tabu com movimento tabu
- Retorna a melhor solução encontrada e respectivo valor de *fitness*

Figura 42. Algoritmo de funcionamento do bloco principal dos algoritmos da PT.

Neste algoritmo, o primeiro passo é gerar uma solução inicial, que vai ser atribuída à solução corrente, para depois ser avaliada, através de uma função de avaliação. Após esta avaliação, a solução corrente é atribuída à melhor encontrada. Seguidamente são lidas as tabelas de precedências e de tempos, e criadas as estruturas de dados que irão guardar os vizinhos de cada solução corrente e os movimentos tabu.

Para cada solução corrente vai ser construída a sua vizinhança, que depois será avaliada através do bloco: Avalia solução. Após a avaliação dos vizinhos, eles são colocados por ordem crescente do valor do movimento. Antes de escolher o melhor vizinho, é actualizada a lista que contém o número de iterações durante o qual cada movimento é considerado tabu. Seguidamente é feita a escolha do melhor vizinho não tabu ou, caso não exista nenhum, escolha o melhor vizinho que respeite o critério de aspiração. Depois de escolher o vizinho é actualizada a solução corrente com o vizinho escolhido, e a lista tabu para o movimento que originou o vizinho (com o valor  $\sqrt{n}$ ). No final de cada iteração é comparado o valor da solução corrente com o valor da melhor solução

encontrada. Se a solução corrente for melhor então guarda-se como melhor solução encontrada. Desta forma garantimos que de iteração para iteração não perdemos a melhor solução encontrada. Este processo repete-se até atingir o número máximo de soluções a testar. No final da execução dos algoritmos PT1 e PT2 eles vão retornar a melhor solução encontrada e o correspondente valor de *fitness*.

Bloco: Avalia solução, para cada solução que é testada vai-se verificar e calcular o tempo necessário para processar todas as actividades. Esta função é igual à que foi usada para avaliar as soluções no algoritmo genético, pelo que podemos ver a descrição detalhada do seu funcionamento na Secção 3.3.2 deste trabalho.

À semelhança do que se tinha efectuado no algoritmo genético também nos algoritmos da procura tabu foram usados dois ficheiros de apoio. No primeiro foram registadas todas as soluções avaliadas. No segundo, em cada iteração, são registadas as soluções correntes e respectivo valor de *fitness*. Desta forma se for necessário terminar o processamento mais cedo, podemos sempre saber qual a melhor solução encontrada até ao momento.

No anexo 3 e 4 podemos ver detalhes da implementação, em R, dos algoritmos PT1 e PT2 respectivamente.

## 5. Algoritmos Genéticos Híbridos

### 5.1. Introdução

A combinação de métodos de procura local descendente, *simulated annealing*, procura tabu e algoritmos evolutivos têm vindo a originar algoritmos de procura muito poderosos [Talbi, 2002]. À combinação destes métodos chamamos de métodos híbridos. Este tipo de métodos têm vindo a ser objecto de grande interesse por parte de muitos autores, especialmente na aplicação a problemas de optimização combinatória.

Alguns autores como é o caso de [Wang e Zheng, 2002], [Wang e Wu, 2004], [Gonçalves et al., 2005], [Ombukiri e Ventresca, 2004], entre outros, têm estudado o problema usando esta nova abordagem. Alguns trabalhos analisados têm mostrado que quando se combinam algoritmos genéticos com um método de procura local, se obtêm melhores resultados do que usando esses mesmos algoritmos de forma isolada.

Os algoritmos genéticos através dos seus operadores, especialmente o de mutação, conseguem explorar todo o espaço de soluções admissíveis, no entanto apresentam algumas dificuldades na obtenção dos óptimos locais. Por esta razão, combinam-se com métodos de procura local, como *simulated annealing*, procura tabu, entre outros, pois estes são bons a convergir para óptimos locais, apesar de tipicamente não terem um bom comportamento na procura global [Wang e Wu, 2004].

O objectivo a que nos propomos é o de testar 2 algoritmos genéticos híbridos capazes de se ajudarem para obter uma boa solução num espaço de tempo aceitável.

### 5.2. Revisão da literatura

Na Figura 43, podemos ver a taxionomia proposta por [Talbi, 2002] no desenho de meta heurísticas híbridas.

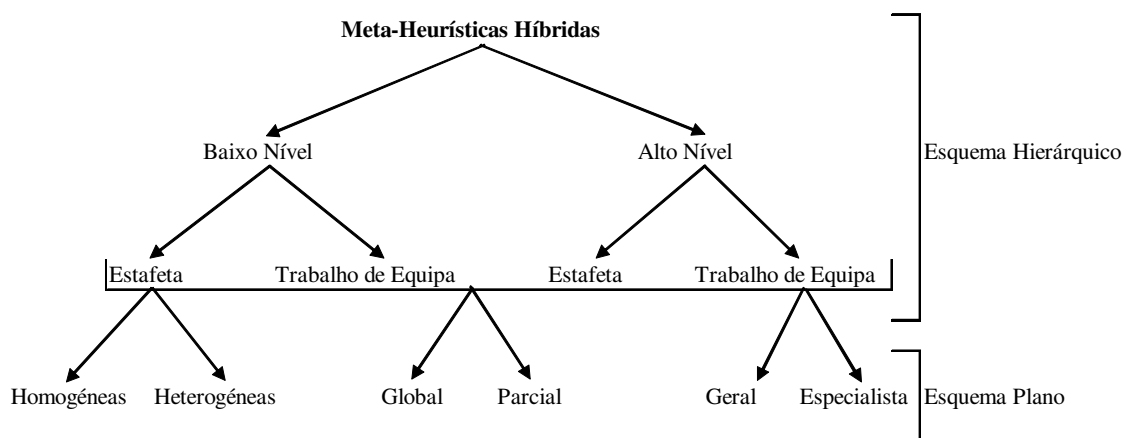


Figura 43. Classificação de meta heurísticas híbridas de acordo com [Talbi, 2002].

De acordo com [Talbi, 2002], o objectivo do seu trabalho foi fornecer um mecanismo que permitisse comparar os algoritmos híbridos de forma qualitativa. A taxionomia proposta é uma combinação de dois esquemas: um hierárquico e um plano.

### 5.2.1. Esquema Hierárquico

No esquema hierárquico, num primeiro nível podemos distinguir os algoritmos híbridos de baixo nível e os de alto nível. Os de baixo nível são aqueles onde se substitui uma função de um método por outro método. Os de alto nível são aqueles que usam os diferentes métodos de forma independente. No segundo nível temos os algoritmos tipo estafeta e o trabalho em equipa. No tipo estafeta usam-se diferentes meta heurísticas, uma a seguir a outra, em que o *output* de uma é o *input* da seguinte. Na híbridização tipo trabalho de equipa, podemos ver modelos de optimização cooperativos, em que temos vários agentes a cooperar em paralelo. Neste tipo de híbridização, cada agente tem a seu cargo um espaço de soluções para efectuar a procura.

Uma grande parte dos algoritmos híbridos propostos na literatura são algoritmos híbridos de baixo nível do tipo estafeta, isto porque substituem uma função do algoritmo genético (tipicamente o operador de mutação) por um método de procura local, e porque o *output* de um método é usado como *input* de outro.

Por exemplo, [Wang e Zheng, 2002] propõem um algoritmo genético híbrido que alia a propriedade dos algoritmos genéticos de exploração de todo o espaço de soluções admissíveis, com a propriedade do método *simulated annealing* de refinar as soluções

encontradas em cada iteração do algoritmo. O método de procura local, *Simulated Annealing*, foi implementado através do operador de mutação no algoritmo genético. Também [Ombuki e Ventresca, 2004] propõem um algoritmo genético híbrido deste tipo para resolução de problemas de *scheduling*. Estes autores analisam o comportamento de um algoritmo genético híbrido e de três algoritmos genéticos: um simples que não usa qualquer operador de mutação, um que usa um operador de mutação e um outro que usa um operador de mutação especial onde foi implementado um mecanismo de procura local.

### **5.2.2. Esquema Plano**

No esquema plano podemos dividir as meta heurísticas em Homogéneas ou Heterogéneas, em Globais ou Parciais, e em Especialistas ou Gerais.

Os algoritmos homogéneos são aqueles cujos algoritmos base usam as mesmas meta-heurísticas, geralmente fazendo variar os seus parâmetros. Os algoritmos heterogéneos têm como base algoritmos que usam diferentes meta-heurísticas. Por exemplo algoritmos que usam a procura tabu para gerar soluções que posteriormente serão usadas numa população inicial para um algoritmo genético.

Nos algoritmos híbridos Globais, todos os algoritmos fazem a procura em todo o espaço de soluções. O objectivo é explorar de forma mais exaustiva todo o espaço de soluções admissíveis. Nos híbridos Parciais, o problema é decomposto em sub-problemas cada um com o seu próprio espaço de procura. Cada algoritmo que constitui o algoritmo híbrido fica responsável por efectuar a procura em cada um destes “sub-espacos”.

Os algoritmos híbridos Gerais são formados por algoritmos com o mesmo objectivo, o de encontrar a solução óptima do problema. Os híbridos especialistas são os menos usuais uma vez que os algoritmos que os compõem têm objectivos diferentes.

Uma grande parte dos algoritmos híbridos propostos na literatura são algoritmos híbridos heterogéneos, globais e gerais, pois usam diferentes meta-heurísticas para fazer a procura em todo o espaço de soluções, com o mesmo objectivo.

Como exemplos temos os algoritmos híbridos propostos por [Wang e Zheng, 2002] e [Ombuki e Ventresca, 2004].

### **5.3. Metodologia usada**

Nesta secção analisamos o tipo de algoritmos híbridos implementados e a motivação dessa implementação. De seguida descrevemos a sua implementação.

#### **5.3.1. Concepção**

Foram implementadas duas variantes de algoritmos híbridos, o Algoritmo Genético Híbrido 1 (AGH1) e o Algoritmo Genético Híbrido 2 (AGH2). Estes dois algoritmos surgem da combinação entre Algoritmos Genéticos (AGs) e Procura Tabu (PT).

Neste trabalho pretendemos combinar os dois métodos puros fazendo o mínimo de alterações. No entanto não podemos deixar de ter em atenção o que outros autores estão a desenvolver nesta área, para criar algoritmos híbridos com características semelhantes. Os métodos híbridos propostos por exemplo por [Wang e Zheng, 2002] e [Ombuki e Ventresca, 2004] usam como base um algoritmo genético cujo operador de mutação é substituído por um método de procura local. Enquanto [Liaw, 2000] e [Gonçalves e al., 2004] em cada nova geração do algoritmo genético aos novos indivíduos da população aplicam um método de procura local.

Neste trabalho optámos por seguir a abordagem de [Liaw, 2000] e [Gonçalves e al., 2004], na implementação do algoritmo AGH1. Este método híbrido é um algoritmo genético semelhante ao descrito anteriormente na Secção 3.3.2., que à melhor solução encontrada em cada nova geração, vai aplicar o método da Procura Tabu cuja implementação se descreveu na Secção 4.3.2. Se após executar a Procura Tabu a solução encontrada for melhor do que a solução inicial, então substituímos na população do algoritmo genético a solução inicial pela nova encontrada.

A motivação para a implementação do AGH2 surge devido às características de cada método puro. O AG é capaz de encontrar a zona do espaço de procura onde se encontra o óptimo mas pode não conseguir encontrá-lo. A PT é eficaz para encontrar óptimos locais. Além disso, [Geyik et Cedimoglu, 2004] referem que a qualidade da solução inicial na Procura Tabu influencia a qualidade da solução final obtida por este método. Daí a implementação do algoritmo AGH2. Inicialmente executamos o nosso AG puro para encontrar uma boa solução. Depois, a melhor solução encontrada pelo AG irá ser

usada como solução inicial pelo método da PT. A PT é usada para procurar uma solução melhor na vizinhança da solução encontrada pelo AG.

De acordo com a taxonomia proposta por [Talbi, 2002], podemos dizer que implementamos dois algoritmos do tipo Estafeta pois o *output* de um método é usado como *input* de outro, sendo um de Baixo Nível outro de Alto Nível. O AGH1 é de Baixo Nível uma vez que uma das funções do algoritmo genético é substituída pela Procura Tabu, enquanto o AGH2 é de Alto Nível por ambos os métodos AG e PT serem executados de forma sequencial.

### **5.3.2. Implementação**

Como referido anteriormente, foram desenvolvidos dois algoritmos híbridos. O AGH1 é um algoritmo genético semelhante ao descrito na Secção 3.3.2, que à melhor solução encontrada em cada geração vai aplicar o método da Procura Tabu e cuja implementação descrevemos na Secção 4.3.2. Em cada geração a procura tabu vai avaliar 100 soluções. No total o algoritmo AGH1 vai avaliar 100000 soluções.

Ver Figura 44, explicativa da estrutura de funcionamento deste algoritmo.

- Ler tabela de precedências das operações dos *jobs*
- Ler tempos de processamento das operações nas máquinas
- Bloco: Gera população inicial
- Bloco: Avalia cromossomas da população gerada
- Até atingir maxSolucoes=100000 fazer:
  - Aplica Operador de Selecção: Selecciona os melhores  $x=10\%$  indivíduos da população corrente e passa-os para a nova população
  - Bloco: Aplica Operador de Cruzamento
  - Bloco: Aplica Operador de Mutação
  - Bloco: Avalia cromossomas da população gerada
  - Para a melhor solução encontrada na geração, fazer:
    - **Procura Tabu:**
      - Atribui solução corrente à melhor encontrada
      - Até atingir o número maxSolPT=100 fazer:
        - Gera os  $n-1$  vizinhos da solução corrente
        - Bloco: Avalia solução (avalia os vizinhos)
        - Actualiza movimentos da lista tabu (diminuindo-lhes uma unidade)
        - Escolhe melhor vizinho não tabu ou não existindo aquele que respeita o critério de aspiração
        - Actualiza solução corrente com o melhor vizinho
        - Se a solução corrente é melhor que a melhor encontrada
          - Então actualiza a melhor encontrada
        - Actualiza lista tabu com o movimento tabu
        - Retorna a melhor solução encontrada e respectivo valor de *fitness*
      - Se a melhor solução encontrada pela PT é melhor que a melhor solução encontrada na geração corrente do AG então:
        - Substitui melhor solução do AG pela solução da PT
    - Retorna a melhor solução e o respectivo valor de *fitness*

Figura 44. Estrutura de funcionamento do algoritmo híbrido 1 (AGH1).

O algoritmo híbrido AGH2 começa por executar o AG descrito na Secção 3.3.2, e usa a melhor solução encontrada por ele como solução inicial da procura tabu. O algoritmo de procura tabu usado foi o explicado na Secção 4.3.2. O algoritmo AGH2 avalia um total

de 100000 soluções, metade avaliadas pelo algoritmo AG e a outra metade pelo algoritmo PT.

Na Figura 45 podemos ver a estrutura de funcionamento do algoritmo AGH2.

- Ler tabela de precedências das operações dos *jobs*
- Ler tempos de processamento das operações nas máquinas
- Bloco: Gera população inicial
- Bloco: Avalia cromossomas da população gerada
- Até atingir maxSolucoes=50000 fazer:
  - Selecciona os melhores  $x=10\%$  indivíduos da população corrente e passa-os para a nova população
  - Bloco: Aplica Operador de Cruzamento
  - Bloco: Aplica Operador de Mutação
  - Bloco: Avalia cromossomas da população gerada
- Retorna a melhor solução e o respectivo valor de *fitness*
- Para a melhor solução encontrada pelo Algoritmo Genético, fazer:
  - **Procura Tabu:**
    - Atribui solução corrente à melhor encontrada
    - Até atingir maxSolucoes=50000 fazer:
      - Gera  $n-1$  vizinhos da solução corrente
      - Bloco: Avalia solução (avalia os vizinhos)
      - Actualiza lista\_tabu com o movimento tabu
      - Escolhe melhor vizinho não tabu ou não existindo aquele que respeita critério de aspiração
      - Actualiza solução corrente com melhor vizinho
      - Se a solução corrente é melhor que a melhor encontrada  
Então actualiza melhor encontrada
      - Actualiza lista\_tabu com movimento tabu
    - Retorna a melhor solução encontrada e respectivo valor de *fitness*

Figura 45. Estrutura de funcionamento do algoritmo híbrido 2 (AGH2).

A descrição detalhada do funcionamento do algoritmo genético e da procura tabu que compõem os algoritmos híbridos pode ser vista em detalhe nas Secções mencionadas.

Para estes algoritmos tal como no algoritmo genético e na procura tabu foram usados dois ficheiros de apoio. No primeiro guardam-se todas as soluções avaliadas, e no segundo registam-se as soluções com melhor valor de *fitness* na geração/iteração e respectivo valor de *fitness*, bem como o tempo decorrido até ao momento da sua obtenção. Com recurso a estes ficheiros, se necessitarmos parar o processamento mais cedo, saberemos qual a melhor solução encontrada até ao momento. Para além disso estes dados permitem analisar a evolução dos resultados de forma mais detalhada.

## 6. Resultados Experimentais

Neste capítulo apresentamos os resultados da avaliação empírica dos métodos descritos nos capítulos 3, 4 e 5.

Na Secção 6.1, descrevemos os problemas usados para testar os algoritmos, na Secção 6.2 descrevemos as experiências computacionais realizadas, e por fim, na Secção 6.3 analisamos os resultados obtidos. Nesta última secção, começamos por fazer uma breve comparação da performance dos 5 algoritmos, para depois comparar o desempenho dos dois algoritmos de procura tabu, e, em seguida, analisar os resultados obtidos com os diferentes parâmetros de controlo do algoritmo genético. De seguida fazemos a análise e comparação dos dois algoritmos híbridos, para no final fazer uma comparação mais detalhada dos cinco algoritmos propostos.

### 6.1. Descrição dos problemas usados

Os métodos foram testados em 53 problemas do tipo  $n \times m$  usados frequentemente em trabalhos de investigação (*benchmark problems*).

Na figura 46 podemos ver a representação do problema  $3 \times 3$  usado.

|   |   | Máquinas |   |   |
|---|---|----------|---|---|
| J | 1 | 3        | 1 | 2 |
| O | 2 | 2        | 3 | 1 |
| b | 3 | 2        | 1 | 3 |
| s |   |          |   |   |

|   |   | Tempos |   |    |
|---|---|--------|---|----|
| J | 1 | 1      | 3 | 6  |
| O | 2 | 8      | 5 | 10 |
| b | 3 | 3      | 8 | 1  |
| s |   |        |   |    |

Figura 46. Tabelas de precedências e tempos de processamento para um problema do tipo  $3 \times 3$ .

A primeira tabela dá-nos as precedências das operações de cada *job* nas máquinas. Como referido antes através destas tabelas podemos dizer que o *job* 1 vai processar a primeira operação na máquina 3, a segunda operação na máquina 1 e a terceira na máquina 2. Na segunda tabela temos os tempos de processamento, pelo que podemos verificar que para o primeiro *job* a primeira operação a processar vai demorar uma unidade de tempo, a segunda demora três unidades de tempo e a terceira demora seis.

Os restantes 52 problemas usados foram os ft6x6 e ft10x10 de Fisher e Thompson, e os 10 problemas dos tipos ta4x4, ta5x5, ta7x7, ta10x10 e ta15x15 de Taillard. Mais detalhes sobre este tipo de problemas são dados em [Jain e Meeran, 1999].

Para efectuar a análise dos resultados obtidos pelos algoritmos, estes problemas foram divididos em problemas “pequenos”, “médios” e de “grande” dimensão. Nos problemas de pequena dimensão incluímos os problemas do tipo 3x3 e 4x4. Nos problemas de média dimensão incluímos os problemas do tipo 5x5, 6x6 e 7x7. Nos problemas de grande dimensão incluímos os problemas do tipo 10x10 e 15x15.

## **6.2. Descrição das experiências computacionais**

O que nos propomos fazer é comparar o resultado obtido pelos 5 algoritmos implementados, mas quando executados em igualdade de circunstâncias. Portanto, o nosso objectivo é comparar a eficácia dos algoritmos na forma como exploram o espaço de soluções e não a eficiência da implementação dos algoritmos. Assim, garantimos que cada um deles avalia o mesmo número de soluções, em vez de analisarmos o tempo de execução.

O primeiro algoritmo a analisar foi o que implementa a procura tabu simples, e que se designou por PT1. Este algoritmo como referido anteriormente vai avaliar 100000 soluções, sendo a solução inicial gerada aleatoriamente.

O segundo algoritmo também implementa uma procura tabu simples, a que designamos de PT2. A diferença entre este algoritmo e o algoritmo anterior é que a sua solução inicial é a melhor solução de 100 geradas aleatoriamente.

O terceiro é um algoritmo genético simples, a que chamamos AG. Este algoritmo será executado com quatro variações de parâmetros de controlo, tal como explicado na Secção 3.3.1. Na Figura 47 podemos ver o detalhe desses parâmetros de controlo.

|         | Nº Soluções Testadas | Nº Repetições | popSize AG | Pm AG | Pc AG |
|---------|----------------------|---------------|------------|-------|-------|
| Param 1 | 100000               | 10            | 50         | 0.01  | 1     |
| Param 2 | 100000               | 10            | 100        | 0.01  | 1     |
| Param 3 | 100000               | 10            | 50         | 0.1   | 1     |
| Param 4 | 100000               | 10            | 100        | 0.1   | 1     |

Figura 47. Número de soluções testadas, número de repetições do algoritmo, tamanho da população, probabilidade de mutação e probabilidade de cruzamento usados em cada teste efectuado para cada problema.

O quarto algoritmo a analisar é um algoritmo híbrido, que designamos de AGH1. Este algoritmo híbrido usa o algoritmo genético AG como base e avalia também 100000 soluções. Neste AG híbrido, em cada geração do algoritmo genético, à melhor solução encontrada aplica o algoritmo PT1 que por sua vez avalia 100 soluções, conforme descrito na Secção 5.3.2.

O quinto algoritmo a executar é também um algoritmo híbrido, que designamos por AGH2. Este algoritmo híbrido começa por executar o algoritmo AG a avaliar 50000 soluções (metade do total das soluções a avaliar), em que a melhor solução encontrada pelo AG é depois usada como solução inicial do algoritmo PT. O PT irá avaliar mais 50000 soluções, tal como descrito na Secção 5.3.2.

Os dois algoritmos híbridos, como usam o algoritmo AG, à semelhança do AG simples, também foram executados com as quatro variantes dos parâmetros descritos na Figura 46.

Cada algoritmo é executado 10 vezes e os resultados apresentados para cada um dos testes efectuados, são a média obtida nas 10 execuções dos algoritmos.

Apesar do objectivo não ser o de comparar o resultado obtido pelos 5 algoritmos com o melhor conhecido para cada um dos 53 problemas usados, decidimos usar a medida:

$$(Makespan - \text{Óptimo}) / \text{Óptimo},$$

para facilitar a análise e comparação dos valores obtidos.

O valor *Makespan* usado na fórmula corresponde à média do resultado obtido por cada um dos algoritmos no conjunto das 10 repetições, e o valor *Óptimo* será o melhor valor conhecido até ao momento para cada um dos problemas. Isto significa que nos problemas em que o resultado da fórmula  $(Makespan - \text{Óptimo}) / \text{Óptimo}$  está próximo de zero se obtiveram valores muito próximos do melhor encontrado até então.

Estas experiências foram realizadas num computador Pentium, processador 1.60 GHz com 512 Mb RAM.

### 6.3. Resultados Obtidos

Após executar os 5 algoritmos com os 4 conjuntos de parâmetros referidos atrás, obtiveram-se os resultados médios em termos de *fitness* da melhor das 100000 soluções testadas que estão representados nos gráficos 1 a 4.

Estes gráficos gerais apresentam uma visão global do comportamento dos 5 algoritmos implementados, quando usados cada um dos quatro conjuntos de parâmetros de controlo dos métodos baseados em AGs.

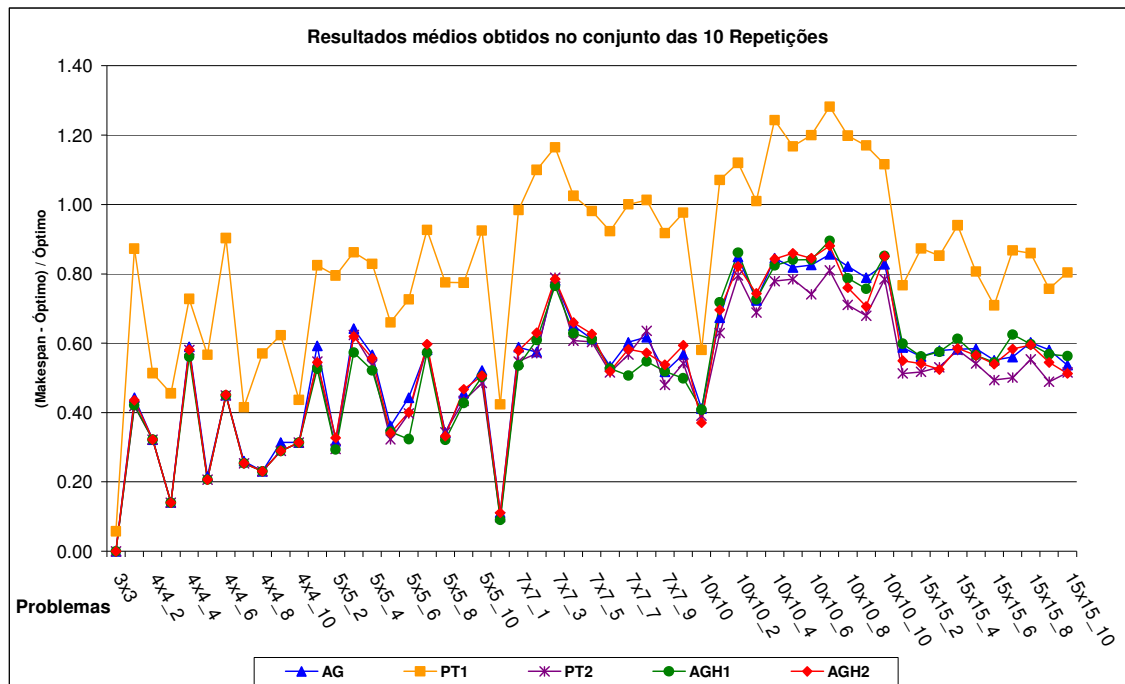


Gráfico 1. Comparação dos valores médios obtidos nas 10 repetições dos 5 algoritmos, para o 1º conjunto de parâmetros (popSize=50 e Pm=0.01).

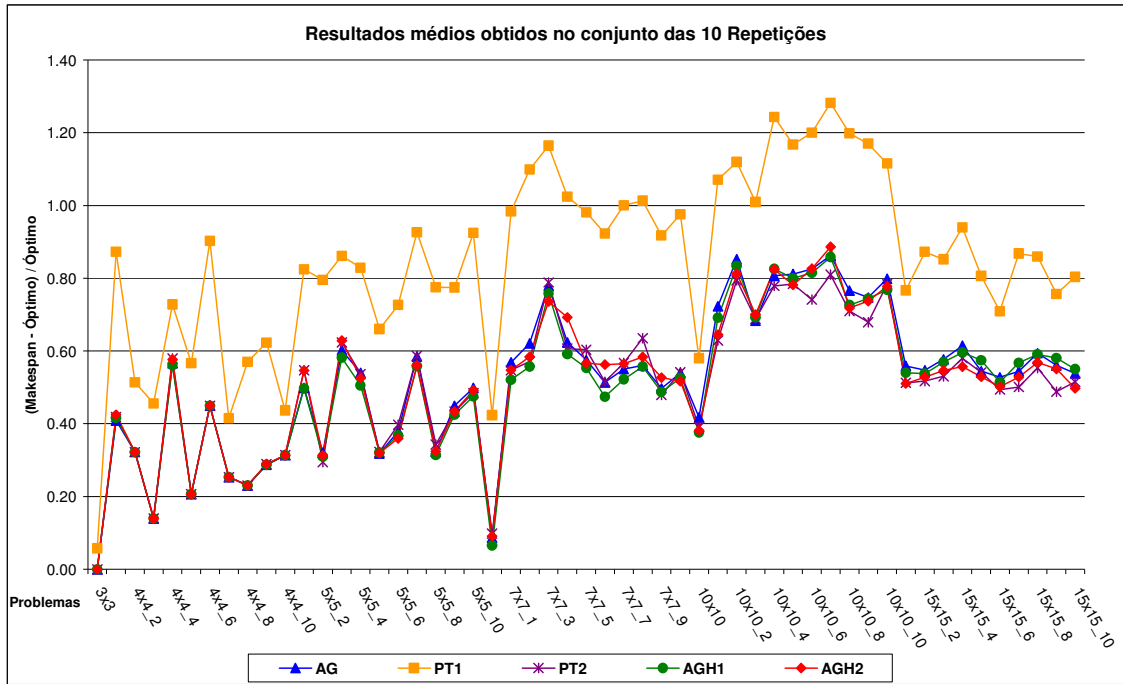


Gráfico 2. Comparação dos valores médios obtidos nas 10 repetições dos 5 algoritmos, para o 2º conjunto de parâmetros (popSize=100, Pm=0.01).

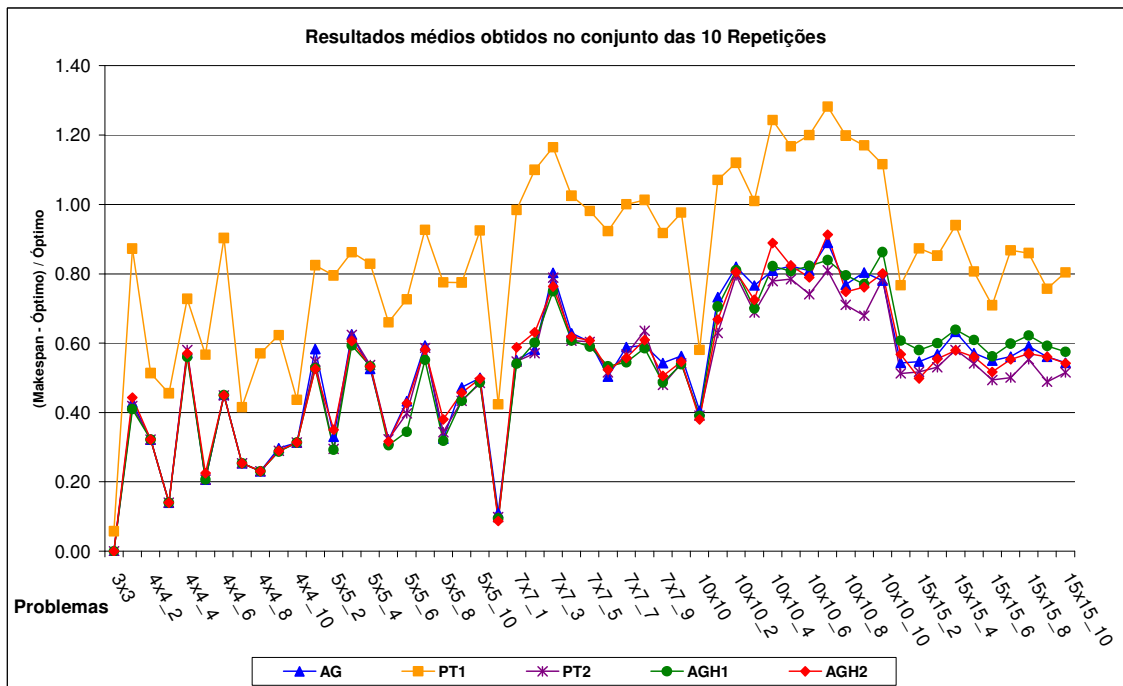


Gráfico 3. Comparação dos valores médios obtidos nas 10 repetições dos 5 algoritmos, para o 3º conjunto de parâmetros (popSize=50, Pm=0.1).

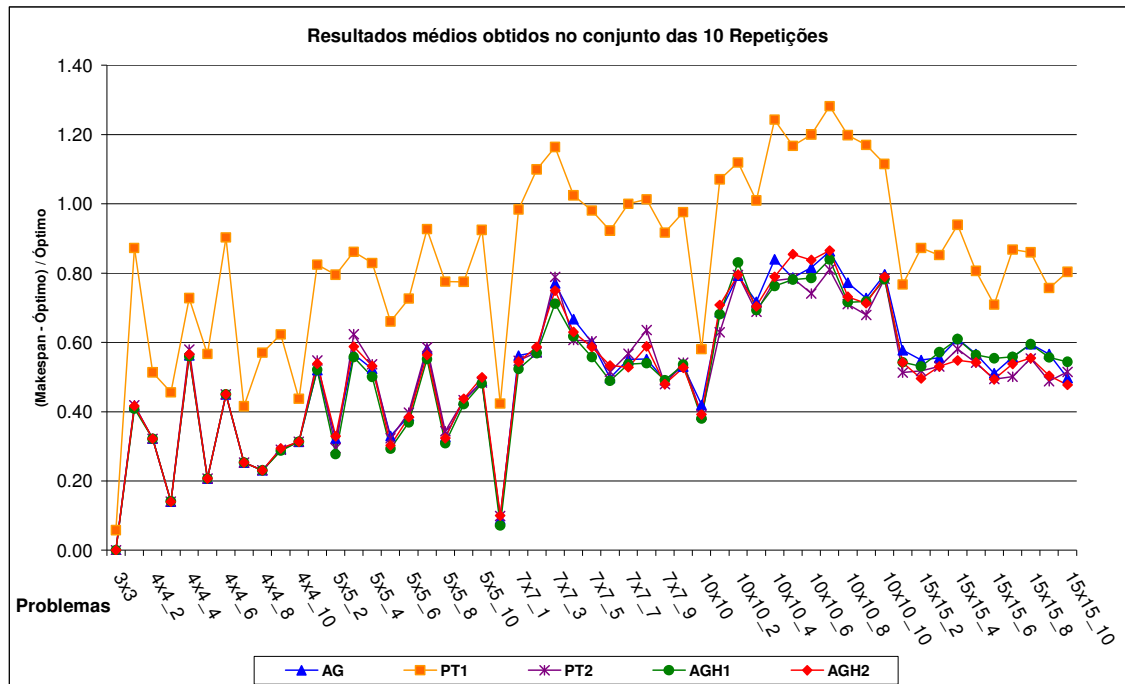


Gráfico 4. Comparação dos valores médios obtidos nas 10 repetições dos 5 algoritmos, para o 4º conjunto de parâmetros (popSize=100, Pm=0.1).

Da análise destes gráficos podemos retirar algumas conclusões iniciais. Após analisar 100000 soluções, o algoritmo PT2 obtém sempre melhores resultados que o algoritmo PT1. O que nos indica que efectivamente a qualidade da solução inicial influencia a qualidade da solução final obtida pela procura tabu.

Relativamente aos restantes algoritmos, para problemas pequenos (3x3 e 4x4) os resultados parecem ser muito parecidos, nos problemas de média dimensão (5x5, 6x6 e 7x7) o algoritmo AGH1, em média, parece obter melhores resultados, enquanto para problemas maiores, em metade dos problemas 10x10 o algoritmo AGH1 obtém melhores resultados e na outra metade o PT2 é melhor. Nos problemas 15x15, o algoritmo PT2 obtém melhores resultados em metade dos problemas e na outra metade surge o AGH2 com melhores resultados.

Nas secções seguintes iremos analisar com mais detalhe o comportamento de cada algoritmo implementado, bem como efectuar a sua comparação.

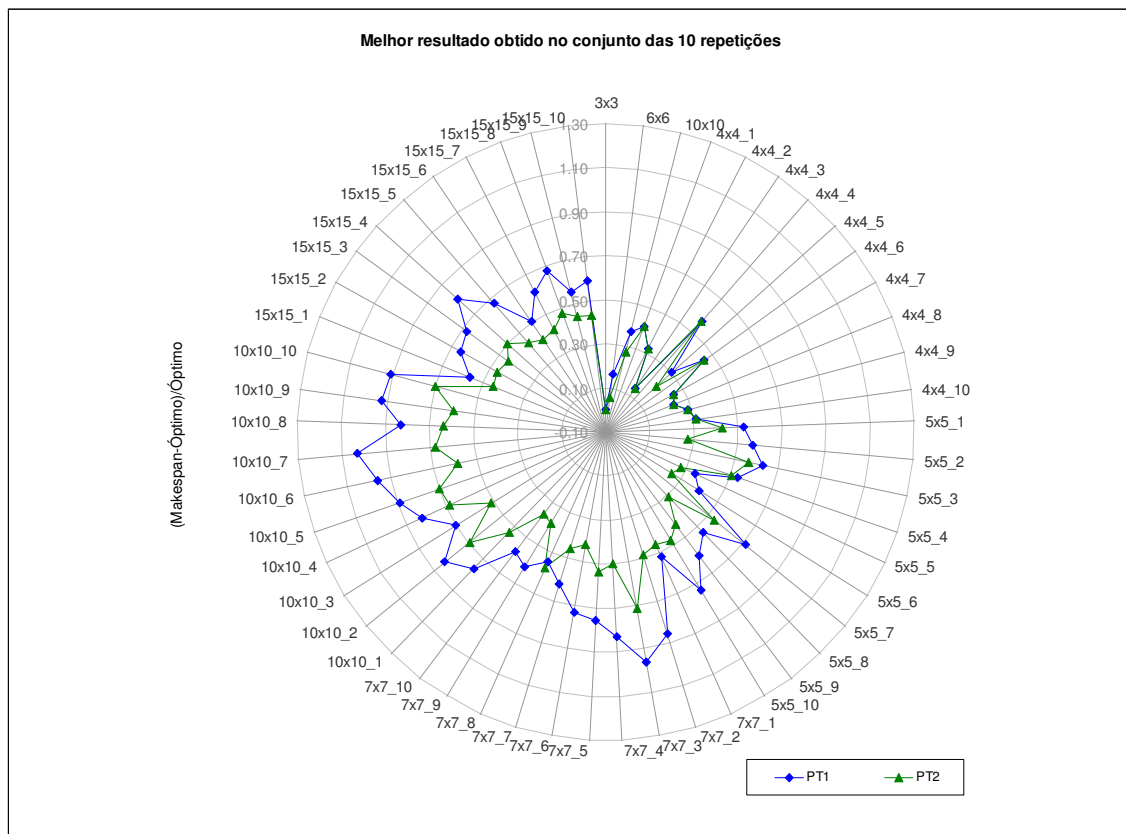
### 6.3.1. Análise do resultado dos algoritmos PT1 e PT2

De acordo com os resultados apresentados nos gráficos 1 a 4, o algoritmo PT1 obtém piores resultados do que os outros algoritmos, isto independentemente do caso analisado e do conjunto de parâmetros usado nos métodos baseados em AG.

Fazendo uma comparação entre o desempenho do PT1, que inicia a procura com uma solução inicial gerada aleatoriamente, e o PT2, com a solução inicial a ser seleccionada através da melhor de 100 soluções geradas aleatoriamente, verifica-se que o desempenho do PT2 é, em média, 18.5% superior ao do PT1.

Daqui podemos comprovar que no caso dos algoritmos PT1 e PT2 que implementam a procura tabu, a qualidade da solução final depende muito significativamente da qualidade da solução inicial, confirmando resultados publicados anteriormente [Geyik e Cedimoglu, 2004].

No Gráfico 5 podemos comparar o resultado global do PT1 e do PT2 no conjunto dos 53 problemas testados.



Como se pode ver pelos resultados apresentados, o algoritmo PT2 obtém sempre melhores resultados que o PT1, isto quando avaliado o mesmo número de soluções.

Isto acontece porque ao fazer a procura local começando na melhor de 100 soluções geradas aleatoriamente, o PT2 está na realidade a identificar uma zona do espaço de soluções promissora, antes de iniciar a procura.

Por seu lado, o PT1 faz a procura numa zona escolhida de forma completamente aleatória.

Embora, como dito anteriormente, o objectivo deste estudo é comparar a performance dos métodos em termos do número de soluções avaliadas e não o tempo de execução, os valores respectivos foram registados e são brevemente analisados. Relativamente ao tempo necessário para obtenção da melhor solução possível pelos algoritmos PT1 e PT2, podemos ver os resultados nos gráficos 6 a 8. Os resultados foram divididos em 3 gráficos para facilitar a leitura uma vez que a escala temporal é muito diferente quando se trata de um problema pequeno, médio ou grande.

No Gráfico 6 podemos ver os resultados obtidos para os problemas 3x3 e 4x4, no Gráfico 7 os resultados dos problemas 5x5, 6x6 e 7x7, e no Gráfico 8 o resultado dos problemas do tipo 10x10 e 15x15. Os tempos apresentados correspondem ao tempo médio obtido no conjunto das 10 execuções dos algoritmos, até ao momento em que cada um dos algoritmos encontra a melhor solução das 100000 avaliadas.

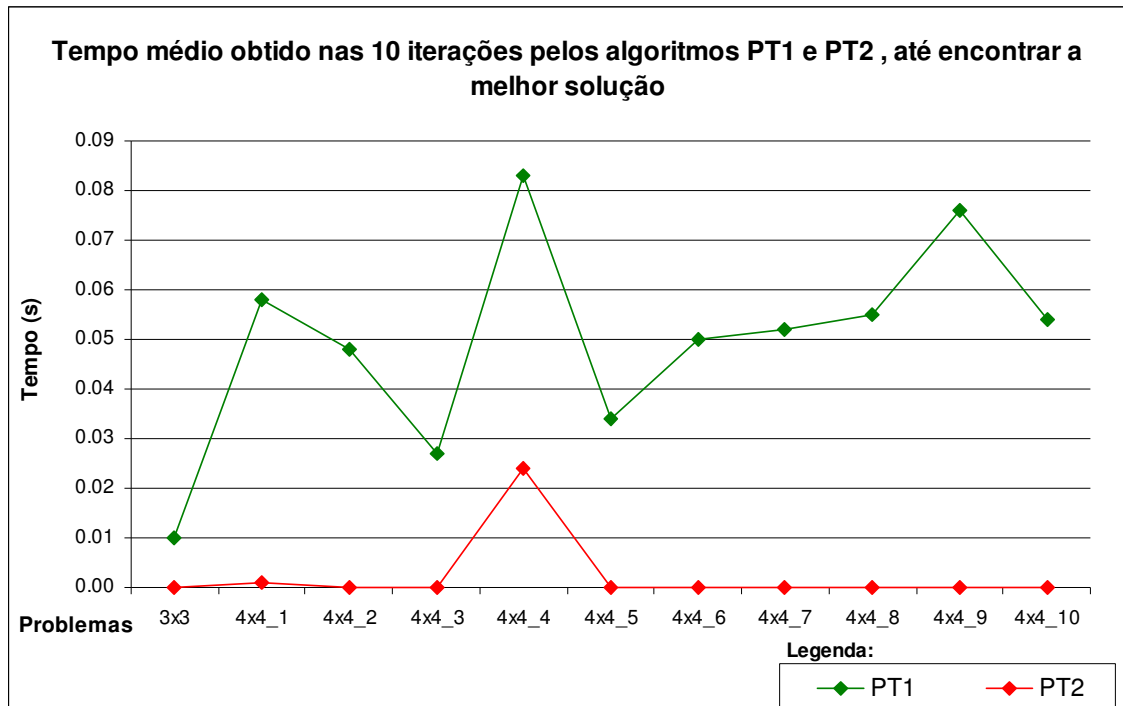


Gráfico 6. Resultado dos tempo médios necessários pelos algoritmos PT1 e PT2, para encontrar a melhor solução, para os problemas do tipo 3x3 e 4x4 testados.

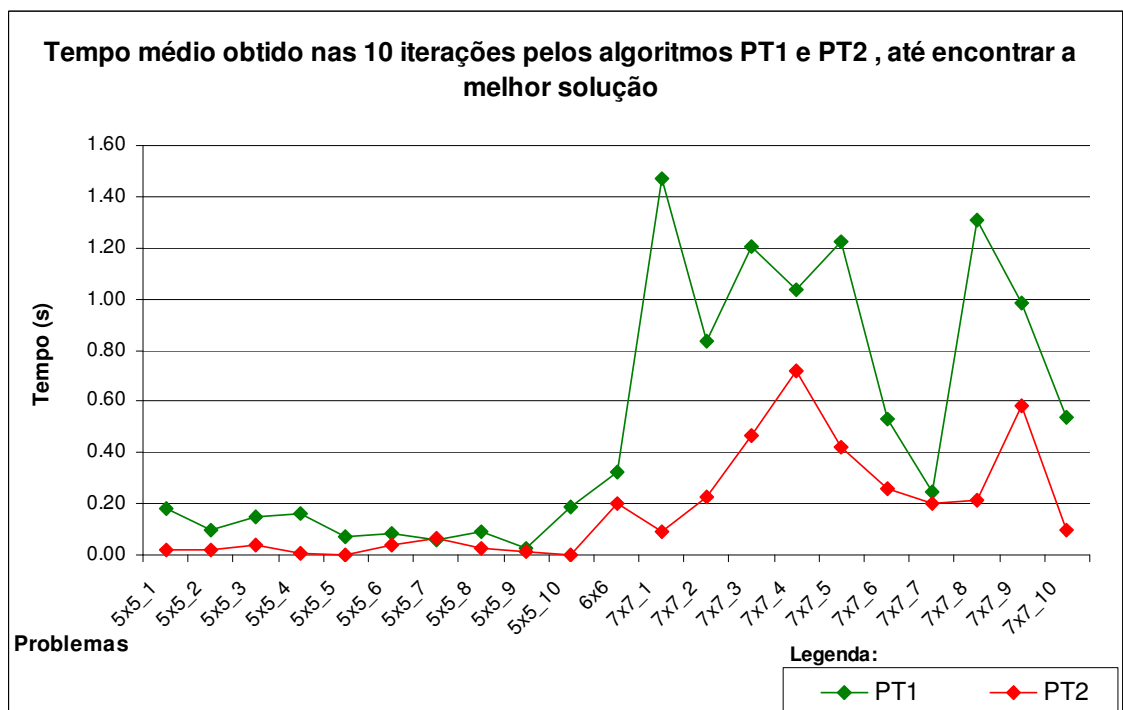


Gráfico 7. Resultado dos tempo médios necessários pelos algoritmos PT1 e PT2, para encontrar a melhor solução, para os problemas do tipo 5x5, 6x6 e 7x7 testados.

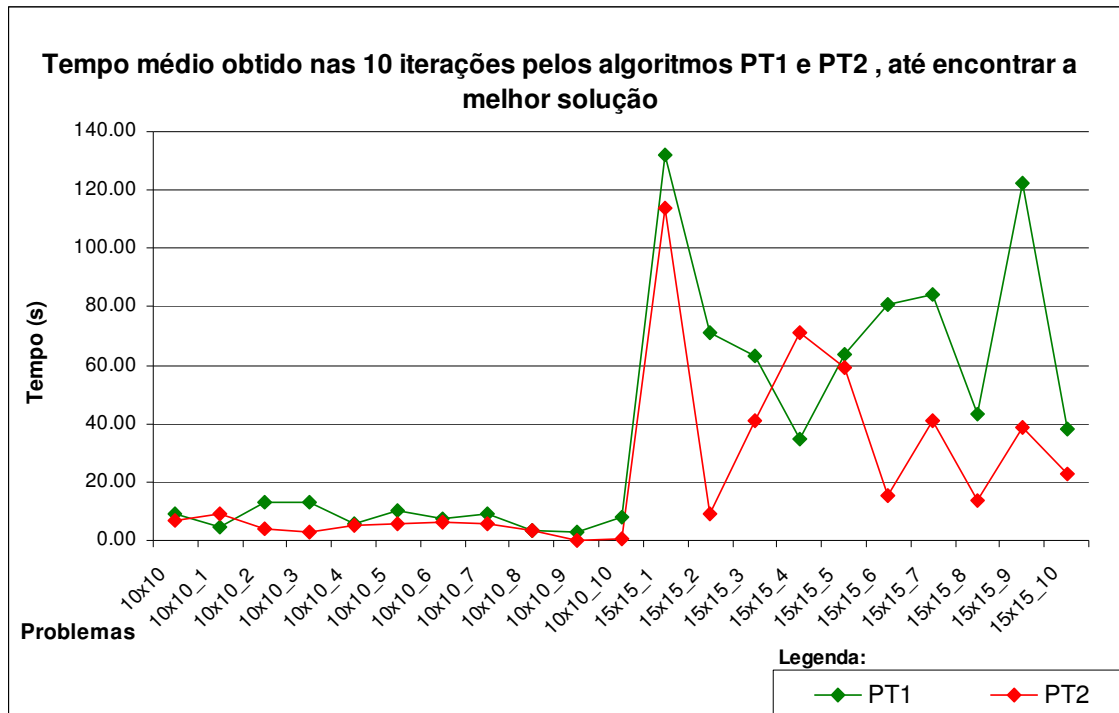


Gráfico 8. Resultado dos tempo médios necessários pelos algoritmos PT1 e PT2, para encontrar a melhor solução, para os problemas do tipo 10x10 e 15x15 testados.

Da análise dos Gráficos 5, 6, 7, e 8 podemos verificar que o algoritmo PT2 além de obter melhores resultados, em média obtém-nos em menos tempo. Isto acontece, porque como o algoritmo PT2 inicia o processo de procura a partir de uma zona do espaço onde se encontra uma solução promissora, ele termina o processo de busca antes do algoritmo PT1.

Relativamente à evolução da procura, verificamos, como seria de esperar, que a procura tabu é muito rápida a encontrar uma solução boa, mas que frequentemente encontra um mínimo local e depois já não consegue continuar o processo de procura noutras regiões do espaço. Para todos os tipos de problemas testados, verificamos que os algoritmos PT1 e PT2 conseguem introduzir grandes melhorias na solução inicial nas primeiras 1000 soluções testadas. Depois das primeiras 1000 soluções avaliadas, estes algoritmos fazem melhorias na solução corrente, mas de forma menos expressiva.

Como exemplo, ver gráficos 9 a 12 para um dos problemas do tipo 4x4, 7x7, 10x10 e 15x15. Nestes gráficos podemos ver o resultado das 5000 primeiras soluções analisadas para cada um destes 5 tipos de problemas. Para os restantes problemas o comportamento é semelhante.

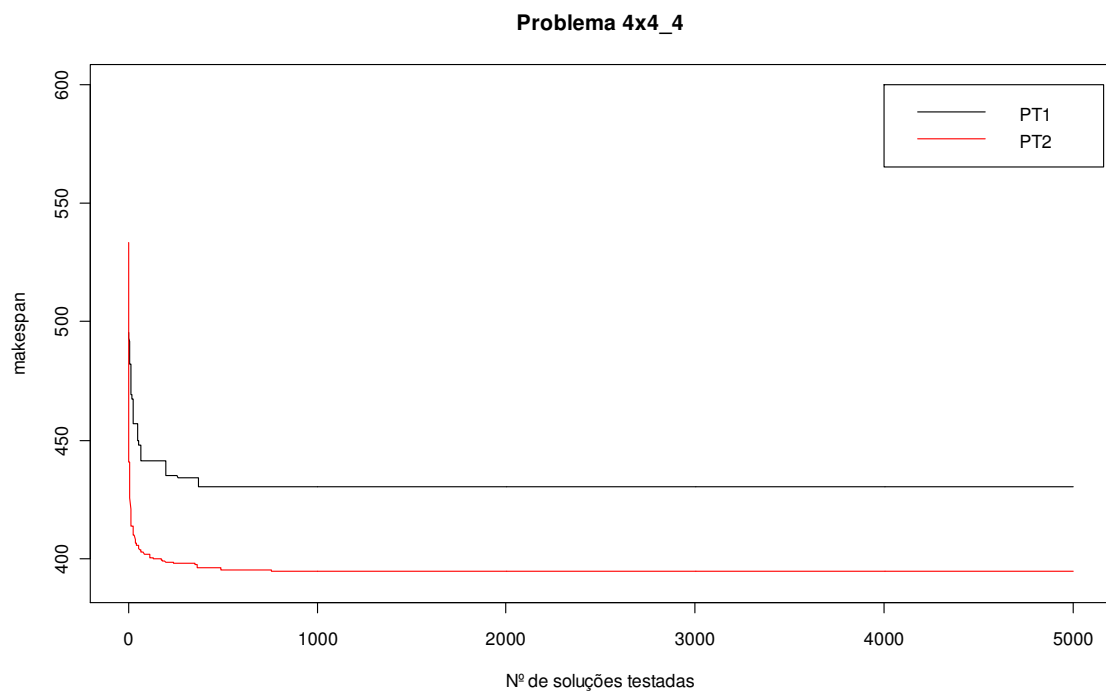


Gráfico 9. Resultados médios obtidos pelos algoritmos PT1 e PT2 na avaliação das primeiras 5000 soluções, para o problema 4x4\_4.

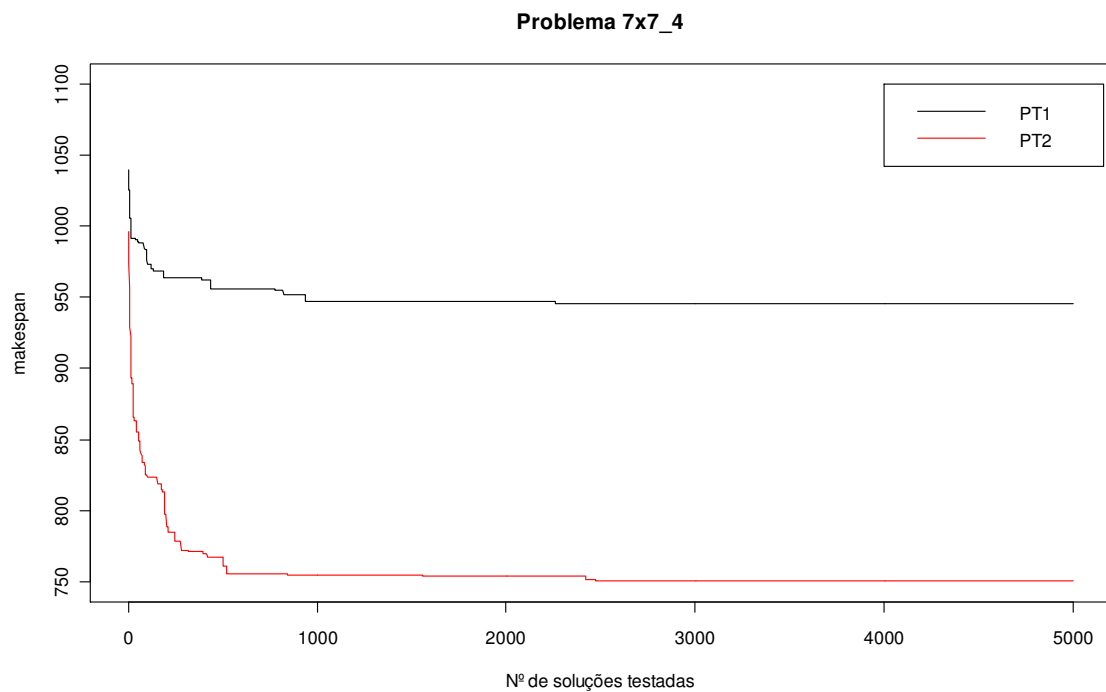


Gráfico 10. Resultados médios obtidos pelos algoritmos PT1 e PT2 na avaliação das primeiras 5000 soluções, para o problema 7x7\_4.

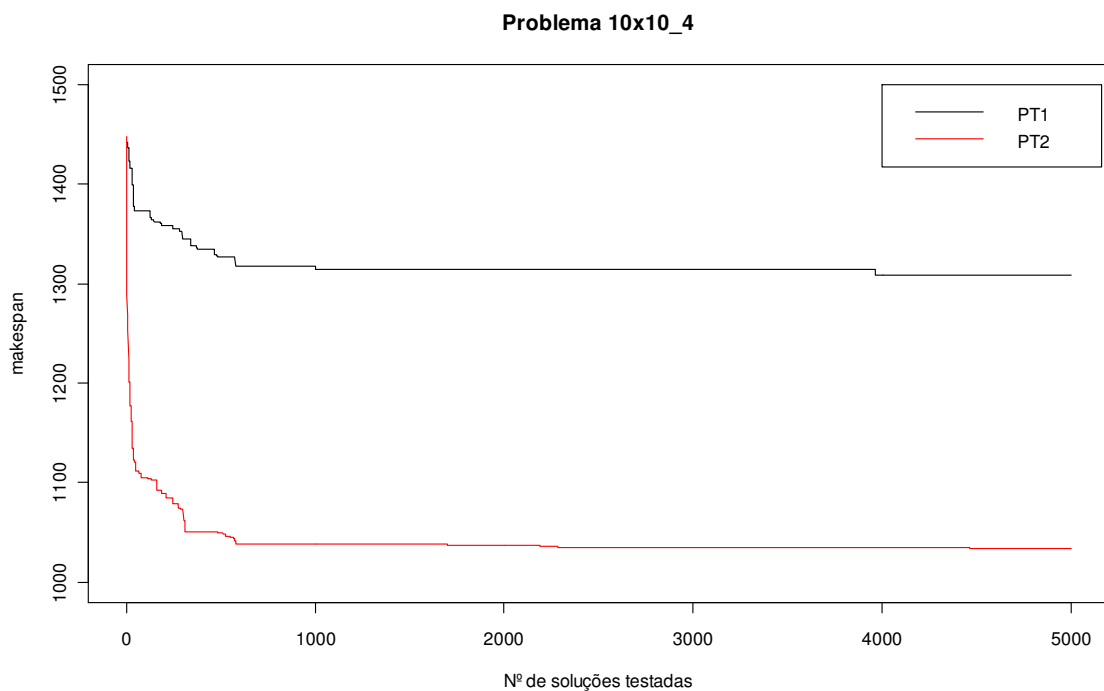


Gráfico 11. Resultados médios obtidos pelos algoritmos PT1 e PT2 na avaliação das primeiras 5000 soluções, para o problema 10x10\_4.

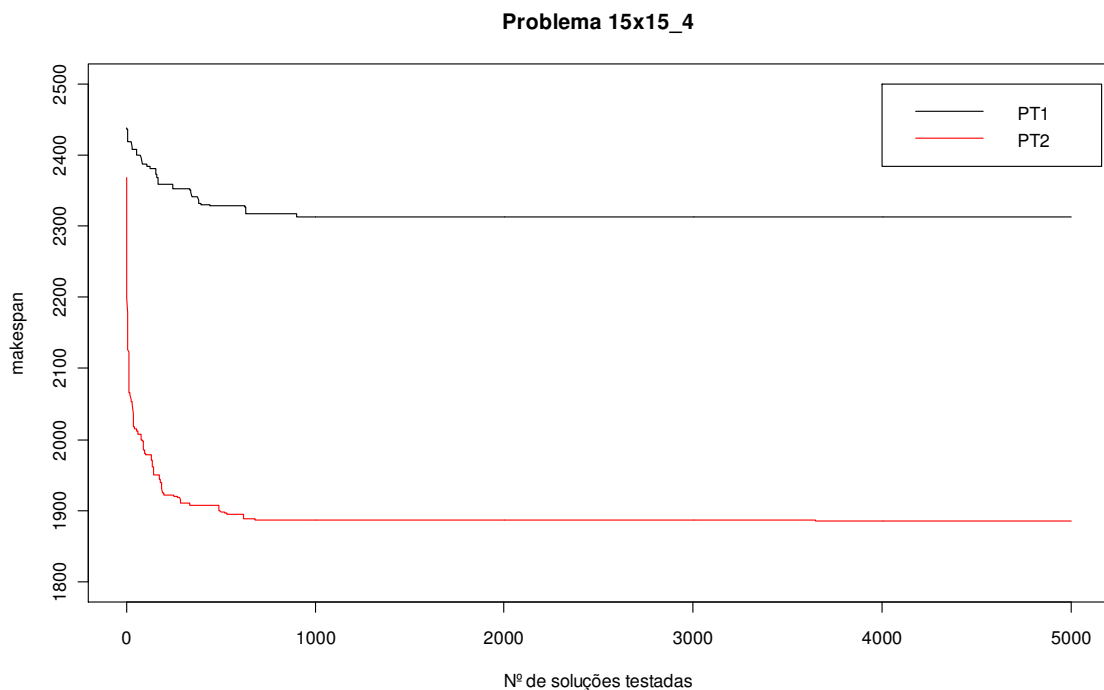


Gráfico 12. Resultados médios obtidos pelos algoritmos PT1 e PT2 na avaliação das primeiras 5000 soluções, para o problema 15x15\_4.

Adicionalmente, da análise destes resultados podemos concluir que as primeiras 1000 soluções testadas pela procura tabu são críticas para a melhoria da solução inicial.

Esta observação indica que vale a pena investir na procura da solução inicial da PT como é feito no AGH2.

De salientar que os algoritmos PT1 e o PT2 continuam o processo de busca da melhor solução para além das 5000 soluções.

Dado que as 100 primeiras soluções a partir das quais o PT2 selecciona a solução inicial também estão reportadas nos gráficos, estes resultados confirmam a hipótese de que os melhores resultados do algoritmo PT2 se devem a começar a procura numa solução melhor.

### **6.3.2. Análise do resultado do algoritmo AG**

Relativamente aos parâmetros usados no algoritmo AG observou-se que quando usadas populações de 100 indivíduos, geralmente se obtêm melhores resultados. Os gráficos 13 a 15 permitem-nos ter uma visão geral e comparar os resultados obtidos pelo AG quando usados os quatro conjuntos de parâmetros de controlo. Como referido anteriormente no primeiro e terceiro conjuntos de parâmetros é usada uma probabilidade de mutação de 0.01 e uma probabilidade de cruzamento de 1, sendo a dimensão da população de 50 indivíduos no primeiro conjunto de parâmetros e 100 indivíduos no terceiro. No segundo e quarto conjuntos de parâmetros, é usada uma probabilidade de mutação de 0.1 e uma probabilidade de cruzamento de 1, sendo a dimensão da população de 50 indivíduos no segundo conjunto de parâmetros e 100 no quarto.

Para facilitar a análise, optamos por dividir os resultados dos 53 problemas testados por três gráficos, de acordo com a dimensão do problema.

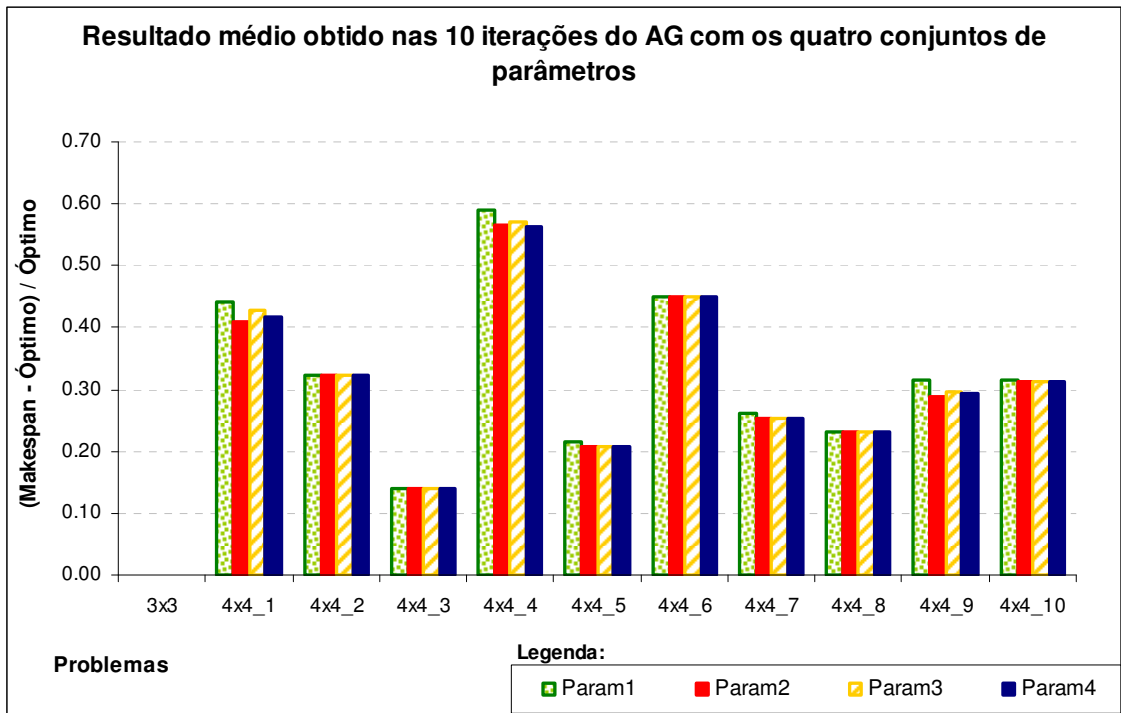


Gráfico 13. Resultados médios obtidos pelo algoritmo AG quando usados os quatro conjuntos de parâmetros, para os problemas do tipo 3x3 e 4x4.

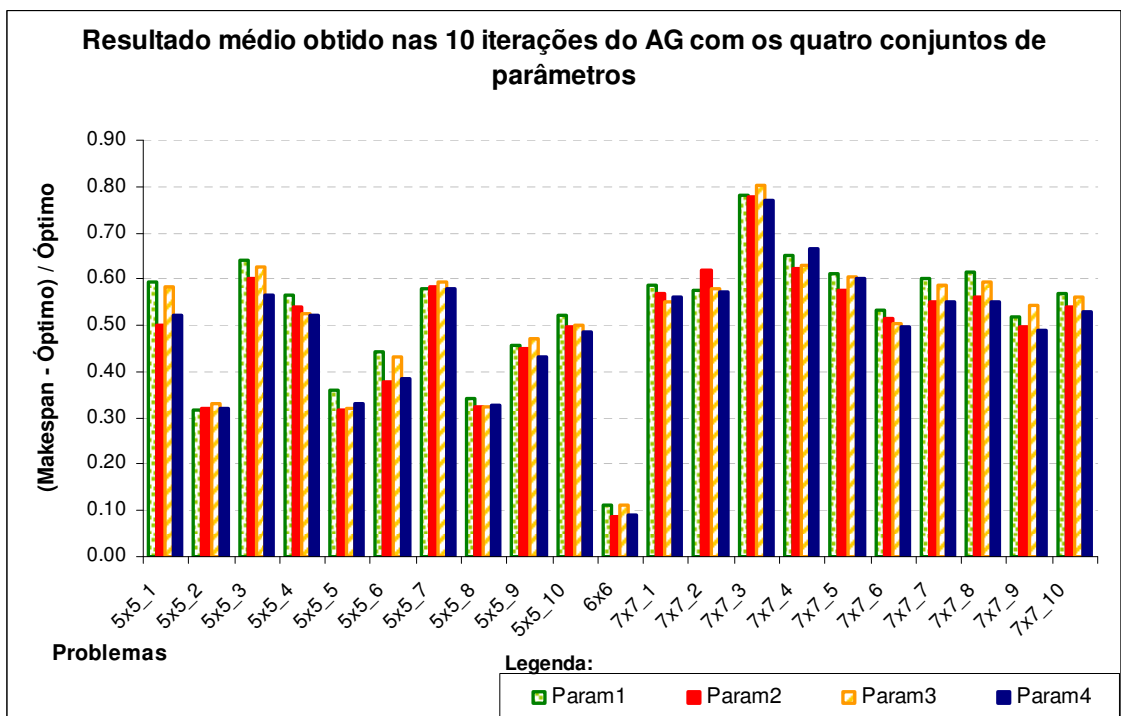


Gráfico 14. Resultados médios obtidos pelo algoritmo AG, quando usados os quatro conjuntos de parâmetros, para os problemas do tipo 5x5, 6x6 e 7x7.

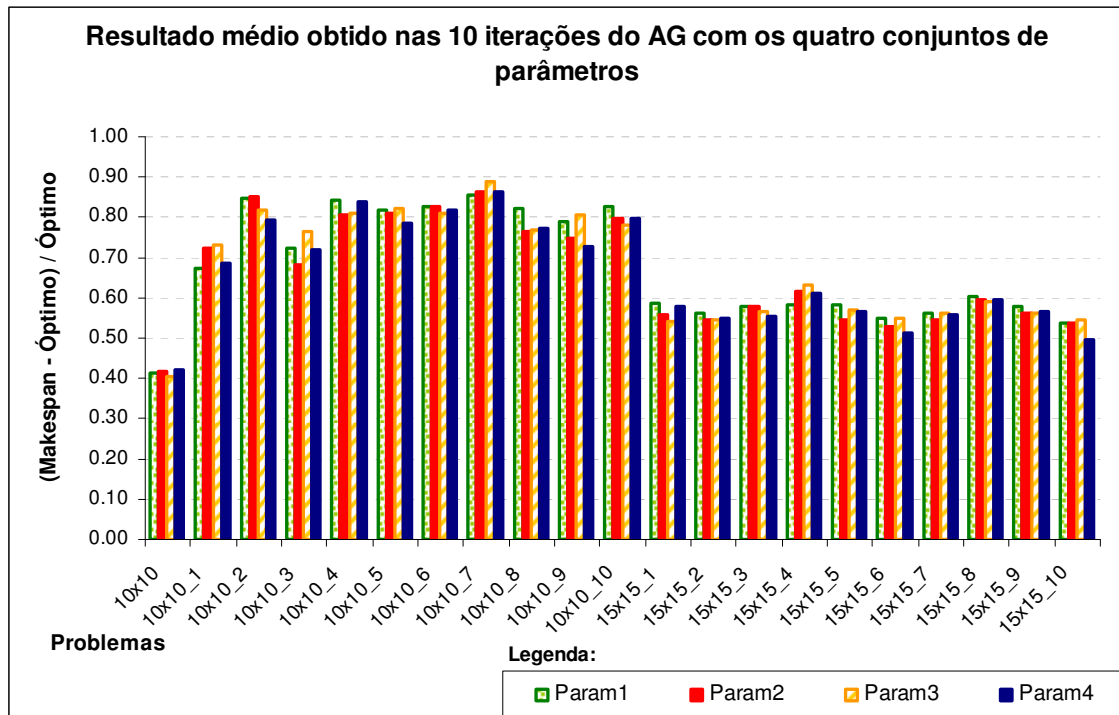


Gráfico 15. Resultados médios obtidos pelo algoritmo AG, quando usados os quatro conjuntos de parâmetros, para os problemas do tipo 10x10 e 15x15.

Como podemos ver pelo Gráfico 13, para os problemas do tipo 3x3 e 4x4, que nós classificamos como sendo de pequena dimensão, os conjuntos de parâmetros 2 e 4 revelam resultados ligeiramente superiores. No Gráfico 14, podemos ver que para os problemas de média dimensão, os 10 problemas do tipo 5x5, o 6x6 e os 10 problemas do tipo 7x7, também se verificaram resultados médios ligeiramente superiores quando usados os conjuntos de parâmetros 2 e 4. O mesmo se verifica, pelo Gráfico 15, com os problemas de maior dimensão, onde tanto para os 10 problemas do tipo 10x10 como para os do tipo 15x15, em média o conjunto de parâmetros 2 e 4 também obtêm melhores resultados.

Da análise destes resultados podemos concluir que a dimensão da população é um factor importante para a performance do algoritmo genético.

No primeiro e segundo conjuntos de parâmetros foi usada uma probabilidade de mutação de 0.01 enquanto no terceiro e quarto conjuntos de parâmetros foi usada uma probabilidade de mutação de 0.1. Aparentemente quando usada uma probabilidade de mutação de 0.1, os resultados são, em média, ligeiramente superiores, mas a diferença é pouco relevante. Dos 53 problemas testados, quando usada uma probabilidade de

mutação de 0.1 obtiveram-se resultados melhores em 30 problemas, e em 12 obtiveram-se resultados iguais.

No geral, o conjunto de parâmetros com melhores resultados médios no total dos 53 problemas é o quarto conjunto de parâmetros, em que se usou uma população de 100 indivíduos com uma probabilidade de mutação de 0.1 e uma probabilidade de cruzamento de 1.

No entanto, como já referido, as diferenças são geralmente pouco relevantes.

Relativamente ao tempo necessário para o algoritmo AG encontrar a melhor solução possível, com um limite máximo de 100000 soluções testadas, quando usados cada um dos 4 conjuntos de parâmetros, podemos nos gráficos 16 a 18 comparar os resultados obtidos. Mais uma vez, para facilitar a análise dos resultados, os 53 problemas foram divididos em 3 gráficos.

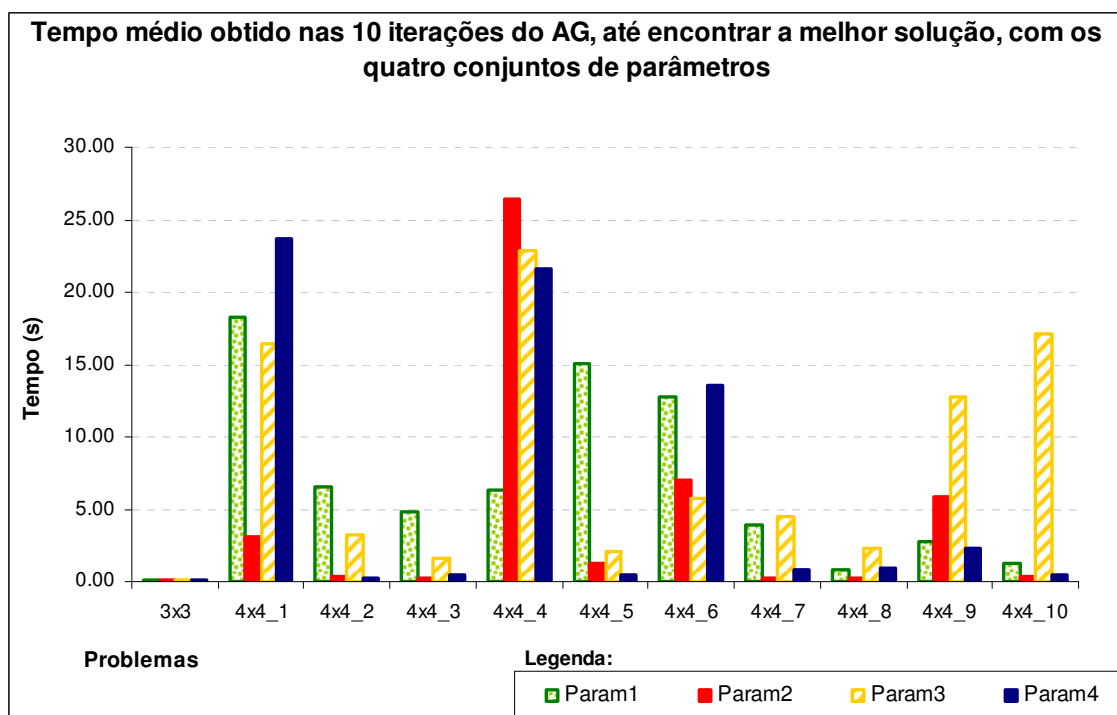


Gráfico 16. Resultado dos tempos médios necessários para encontrar a melhor solução possível, quando usados os quatro conjuntos de parâmetros do AG, para os problemas do tipo 3x3 e 4x4.

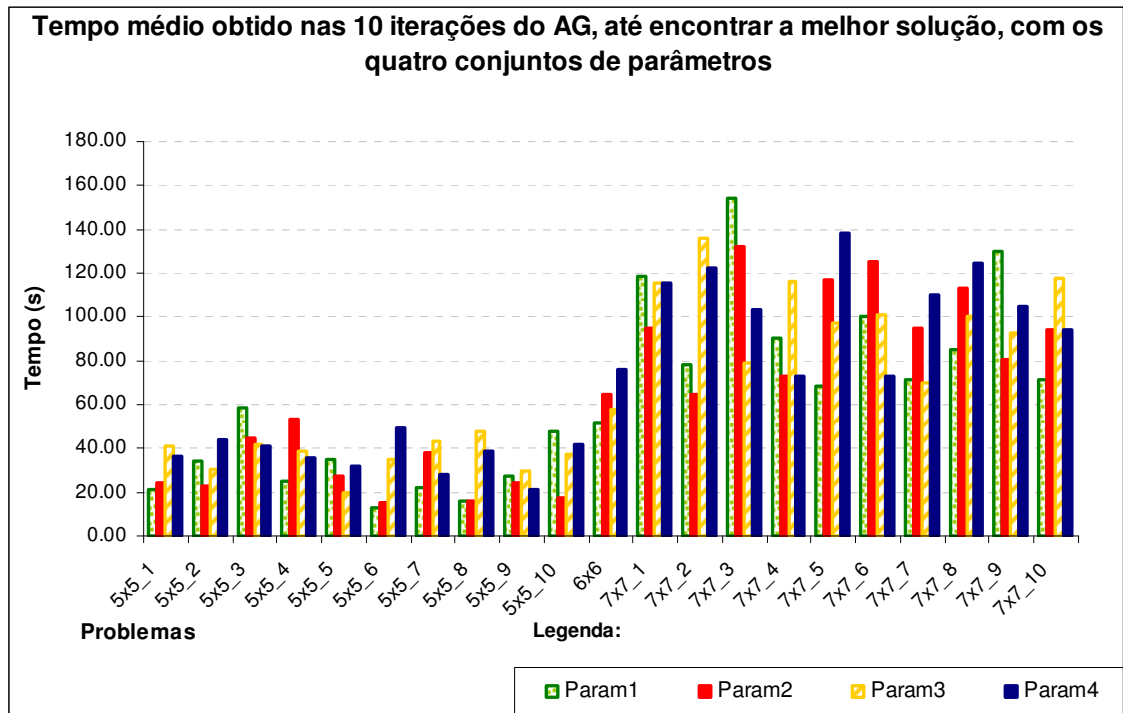


Gráfico 17. Resultado dos tempos médios necessários para encontrar a melhor solução possível, quando usados os quatro conjuntos de parâmetros do AG, para os problemas do tipo 5x5, 6x6 e 7x7.

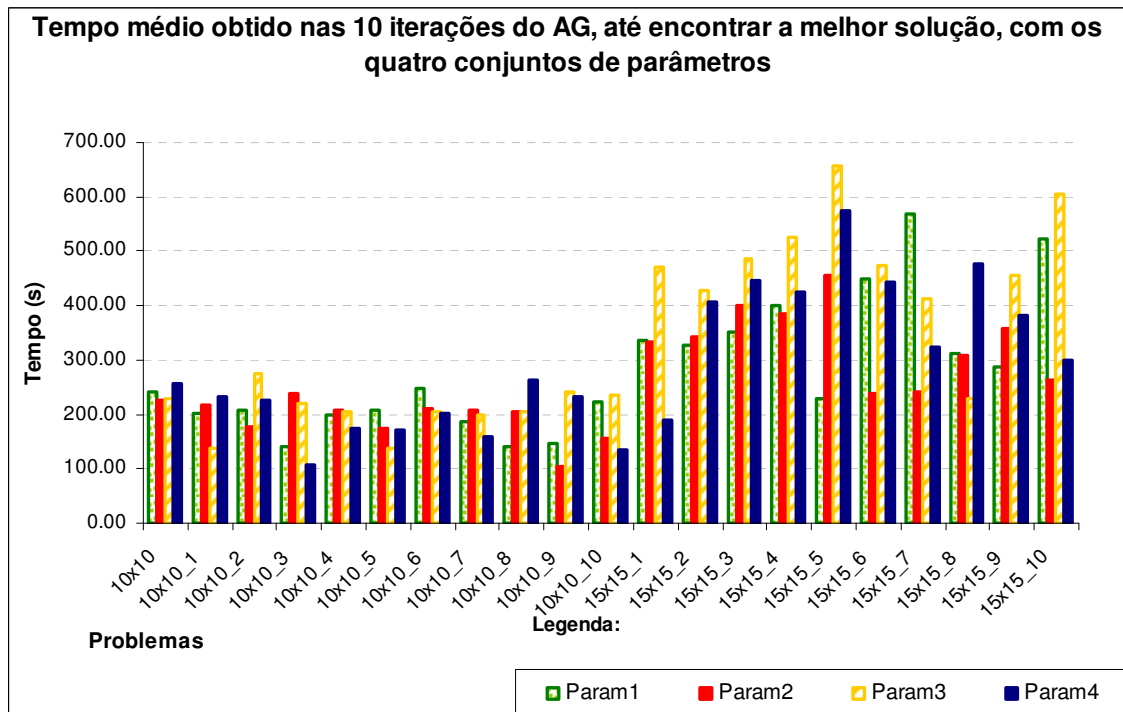


Gráfico 18. Resultado dos tempos médios necessários para encontrar a melhor solução possível, quando usados os quatro conjuntos de parâmetros do AG, para os problemas do tipo 10x10 e 15x15.

Pela análise destes gráficos podemos ver que os tempos de computação necessários pelo AG para encontrar a melhor solução possível de entre 100000 testadas, são praticamente independentes do conjunto de parâmetros.

Apesar do quarto conjunto de parâmetros (população de 100 indivíduos com probabilidade de mutação de 0.1), nos 53 problemas testados ter obtido resultados médios ligeiramente melhores que quando usados os restantes conjuntos de parâmetros, na verdade a vantagem é pouco expressiva. Esta vantagem pode ser explicada pelo facto de que quanto maior for a população e a probabilidade de mutação, maior é a diversidade que se introduz na população (mais soluções novas) em cada geração do AG. No entanto esta vantagem é pouco expressiva, até porque independentemente do conjunto de parâmetros usados, o AG ainda não terminou o seu processo de procura. Isto é visível nos gráficos 19 a 22, onde apresentamos os resultados da evolução da procura do AG, quando usados os quatro conjuntos de parâmetros, para um problema de cada um dos tipos 4x4, 7x7, 10x10 e 15x15.

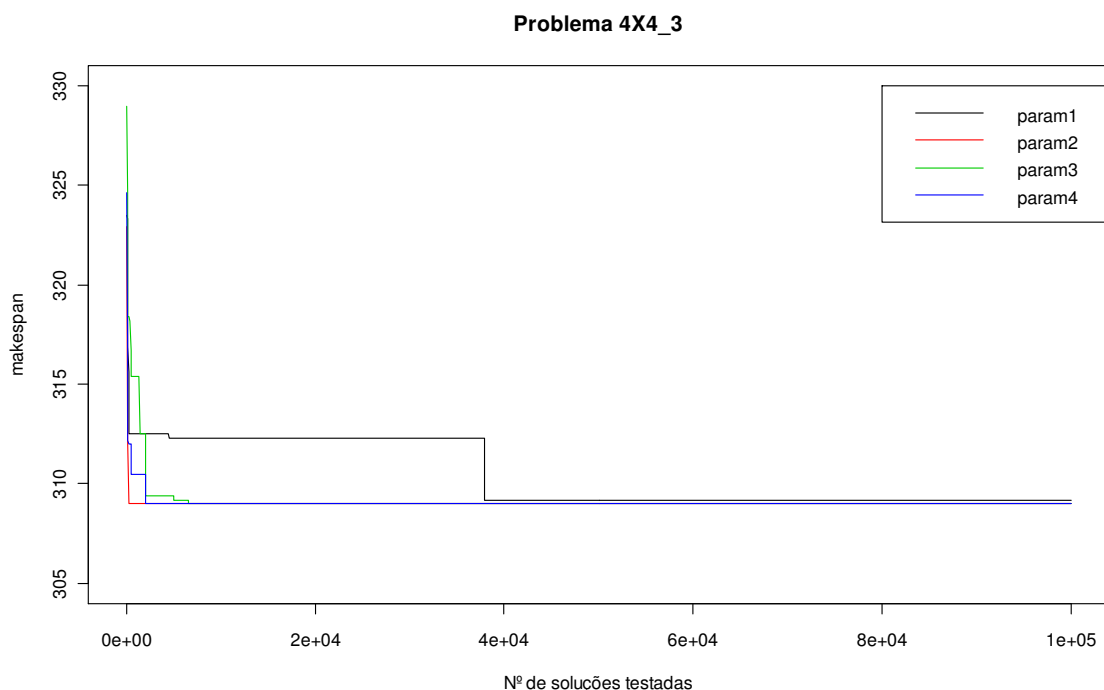


Gráfico 19. Resultados médios da evolução da procura, quando usados os quatro conjuntos de parâmetros, na execução do AG, para o problemas 4x4\_3.

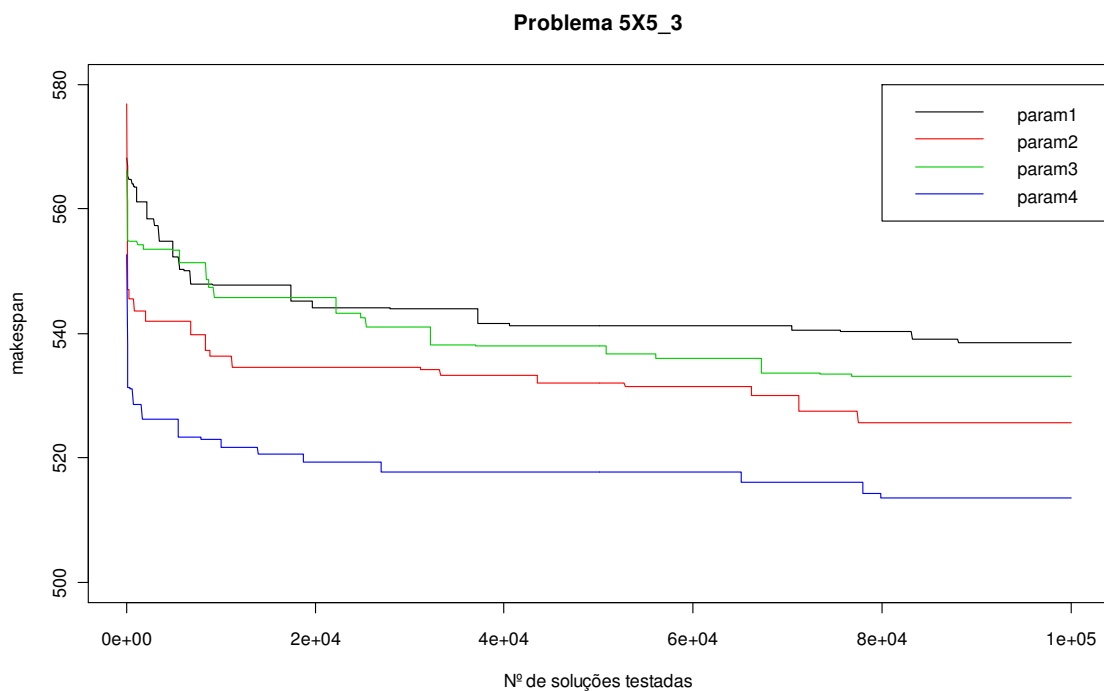


Gráfico 20. Resultados médios da evolução da procura, quando usados os quatro conjuntos de parâmetros, na execução do AG, para o problemas 5x5\_3.

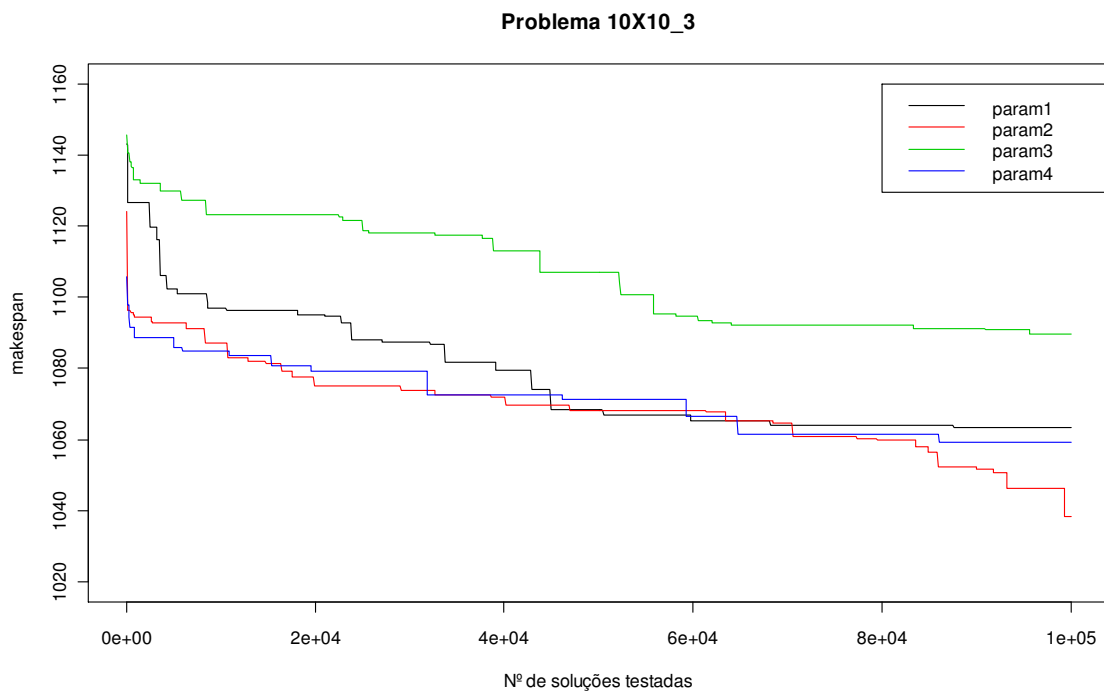


Gráfico 21. Resultados médios da evolução da procura, quando usados os quatro conjuntos de parâmetros, na execução do AG, para o problemas 10x10\_3.

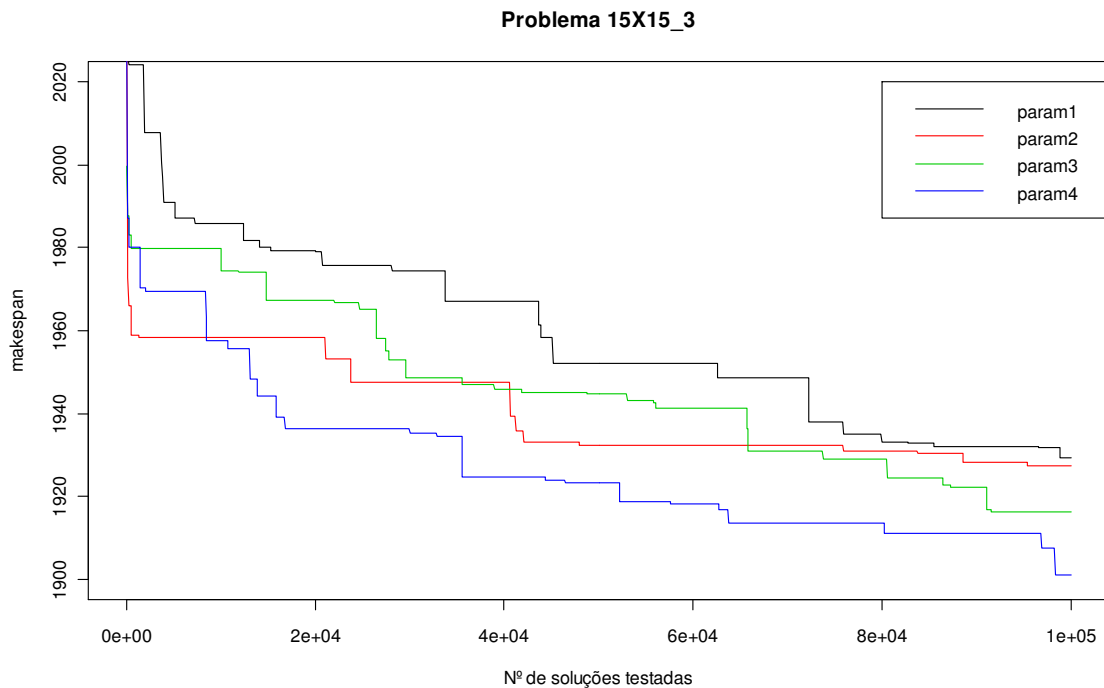


Gráfico 22. Resultados médios da evolução da procura, quando usados os quatro conjuntos de parâmetros, na execução do AG, para o problemas 15x15\_3.

Para além de em média obter melhores resultados, o quarto conjunto de parâmetros, tem tipicamente um comportamento melhor desde o início do processo de procura. Isto pode ser explicado pelo facto de, ao usar uma população maior e uma probabilidade de mutação maior, em cada geração é introduzida uma maior variabilidade na população, permitindo que se direcione a procura para diferentes regiões do espaço, efectuando uma procura mais eficaz.

### 6.3.3. Análise do resultado dos algoritmos AGH1 e AGH2

Os gráficos 23 a 25 permitem-nos comparar os resultados obtidos pelo algoritmo AGH1 quando usados os quatro conjuntos de parâmetros de controlo.

Mais uma vez, optamos por dividir os resultados dos 53 problemas testados por três gráficos, de acordo com a dimensão do problema.

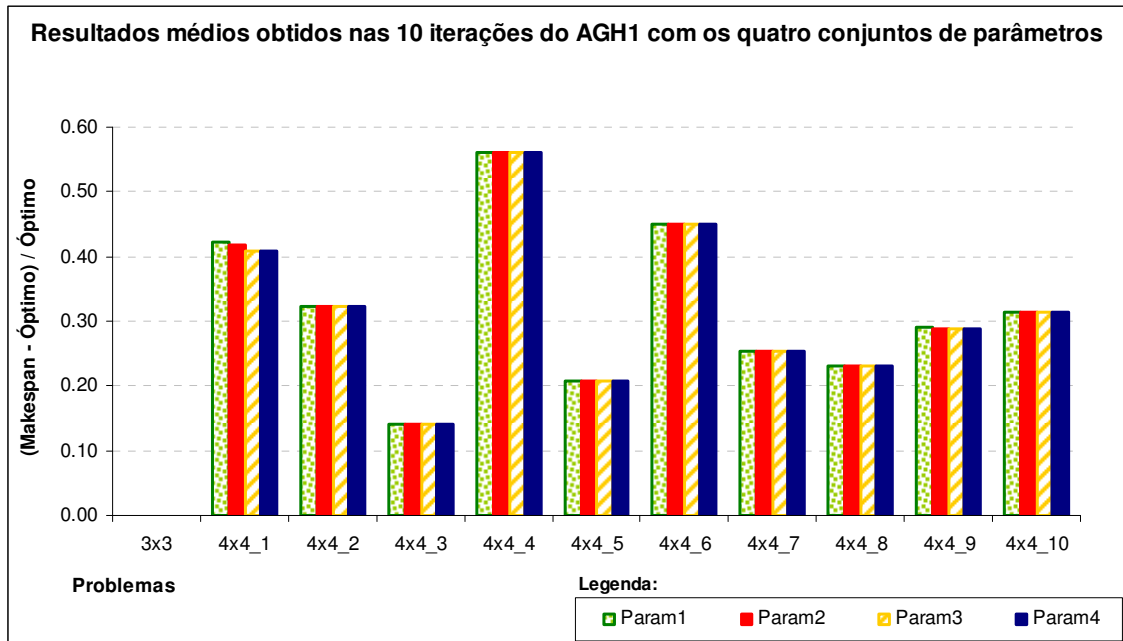


Gráfico 23. Resultados médios obtidos pelo algoritmo AGH1, quando usados os quatro conjuntos de parâmetros, para os problemas do tipo 3x3 e 4x4.

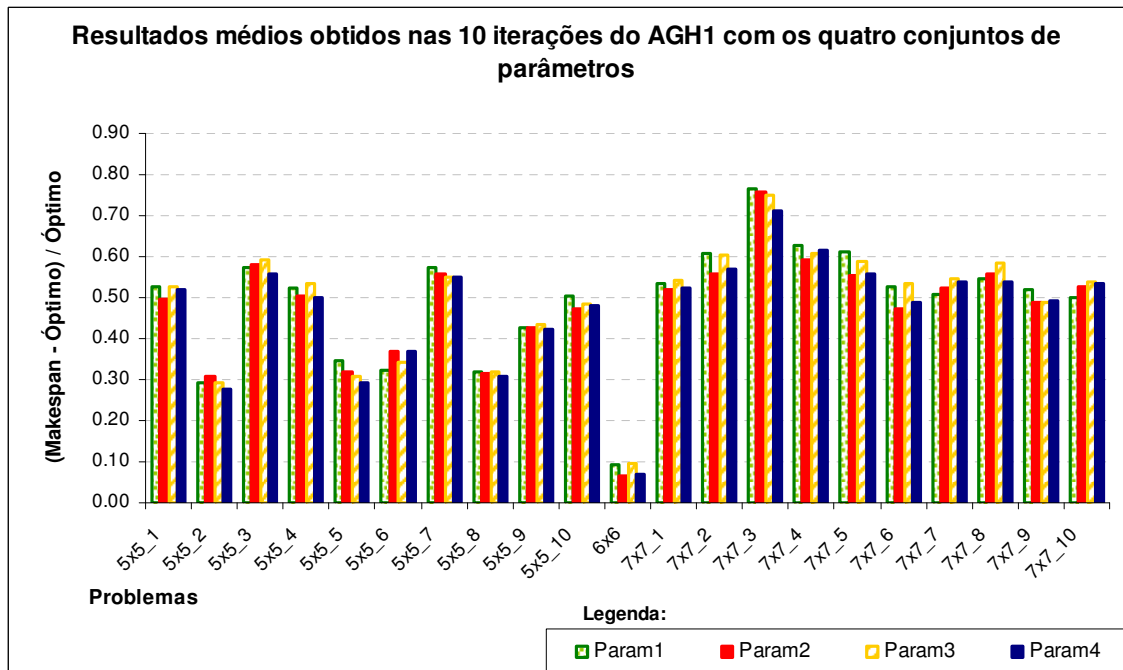


Gráfico 24. Resultados médios obtidos pelo algoritmo AGH1, quando usados os quatro conjuntos de parâmetros, para os problemas do tipo 5x5, 6x6 e 7x7.

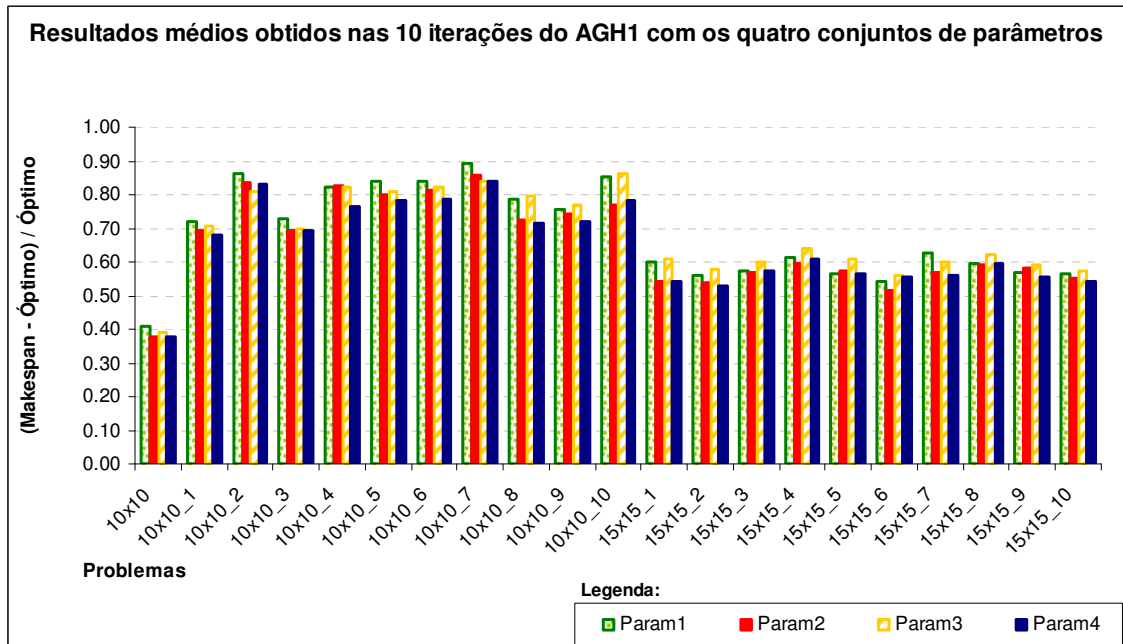


Gráfico 25. Resultados médios obtidos pelo algoritmo AGH1, quando usados os quatro conjuntos de parâmetros, para os problemas do tipo 10x10 e 15x15.

Nos gráficos 23 a 25, podemos ver que quer para os problemas pequenos, médios ou grandes, o algoritmo AGH1 é pouco influenciado pelo conjunto de parâmetros usados.

Ainda assim, podemos observar que em média, se verifica uma ligeira melhoria da solução final quando se usam o segundo e o quarto conjunto de parâmetros, ou seja quando usadas populações de 100 indivíduos. Sendo que o quarto conjunto de parâmetros revela resultados ligeiramente superiores para os problemas maiores. Dado que este último conjunto tem uma maior probabilidade de mutação, isto significa que em cada geração do AG vamos introduzir maior diversidade na população (novas soluções de outras zonas do espaço de procura), e por outro lado, ao aplicar a PT na melhor solução da geração estamos a contribuir para uma procura mais eficaz.

Relativamente ao tempo médio necessário para encontrar a melhor solução, o algoritmo AGH1, também não é influenciado pelo conjunto de parâmetros usado, tal como se tinha observado anteriormente com o algoritmo AG. Para ver detalhes, consultar anexo 7.

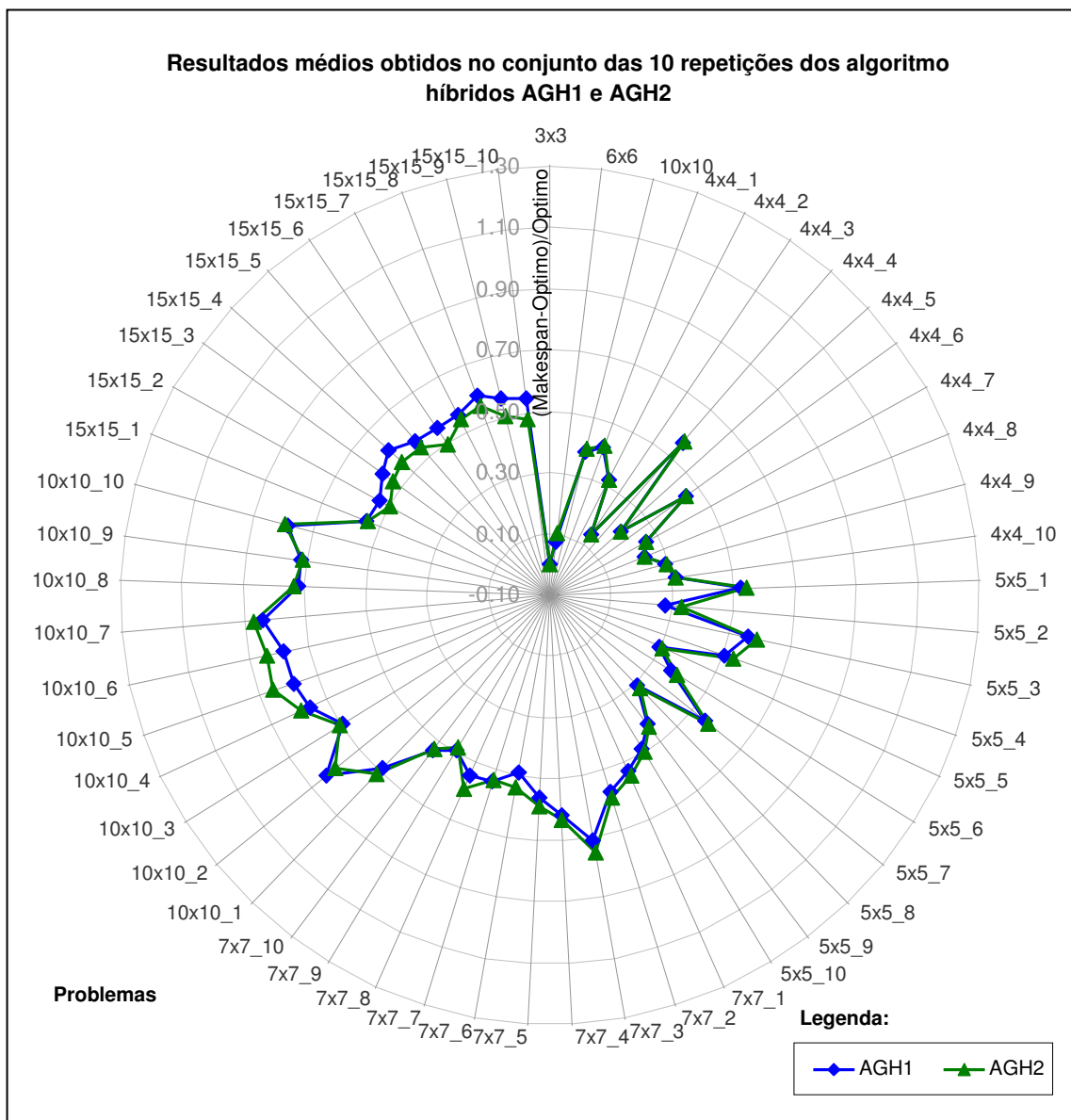
O comportamento do algoritmo AGH2, relativamente aos parâmetros de controlo usados, é muito semelhante ao comportamento do algoritmo AG e do AGH1.

Apesar dos resultados do AGH2 serem pouco influenciados pelo conjunto de parâmetros usado, revela-se uma ligeira melhoria quando usado o segundo e quarto

conjuntos de parâmetros. Como para o AGH1, nota-se uma ligeira vantagem, em média, quando usado o quarto conjunto de parâmetros nos problemas maiores. Ou seja, quando é usada uma população de 100 indivíduos com uma probabilidade de mutação de 0.1.

Como referido anteriormente, estes resultados são semelhantes aos obtidos com os algoritmos AG e AGH1. Esta semelhança, particularmente no que diz respeito ao AG, seria de esperar porque o algoritmo AGH2 consiste na utilização sequencial do algoritmo AG e do algoritmo PT.

Relativamente à performance dos algoritmos híbridos, comparativamente, podemos ver que depois de terminar a avaliação das 100000 soluções, o algoritmo híbrido AGH1 obteve resultados semelhantes aos do algoritmo híbrido AGH2 para problemas pequenos, foi melhor em problemas de média dimensão e perde performance de forma sistemática em problemas maiores. Isto pode ser visto no Gráfico 26, onde apresentamos uma comparação dos resultados médios obtidos pelos dois algoritmos híbridos, no conjunto das 10 repetições, para os 53 problemas testados.



No Gráfico 26, para comparar o resultado dos dois algoritmos híbridos, foram apresentados os resultados obtidos quando usadas populações de 100 indivíduos e uma probabilidade de mutação dos indivíduos de 0.1, uma vez que apesar de uma diferença pouco significativa, este foi o conjunto de parâmetros que revelou melhores resultados nestes algoritmos.

Como referimos anteriormente, pela análise do Gráfico 26, verificamos que à medida que aumenta a dimensão do problema, o algoritmo híbrido AGH1 perde competitividade relativamente ao algoritmo híbrido AGH2.

Tendo presente o comportamento dos algoritmos PT1 e PT2, podemos deduzir uma possível explicação para a perda de competitividade do algoritmo AGH1. O que verificamos com os algoritmos que implementam a procura tabu é que eles, em poucos movimentos/iterações, conseguem grandes melhorias da solução inicial. O algoritmo AGH1 vai efectuar procura tabu para a melhor solução encontrada em cada geração do algoritmo genético, analisando um máximo de 100 soluções de cada vez. Isto significa, que por exemplo, para um problema do tipo 3x3, a procura tabu vai efectuar os 12 movimentos/iterações possíveis, isto tendo em conta que cada solução tem uma estrutura de vizinhos composta por 8 soluções. Já para um problema do tipo 15x15, como cada solução tem uma estrutura de vizinhos composta por 244 soluções, a procura tabu só testará os 100 primeiros vizinhos analisados, escolhendo o melhor deles. Concluindo, para problemas de maior dimensão, acabamos por não aproveitar o potencial da procura tabu. Isto sugere uma possível melhoria no algoritmo AGH1, em que o algoritmo PT deveria analisar o primeiro nível de vizinhança de forma completa ou, pelo menos, de forma proporcional ao tamanho do problema. Por outro lado, o algoritmo AGH2 combina os dois métodos de uma forma que maximiza de forma directa o seu potencial: o algoritmo AG faz a exploração do espaço de soluções e encontra uma zona onde existe uma boa solução; o algoritmo PT depois faz uma pesquisa local exaustiva, aumentando a probabilidade de encontrar a melhor solução nessa zona.

A evolução do comportamento dos algoritmos híbridos AGH1 e AGH2 ao longo das 100000 soluções testadas pode ser analisado nos gráficos 27 a 31 para os problemas 4x4\_4, 5x5\_4, 7x7\_4, 10x10\_4 e 15x15\_4. Os dados apresentados são recolhidos a cada 50 soluções testadas.

Mais uma vez, vamos comparar o resultados destes algoritmos quando usado o quarto conjunto de parâmetros pelas mesmas razões apontadas atrás.

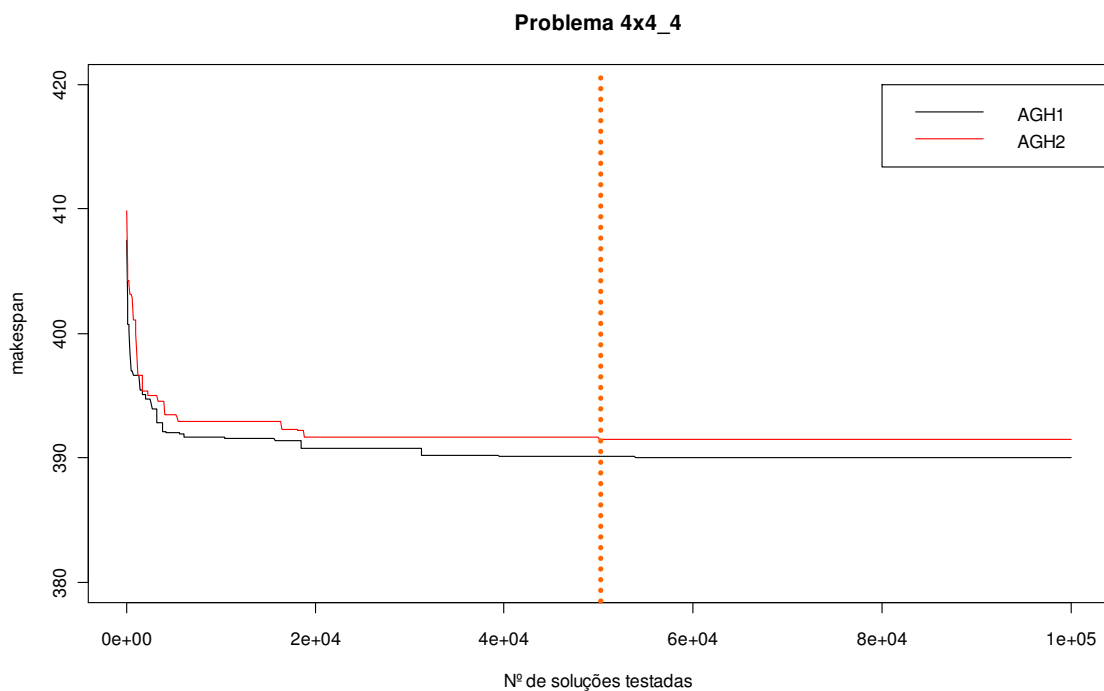


Gráfico 27. Comparação da evolução dos resultados médios obtidos pelo AGH1 e pelo AGH2 no conjunto das 10 repetições, para o problema 4x4\_4 (quando usadas populações de 100 indivíduos e prob. de mutação de 0.1).

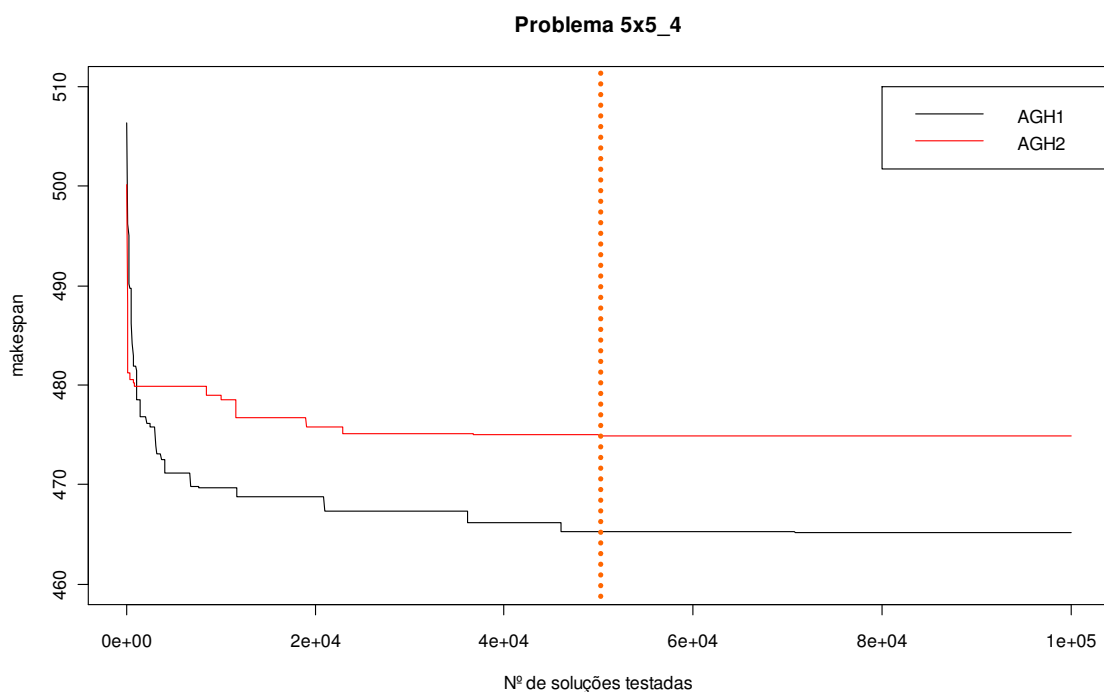


Gráfico 28. Comparação da evolução dos resultados médios obtidos pelo AGH1 e pelo AGH2 no conjunto das 10 repetições, para o problema 5x5\_4 (quando usadas populações de 100 indivíduos e prob. de mutação de 0.1).

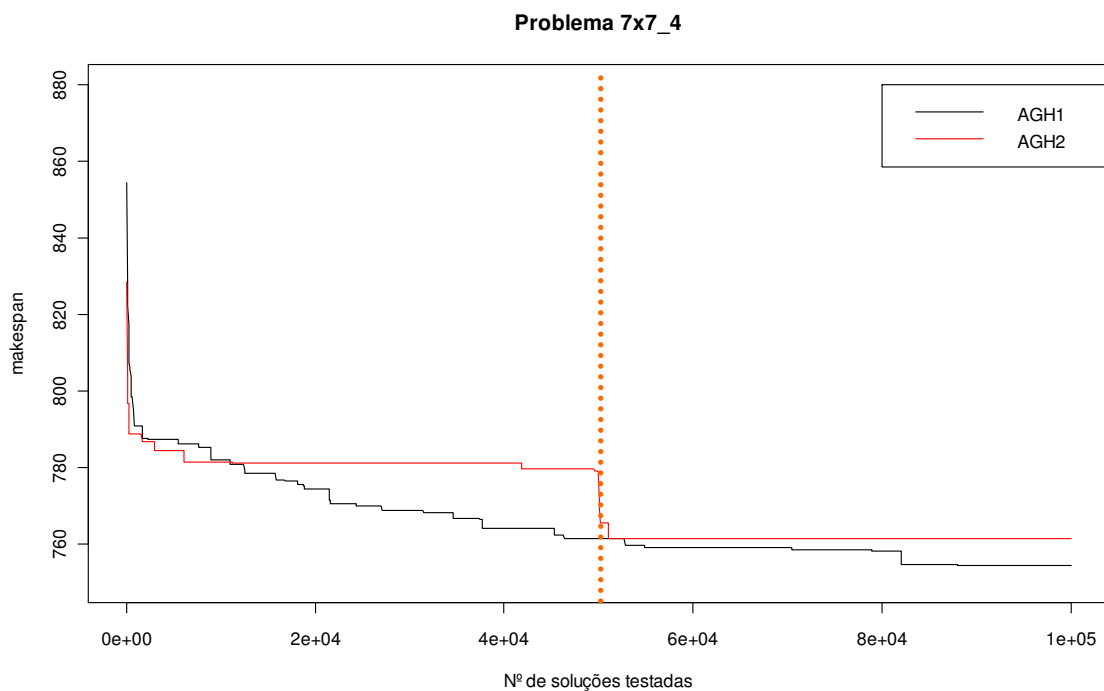


Gráfico 29. Comparação da evolução dos resultados médios obtidos pelo AGH1 e pelo AGH2 no conjunto das 10 repetições, para o problema 7x7\_4 (quando usadas populações de 100 indivíduos e prob. de mutação de 0.1).

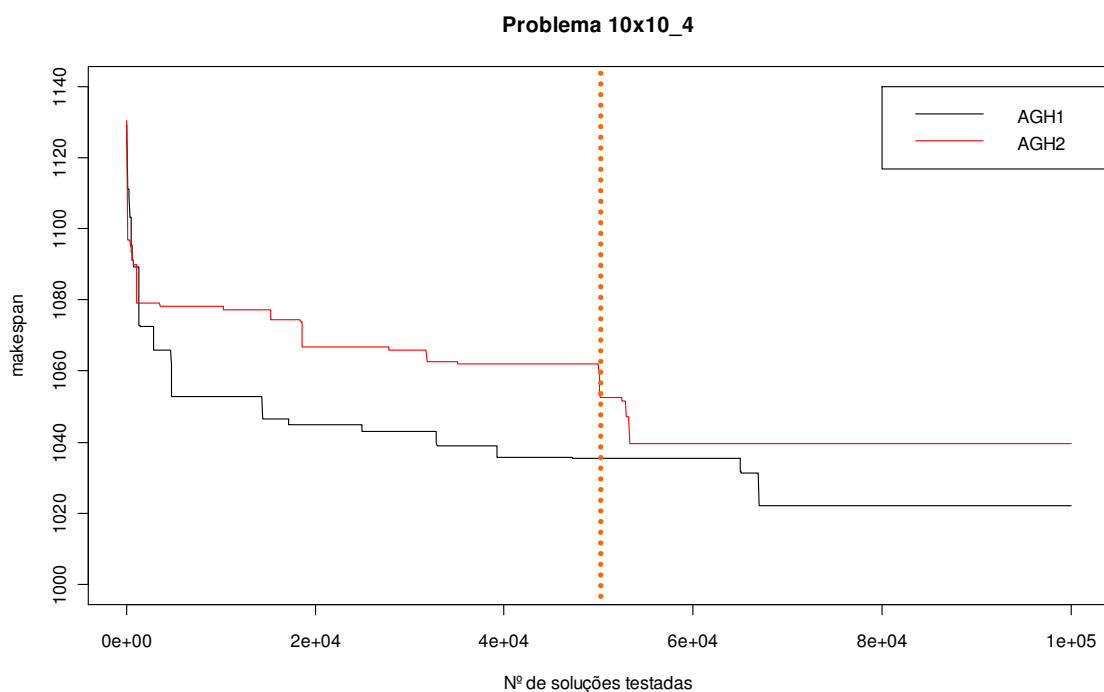


Gráfico 30. Comparação da evolução dos resultados médios obtidos pelo AGH1 e pelo AGH2 no conjunto das 10 repetições, para o problema 10x10\_4 (quando usadas populações de 100 indivíduos e prob. de mutação de 0.1).

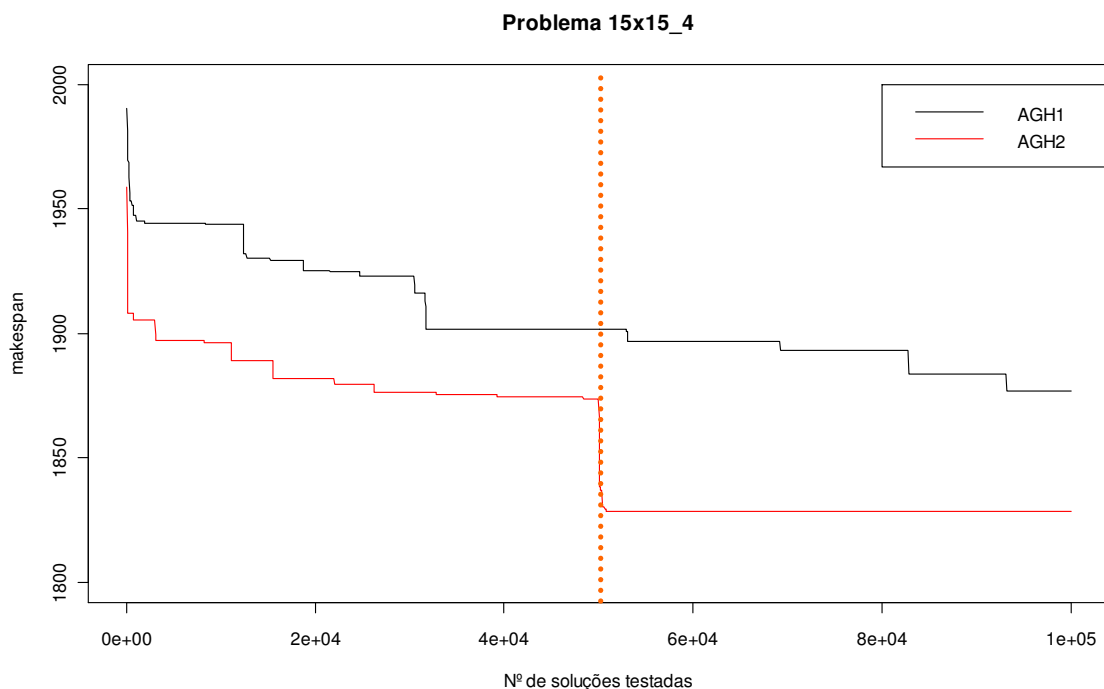


Gráfico 31. Comparação da evolução dos resultados médios obtidos pelo AGH1 e pelo AGH2 no conjunto das 10 repetições, para o problema 15x15\_4 (quando usadas populações de 100 indivíduos e prob. de mutação de 0.1).

Como podemos observar nestes gráficos, o algoritmo híbrido AGH1 geralmente é mais eficiente no processo de procura perdendo no entanto a competitividade nos problemas maiores. Em problemas de dimensão inferior a 7x7 e até em alguns 10x10, verificamos que o AGH1 é mais eficiente no processo de busca praticamente desde a primeira solução analisada. O que se explica pelo uso da procura tabu na melhor solução de cada geração do algoritmo genético. Este comportamento verifica-se em todos os problemas analisados, apesar de na análise dos gráficos tal só ser evidente para os problemas 4x4\_4, 5x5\_4, 7x7\_4, 10x10\_4 e 15x15\_4. No entanto podemos ver o resultado obtido para outros problemas no anexo 9.

Podemos ainda observar que o algoritmo AGH1 aparenta ainda não ter terminado o processo de procura.

Relativamente ao algoritmo híbrido AGH2 verificamos que a introdução da procura tabu após o algoritmo genético ter avaliado 50000 soluções, introduz sempre grandes melhorias na solução inicial, como se pode ver nos gráficos anteriores pelo ponto assinalado com uma recta vertical de cor laranja. Tal como se verificou nos algoritmos PT1 e PT2, durante as primeiras 1000 a 2000 soluções a procura tabu introduz grandes

melhorias mas depois tende a estabilizar o processo de procura. Este resultado indica que este método poderá obter melhores resultados se atribuirmos mais soluções ao algoritmo AG do que à PT. Por outro lado, os resultados apresentados nesta secção sugerem que a combinação do AGH1 com o AGH2, isto é um AG que faz procura tabu à melhor solução encontrada em cada geração e a usar o mesmo método de procura tabu para afinar a solução final, poderá ser promissor.

#### **6.3.4. Análise comparativa do resultado obtido pelos algoritmos PT1, PT2, AG, AGH1 e AGH2**

Na análise comparativa dos 5 algoritmos, só iremos apresentar os resultados obtidos quando usado o quarto conjunto de parâmetros nos métodos baseados em AG, ou seja, usando populações de 100 indivíduos e uma probabilidade de mutação de 0.1. No caso do algoritmo AG este foi o conjunto de parâmetros que mostrou melhor desempenho, e no caso dos algoritmos híbridos apesar de não ser significativo, na verdade também foi com este conjunto de parâmetros que se obtiveram os melhores resultados médios.

Comparando a performance dos cinco algoritmos apresentados, verificamos que para os problemas de menor dimensão como, o 3x3 e os 4x4, os algoritmos PT2, AG, AGH1 e AGH2 obtêm resultados semelhantes após testarem 100000 soluções. No entanto não podemos deixar de salientar que para alguns destes problemas, o AGH1 obtém melhores resultados. Isto pode ser visto no Gráfico 29, onde apresentamos como exemplo o resultado médio obtido pelos 5 algoritmos para o problema 4x4\_4, analisado também em secções anteriores deste capítulo.

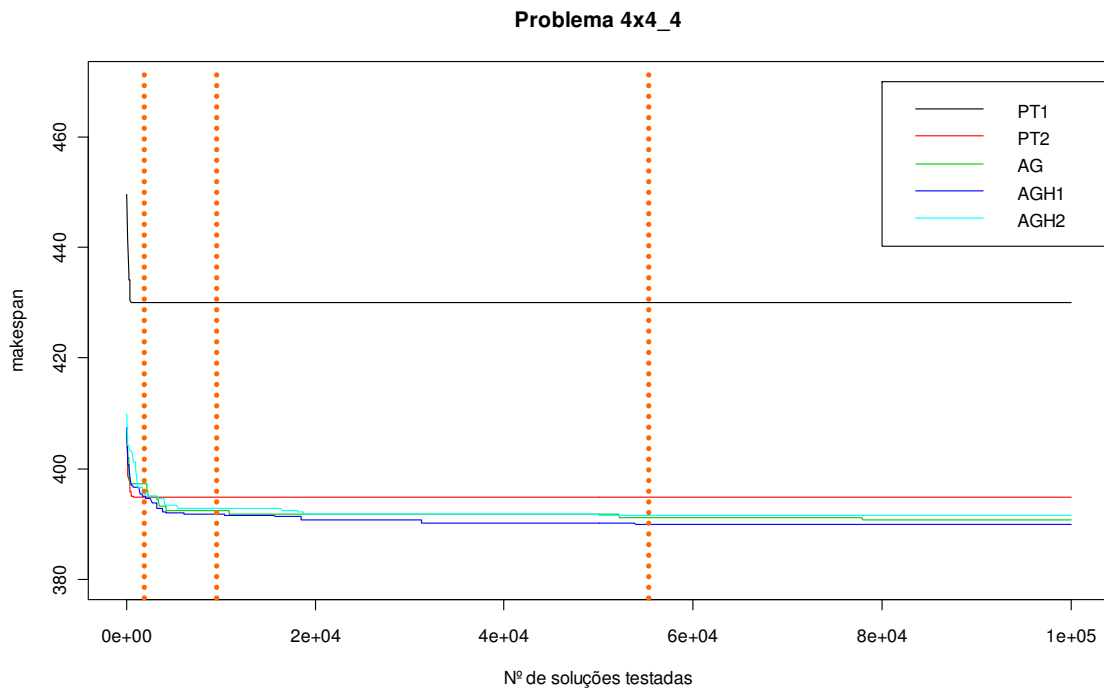


Gráfico 32. Resultados médios obtidos pelos 5 algoritmos nas 10 repetições, quando usadas populações de 100 indivíduos e prob. de mutação de 0.1, nas 100000 soluções avaliadas (valores analisados em intervalos de 50) do problema 4x4\_4.

Uma das características destes 5 algoritmos é a capacidade de se poder parar a qualquer momento o seu processamento devolvendo a melhor solução encontrada até esse momento. Assim, vamos analisar o resultado obtido se tivéssemos que parar o seu processamento após analisar 2000, 10000, e 55000 soluções. Estes pontos foram assinalados no Gráfico 32 por três linhas verticais de cor laranja. Pela análise dos resultados obtidos em cada um desses 3 pontos, e tal como seria de esperar de acordo com a análise individual anteriormente feita aos algoritmos, os resultados poderão ser diferentes.

No Gráfico 32 apresentamos o resultado obtido para o problema 4x4\_4. O comportamento observado é semelhante para os restantes problemas de pequena dimensão, o 3x3 e os 4x4. Podemos por isso dizer que quando pretendemos obter resultados rápidos, até 2000 soluções testadas, o algoritmo PT2 apresenta melhores resultados. Às 10000 soluções testadas a performance do algoritmo híbrido AGH1 é superior ou igual à dos algoritmos PT2, AG e AGH2, o mesmo acontece após analisar 55000 soluções. A performance do algoritmo PT1 é sempre inferior à dos restantes algoritmos.

Para os problemas do tipo 5x5, 6x6 e 7x7, considerados de média dimensão, verificamos que o algoritmo PT2 converge rapidamente para uma boa solução, sendo no entanto ultrapassado, especialmente pelo AGH1, quando se observa um maior número de iterações. Nas primeiras 2000 soluções o PT2 consegue melhorar muito a solução inicial mas depois não se verificam grandes alterações durante as restantes 98000 soluções avaliadas. Como exemplo, no Gráfico 33 podemos ver os resultados obtidos para o problema 5x5\_4.

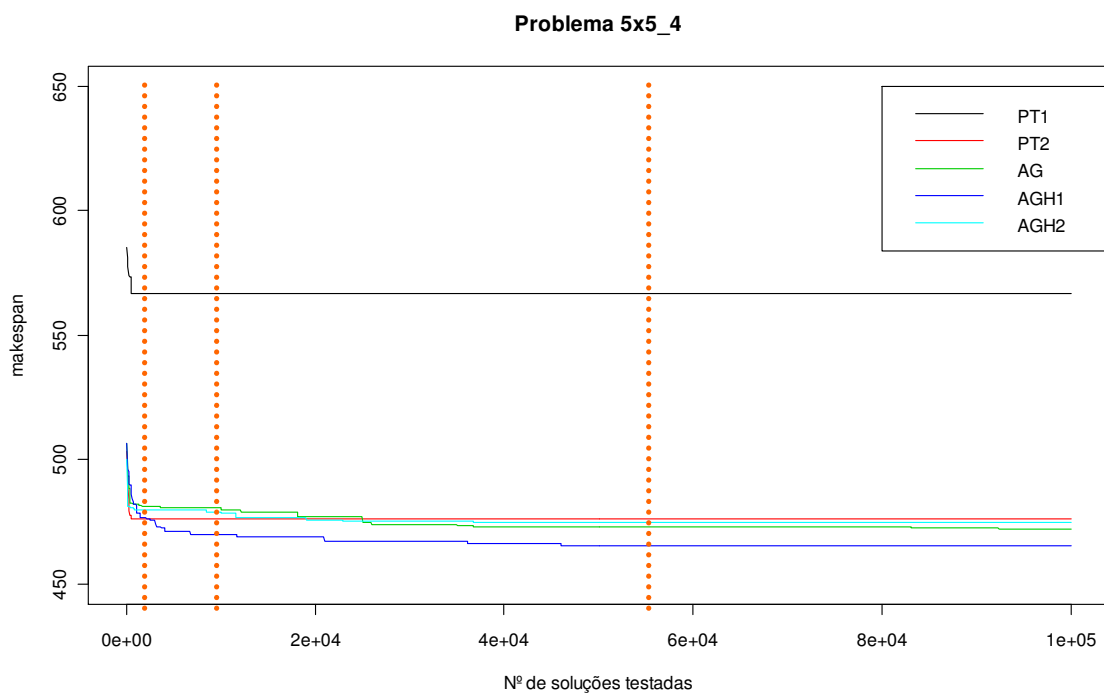


Gráfico 33. Resultados médios obtidos pelos 5 algoritmos nas 10 repetições, quando usadas populações de 100 indivíduos e prob. de mutação de 0.1, nas 100000 soluções avaliadas (valores analisados em intervalos de 50) do problema 5x5\_4.

Como se pode ver pelo Gráfico 33 apresentado acima o algoritmo PT2 é muito rápido a convergir. Por outro lado os algoritmos que usam AGs apesar de serem mais lentos a convergir, a longo prazo apresentam melhores resultados, especialmente o AGH1. Este comportamento é comum à maior parte dos problemas de média dimensão, no entanto podemos observar algumas exceções como é o caso dos resultados obtidos para o problema 7x7\_4, como se pode ver no Gráfico 34.

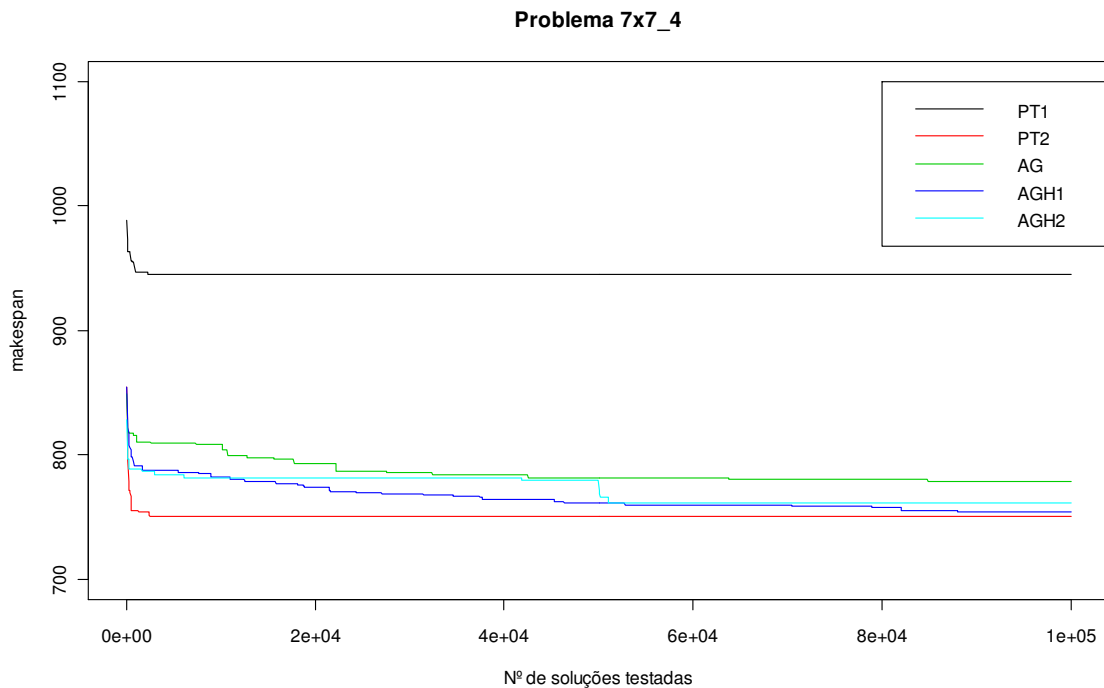


Gráfico 34. Resultados médios obtidos pelos 5 algoritmos nas 10 repetições, quando usadas populações de 100 indivíduos e prob. de mutação de 0.1, nas 100000 soluções avaliadas (valores analisados em intervalos de 50) do problema 7x7\_4.

Da observação do Gráfico 34 podemos concluir que para o problema 7x7\_4, o algoritmo PT2 é o que obtém melhores resultados após avaliar as 100000 soluções, no entanto é de salientar que o algoritmo AG e AGH1 parecem ainda não ter terminado o processo de procura.

Tal como para os problemas de menor dimensão, também nos problemas de média dimensão se verifica que o algoritmo PT2 seria a opção a usar se tivéssemos de interromper o processo de procura da melhor solução numa fase inicial.

Verifica-se que à medida que aumenta a dimensão do problema, o algoritmo PT2 mantém-se competitivo durante mais tempo, como se pode ver no Gráfico 35 onde apresentamos o resultado dos 5 algoritmos para o problema 7x7\_5, e assinalamos com linha vertical de cor laranja o ponto onde foram testadas 2000, 10000 e 5000.

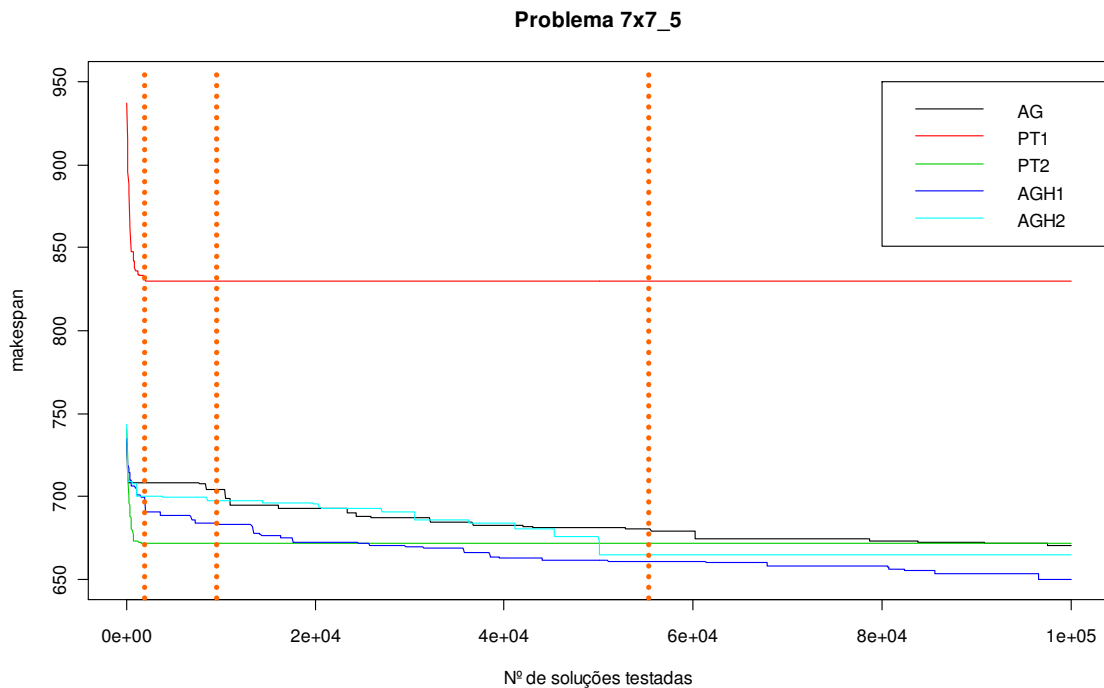


Gráfico 35. Resultados médios obtidos pelos 5 algoritmos nas 10 repetições, quando usadas populações de 100 indivíduos e prob. de mutação de 0.1, nas 100000 soluções avaliadas (valores analisados em intervalos de 50) do problema 7x7\_5.

Como podemos ver pelo Gráfico 35, quando avaliamos 2000 ou 10000 soluções o algoritmo PT2 obtém melhores resultados. Quando avaliamos 55000 soluções o AGH1 consegue obter melhores resultados, mas o AGH2 aproxima-se.

Relativamente aos problemas maiores (os do tipo 10x10 e 15x15) após avaliar as 100000 soluções, o algoritmo AGH1 obteve melhores resultados em 5 dos 10 problemas do tipo 10x10, o AGH2 obteve melhores resultados em 5 dos 10 problemas do tipo 15x15 e nos restantes o PT2 obteve melhores resultados. Nos gráficos 36 a 38 podemos ver os resultados obtidos pelos 5 algoritmos após avaliar 100000 soluções para os problemas 10x10\_4, 10x10\_8 e 15x15\_4.

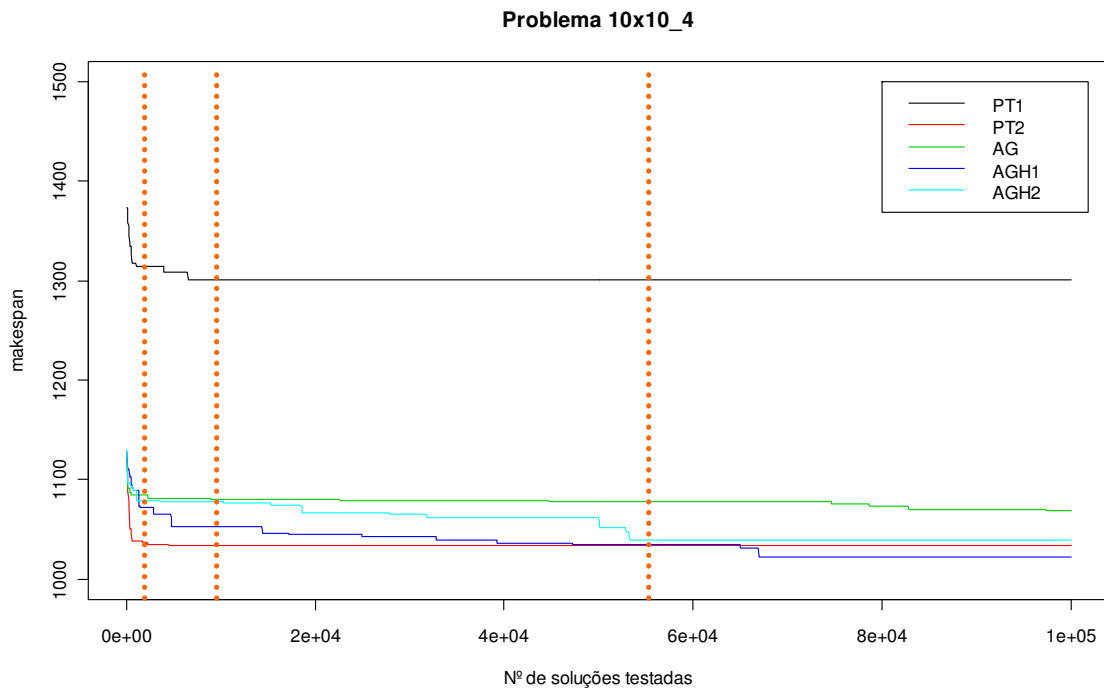


Gráfico 36. Resultados médios obtidos pelos 5 algoritmos nas 10 repetições, quando usadas populações de 100 indivíduos e prob. de mutação de 0.1, nas 100000 soluções avaliadas (valores analisados em intervalos de 50) do problema 10x10\_4.

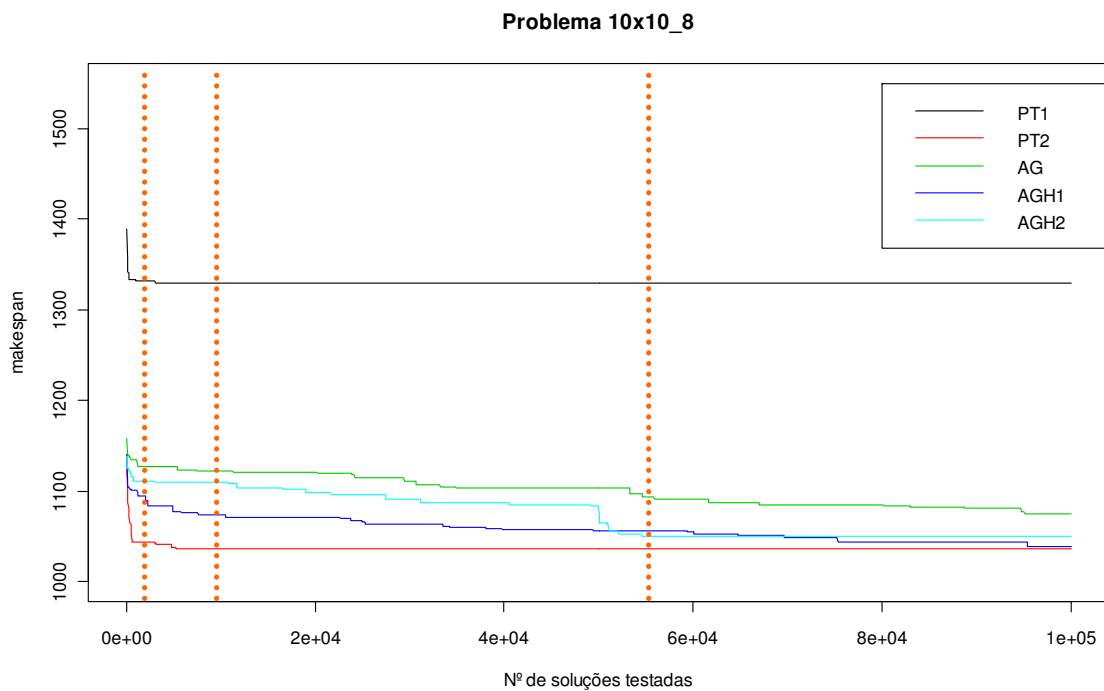


Gráfico 37. Resultados médios obtidos pelos 5 algoritmos nas 10 repetições, quando usadas populações de 100 indivíduos e prob. de mutação de 0.1, nas 100000 soluções avaliadas (valores analisados em intervalos de 50) do problema 10x10\_8.

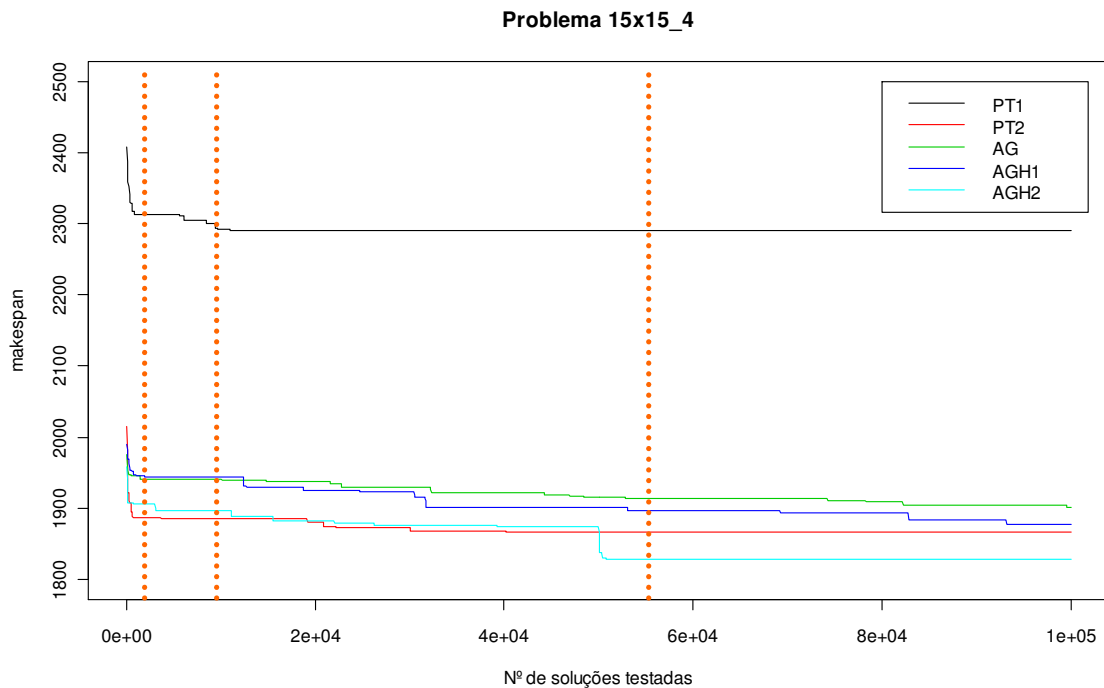


Gráfico 38. Resultados médios obtidos pelos 5 algoritmos nas 10 repetições, quando usadas populações de 100 indivíduos e prob. de mutação de 0.1, nas 100000 soluções avaliadas (valores analisados em intervalos de 50) do problema 15x15\_4.

Os resultados indicam também que nos problemas onde a PT2 obtém melhores resultados, aparentemente o processo de procura dos algoritmos AG, AGH1 e AGH2 ainda não terminou, isto após avaliar as 100000 soluções.

Para confirmar esta possibilidade, decidimos executar novamente os algoritmos PT2, AG, AGH1 e AGH2, para o problema 10x10\_8, onde o algoritmo PT2 ao fim de avaliar 100000 soluções tinha obtido melhor resultado, mas desta vez a avaliar 500000 soluções. No Gráfico 39 apresentamos os resultados médios obtidos no conjunto das 10 repetições dos algoritmos.

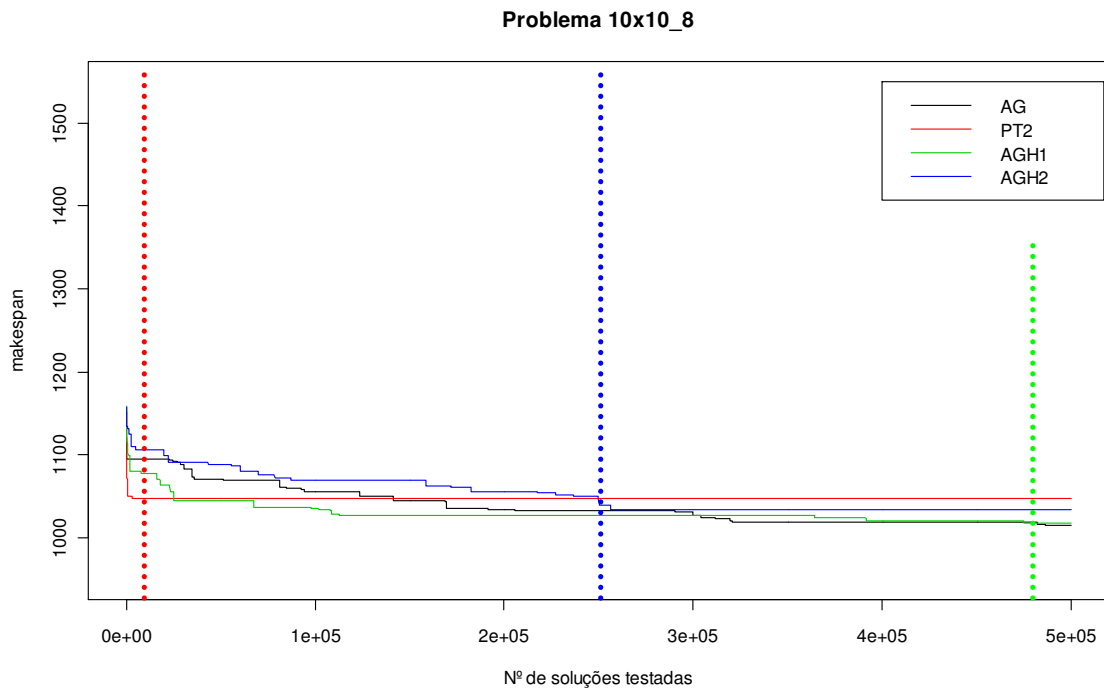


Gráfico 39. Resultados médios obtidos pelos algoritmos PT2, AG, AGH1 e AGH2, nas 10 repetições, quando usadas populações de 100 indivíduos com prob. de mutação de 0.1, nas 500000 soluções avaliadas (valores analisados em intervalos de 50) do problema 10x10\_8.

Os resultados apresentados no Gráfico 39, para o problema 10x10\_8, confirmam que os algoritmos AG e AGH1 ainda não tinham terminado o processo de procura ao fim de 100000 soluções. Estes resultados também confirmam que o algoritmo PT2 é muito rápido a identificar uma zona do espaço onde se encontra uma boa solução, mas depois não consegue dar o salto para outras zonas do espaço de procura. Na execução do algoritmo AGH2, verificamos que a entrada da procura tabu para refinar a solução obtida pelo AG, leva sempre a uma melhoria da solução, mas esse processo de melhoria só é relevante nas primeiras soluções avaliadas. A partir de uma dada altura a melhoria é insignificante ou inexistente. Neste gráfico podemos ainda observar que os algoritmos AG e AGH1 mostraram-se mais competitivos, e, mesmo assim, aparentam ainda não ter terminado o processo de procura. Estes resultados vêm mais uma vez sugerir uma variante do AGH1 e outra do AGH2 que aproveitem melhor as capacidades do AG e da PT. A nova variante do AGH1 poderia executar a procura tabu para a melhor solução encontrada em cada geração durante um número maior de iterações, e a variante do AGH2 deveria dar mais soluções ao AG para avaliação.

Alem disto, relativamente aos problemas de grande dimensão, tal como já se tinha verificado antes, nos problemas de pequena e média dimensão, quando se pretendem resultados e não se está disposto a esperar muito tempo, o algoritmo PT2 apresenta melhores resultados. Também como já se tinha observado anteriormente, em problemas de média dimensão, observamos que o algoritmo PT2 se mantém competitivo durante mais tempo à medida que aumenta a dimensão do problema. Como exemplo veja-se no Gráfico 36 onde reportamos o resultado dos 5 algoritmos, para o problema 10x10\_4, onde foram assinalados os pontos de avaliação das 2000, 10000 e 55000 soluções através de uma linha vertical de cor laranja.

Pela análise do Gráfico 36, podemos ainda dizer que se tivéssemos de parar a execução do algoritmo após avaliar 2000, 10000 ou 55000 soluções, o algoritmo PT2 seria o que iria obter o melhor resultado, apesar de sabermos que para este problema em concreto, no final das 100000 soluções avaliadas, o algoritmo AGH1 obtém melhores resultados. Este comportamento, do algoritmo PT2, observa-se em quase todos os problemas do tipo 10x10 e 15x15.

Comparando os resultados dos algoritmos híbridos, verificamos que nos problemas pequenos e médios, como já tinha sido referido na Secção 6.3.3, o AGH1 obtém melhores resultados que o AGH2, no entanto isto não se verifica para os problemas maiores. À medida que aumenta a dimensão do problema o AGH1 vai perdendo competitividade para o AGH2. Este resultado é visível nos gráficos gerais 1, 2, 3 e 4. Relativamente ao AG simples, podemos observar que os algoritmos híbridos para a generalidade dos problemas obtiveram melhores resultados.

Através dos resultados obtidos pelo algoritmo AGH2, podemos verificar que a entrada da PT geralmente resulta em melhorias significativas.

Relativamente ao tempo necessário para cada algoritmo processar as 100000 soluções, podemos ver a comparação dos tempos médios dos 5 algoritmos, no Gráfico 40.

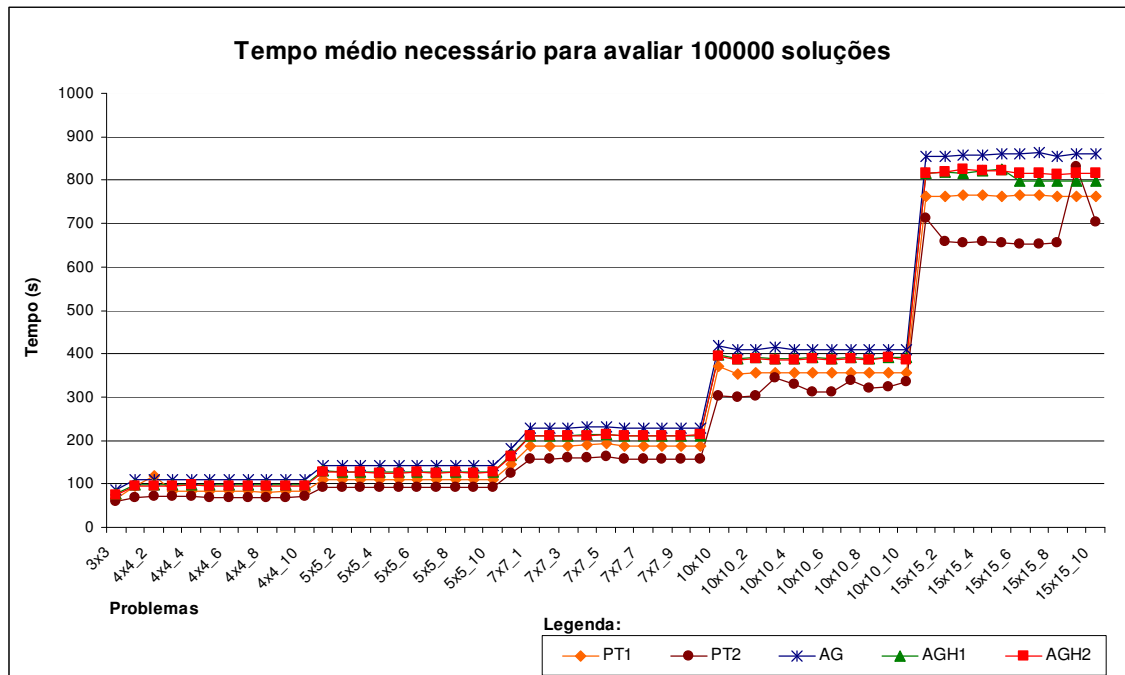


Gráfico 40. Comparação dos tempos médios necessários para processar as 100000 soluções quando usadas populações de 100 indivíduos com prob. de mutação de 0.1.

Como podemos ver pelos resultados apresentados no Gráfico 40, os algoritmos PT1 e PT2 são mais rápidos a processar as 100000 soluções. O AG é o que demora mais tempo, mas na verdade não há diferenças significativas entre os 5 algoritmos.

Relativamente ao tempo médio necessário até que cada um dos algoritmos encontre a melhor solução de 100000 avalidas, verificamos que os algoritmos de procura tabu são mais rápidos como podemos ver pelo Gráfico 41.

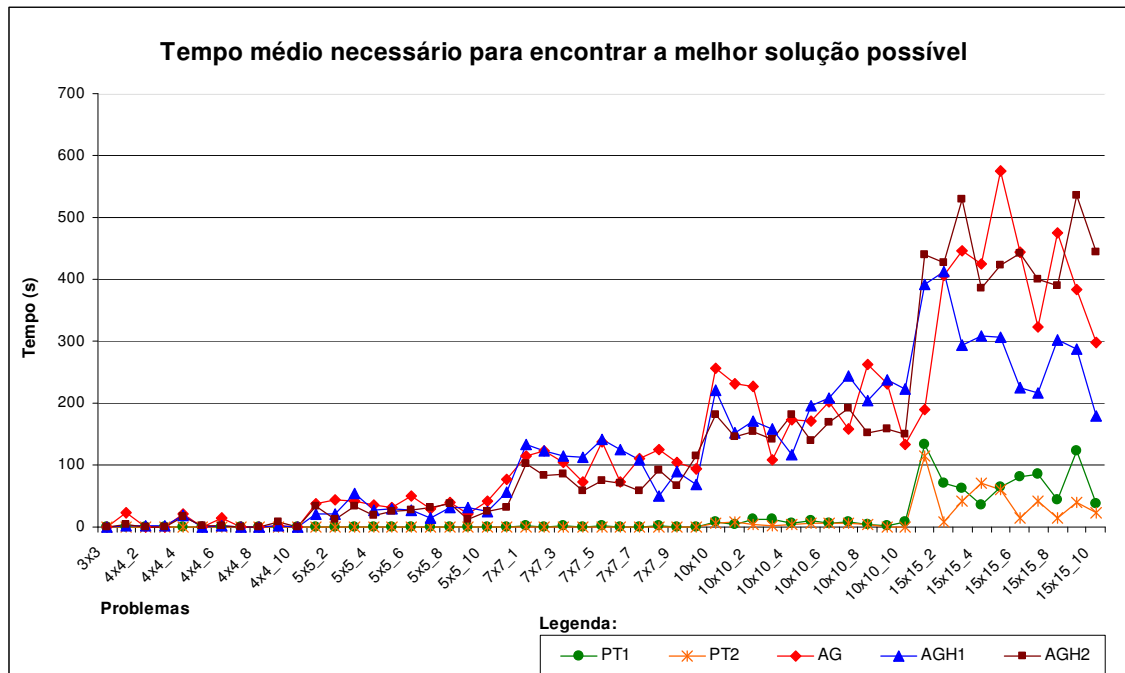


Gráfico 41. Comparação dos tempos médios necessários até os cinco algoritmos encontrarem a melhor solução das 100000 avaliadas, quando usadas populações de 100 indivíduos com prob. de mutação de 0.1.

Tendo em conta a evolução dos resultados obtidos pelos 5 algoritmos, e sabendo que os algoritmos PT1 e PT2 convergem muito cedo, facilmente nos apercebemos da razão pela qual estes algoritmos são mais rápidos a encontrar a melhor solução.

Relativamente aos outros algoritmos, como eles continuam a introduzir melhorias na solução durante as 100000 soluções avaliadas, os tempos apresentados são o tempo médio para avaliar a maior parte das soluções.

## 7. Conclusões

### 7.1. Objectivos

Neste trabalho pretendeu-se verificar se é possível obter melhores resultados usando métodos de aproximação híbridos do que usados os métodos puros, numa versão simples do problema de *job-shop scheduling*.

Escolheram-se os algoritmos genéticos por serem capazes de percorrer todo o espaço de soluções admissíveis, apesar da dificuldade que têm em encontrar os óptimos locais. Escolheu-se também a Procura Tabu porque, apesar de ter dificuldades em explorar todo o espaço de soluções admissíveis, tem facilidade em encontrar óptimos locais que se encontram perto de soluções dadas.

Assim, neste trabalho pretendemos explorar os pontos fortes de cada um destes métodos para verificar se conseguem obter melhores resultados quando combinados do que quando usados individualmente.

Neste contexto, foram implementados cinco algoritmos, dois algoritmos de procura tabu simples, um algoritmo genético simples e dois algoritmos híbridos que combinam o algoritmo genético e a procura tabu simples.

Para relembrar, o algoritmo PT1 implementa uma procura tabu simples, em que a solução inicial é gerada aleatoriamente. O algoritmo PT2 também implementa uma procura tabu simples, diferindo do algoritmo anterior por a solução inicial ser a melhor de 100 geradas aleatoriamente. O algoritmo AG implementa um algoritmo genético simples, e foi executado com quatro variantes dos parâmetros de controlo. O algoritmo AGH1 é um algoritmo genético híbrido, que usa o algoritmo AG e, à melhor solução de cada geração, aplica o algoritmo PT como um operador de mutação, fazendo procura local durante 100 soluções. O algoritmo AGH2 é também um algoritmo híbrido, que executa o algoritmo AG e o algoritmo PT1 de forma sequencial, sendo a solução encontrada pelo AG usada como solução inicial do algoritmo PT1.

Para que a comparação fosse o mais justa possível, decidimos que todos os algoritmos implementados iriam testar o mesmo número de soluções. O nosso objectivo é assim o

de comparar a eficácia dos algoritmos na forma como exploram o espaço de soluções e não a eficiência da implementação dos algoritmos.

## **7.2. *Resumo dos resultados***

Como dito acima, o que pretendíamos era determinar se, tal como defendem alguns autores, os algoritmos híbridos obtêm melhores resultados do que os algoritmos simples.

Da análise dos resultados obtidos pelos algoritmos PT1 e PT2 verificamos que eles conseguem, em poucos movimentos/iterações, introduzir grandes melhorias na solução inicial. Da sua comparação verificamos que o algoritmo PT2 obtém sempre melhores resultados do que o algoritmo PT1, o que nos leva a concluir que tal como revelam alguns autores, a qualidade da solução inicial influencia de forma muito significativa a qualidade da solução final.

Da análise dos resultados obtidos pelo AG, e comparando os quatro conjuntos de parâmetros, concluímos que a dimensão da população é um factor importante na performance do AG. Por outro lado, os algoritmos AGH1 e AGH2 mostraram ser pouco influenciados pelo conjunto de parâmetros usado, de acordo com os resultados obtidos.

Considerando os resultados obtidos pelos dois algoritmos híbridos, observamos que depois de avaliar as 100000 soluções, o algoritmo AGH1 obteve resultados iguais aos do AGH2 nos problemas de pequena dimensão, foi melhor nos de média dimensão e perdeu competitividade à medida que aumenta a dimensão do problema. Dado que o número de soluções testadas pela PT no AGH1 é constante, independentemente da dimensão do problema, isto pode ser explicado pelo facto da procura efectuada pela PT ficar incompleta. Ou seja, a PT não avalia um número suficiente de soluções de forma a introduzir melhorias significativas na solução inicial. Isto sugere-nos uma variante do AGH1 com a PT a avaliar um número de soluções proporcional à dimensão do problema. No algoritmo AGH2 observamos que a entrada da procura tabu traz sempre grandes melhorias à solução encontrada pelo AG.

Após comparar os resultados dos cinco algoritmos observamos que nos problemas pequenos se obtêm resultados equivalentes, apesar de haver ligeira vantagem para o

algoritmo AGH1. Nos problemas de média dimensão o algoritmo PT2 converge rapidamente mas geralmente o AGH1 é melhor ao fim das 100000 soluções.

Nos problemas de grande dimensão, verificou-se que nas 11 instâncias do tipo 10x10, o algoritmo PT2 obtém melhores resultados em 5 problemas e que o algoritmo AGH1 obtém melhores resultados noutros 5 problemas (o AG simples obtém melhores resultados no restante problema). Nos dez problemas do tipo 15x15, vemos que o algoritmo PT2 obtém melhores resultados em cinco problemas e o algoritmo AGH2 obtém melhores resultados nos restantes cinco problemas.

Verificamos também, que exceptuando nos problemas do tipo 15x15, de maior dimensão, o algoritmo PT2 converge muito cedo, por outro lado vemos que o algoritmo genético e os algoritmos híbridos, especialmente o AGH1, aparentam ainda não ter terminado a procura. Isto sugere-nos uma variante do algoritmo AGH2, que dê mais soluções ao AG para avaliar.

Da análise da evolução dos resultados dos cinco algoritmos observamos que os algoritmos que usam AGs apesar de serem mais lentos a convergir, a longo prazo apresentam-se mais promissores, especialmente o AGH1. Verificamos também que o algoritmo PT2 é o mais competitivo quando se pretendem resultados rápidos. Nos problemas onde o algoritmo PT2, após avaliar as 100000 soluções, obteve melhores resultados, os algoritmos baseados em AGs aparentam ainda não terem terminado o processo de procura.

Relativamente aos tempos de computação verificamos que todos os algoritmos têm tempos semelhantes quando avaliam o mesmo número de soluções, mas a procura tabu é mais rápida a encontrar a melhor solução possível de entre 100000 avaliadas.

### **7.3. Contribuições**

Neste trabalho fizemos uma avaliação empírica extensa comparando dois métodos híbridos, baseados em algoritmos genéticos e procura tabu, com os métodos quando aplicados individualmente.

Os nossos resultados mostram que, embora a combinação permita obter melhores resultados em muitos casos, existem casos em que os algoritmos base são melhores.

Assim, verificamos, por exemplo, que o uso da procura tabu como um operador de mutação de um algoritmo genético deve ser ajustado à dimensão do problema.

Numa perspectiva diferente, a análise da evolução dos resultados obtidos, indica que, se precisarmos de uma solução boa em pouco tempo, então devemos optar por usar o algoritmo de procura tabu, pois ele é muito rápido a encontrar uma boa solução. Por outro lado, a longo prazo os algoritmos híbridos apresentam-se mais promissores.

Confirmando os resultados obtidos em estudos anteriores, comprovamos também que a qualidade da solução final encontrada pela procura tabu depende de ele anteriormente ter identificado uma zona do espaço de procura onde se encontram soluções boas.

Ainda relativamente ao método de procura tabu, verificamos que depois de identificar uma zona promissora do espaço de soluções, ela traz grandes mais valias até cerca 10000 soluções avaliadas, mas que, depois disso, é pouco relevante.

Finalmente, e embora o número de variantes testadas seja pequeno, verificamos que o tamanho da população é um parâmetro de controlo importante para a qualidade final da solução encontrada pelo algoritmo Genético. No entanto, observamos também que este efeito é menos relevante na aplicação de algoritmos híbridos.

#### **7.4. Trabalho futuro**

Da análise dos resultados obtidos pelos 5 algoritmos, nos 53 problemas testados, e apesar dos resultados positivos que foram obtidos, verificamos que a forma como os algoritmos híbridos combinam a procura tabu e os algoritmos genéticos podem ser melhoradas, de forma a maximizar os pontos fortes de cada um.

No AGH1 pretendemos colocar a PT a avaliar um número de soluções dependente da dimensão do problema. Por exemplo, o número de soluções poderia ser dado por *jobs* x máquinas x 10, ou mínimo 1000.

Pretendemos também criar um novo algoritmo híbrido, variante do AGH1, que em vez aplicar a procura local à melhor solução de cada geração do AG, o faça à melhor solução testada durante toda a procura feita pelo AG e à qual ainda não foi aplicada a procura local. Alternativamente, a procura local pode ser aplicada à solução analisada há mais tempo. A ideia é criar uma lista tabu de soluções antes analisadas pela PT. Também neste algoritmo o número de soluções avaliadas pela PT deveria ser calculado

com base na dimensão do problema. Por exemplo através do produto do número de *jobs* pelo número de máquinas por um número mínimo de iterações.

No AGH2 pretendemos colocar a PT a avaliar menos soluções, pois a mais valia que se obtém no seu uso é muito significativa nas primeiras 1000 soluções, mas depois tende a estabilizar ou melhorar pouco.

## Bibliografia

- [Moreira, na] M. Rosário Moreira (na), “Caracterização do processo de produção job shop”, UP-FEP.
- [Baker, 1974] Kenneth R. Baker (1974), “Introduction to Sequencing and Scheduling”, John Wiley & Sons Inc, Capítulo 7.
- [Merbach, 1981] Mueller-Merbach (1981), “Heuristics and their Design: a Survey”, European Journal of Operational Research, Vol.8 pp. 1-23.
- [Chase e Aquilano, 1989] Richard B. Chase e Nicholas J. Aquilano (1989), “Gestão da Produção e das Operações”, Monitor, Capítulo 3 e 14.
- [Goldberg, 1989] David E. Goldberg (1989), “Genetic Algorithms in Search, Optimization, and Machine Learning”, Addison-Wesley Publishing Company, Inc.
- [Yamada e Nakano, 1992] Takeshi Yamada e Ryohei Nakano (1992), “A Genetic Algorithm Applicable to Large-Scale Job shop-Shop Problems”, Elsevier – pp. 281-290.
- [Reeves, 1993] Colin R. Reeves (1993), “Modern Heuristic Techniques for Combinatorial Problems”, Blackwell Scientific Publications.
- [Bruns, 1993] Ralf Bruns (1993), “Direct Chromosome Representation and Advanced Genetic Operators for Production Scheduling“, In Forrest ICGA93 , pp. 352-359.
- [Al-Attar, 1994] Akeel Al-Attar (1994), “A Hybrid GA-Heuristic Search Strategy”, AI Expert.
- [Soares et al., 1995] Carlos Soares, Gusztáv E. Eiben e Joost N. Kok (1995), “Two Representations for the Simple and Enhanced JSS”, Utrecht University – Department of Computer Science.

- [Courtois et al., 1996] Alain Courtois, Chantal Martin-Bonnefous e Maurice Pillet (1996), “Gestão da Produção”, Lidel Edições Técnicas, Capítulo 2.
- [Yagiura e Ibaraki, 1996] Mutsunori Yagiura e Toshihide Ibaraki (1996), “Metaheuristics as Robust and Simple Optimization Tools”, Proc. IEEE International Conference on Evolutionary Computation, pp.541-546.
- [Mitchel, 1997] Tom M. Mitchell (1997), “Machine Learning”, McGraw-Hill, Cap. 9.
- [Land e Gaalman, 1997] Martin J. Land e Gerard J.C. Gaalman (1997), “The performance of workload control concepts in job shops: Improving the release method”, International Journal of Production Economics.
- [Hendry et al., 1997] L.C. Hendry, B.G. Kingsman e P. Cheung (1997), “The effect of workload control (WLC) on performance in make-to-order companies”, Journal of Operations Management.
- [Glover e Laguna, 1997] Fred Glover e Manuel Laguna (1997), "Tabu Search", Kluwer Academic Publishers.
- [Colorni et al., 1998] Alberto Colorni, Marco Dorigo e Vittorio Maniezzo (1998), “Metaheuristics for High School Timetabling”, Kluwer Academic Publishers – Computational Optimization and Applications 9 pp. 275-298.
- [Ikeuchi et al., 1998] Tomoya Ikeuchi, Yoshitomo Ikkai, Dai Araki, Takenao Ohkawa e Norihisa Komoda (1998), “Project Scheduling using a Genetic Algorithm with adaptable Changing Genetic Operators”, Scripta Technica – Electrical Engineering in Japan, Vol. 124 N°2.
- [Jones e Rabelo, 1998] Albert Jones e Luis C. Rabelo (1998), "Survey of Job shop Scheduling Techniques", Gaithersburg - National Institute of Standards and Technology.

- [Bierwirth e Mattfeld, 1999] Christian Bierwirth e Dirk C. Mattfeld (1999), “Production Scheduling and Rescheduling with Genetic Algorithms”, *Evolutionary Computation*, 7 pp. 1-17.
- [Jain e Meeran, 1999] A.S. Jain e S. Meeran (1999), “Theory and Methodology Deterministic job shop scheduling: Past, Present and Future”, *Elsevier-European Journal of Operational Research*, 113 pp. 390-434.
- [Liaw, 2000] Ching-Fang Liaw (2000), “A hybrid genetic algorithm for the open shop scheduling problem”, *Elsevier – European Journal of Operation Research*, 124 pp. 28-42.
- [Ponnambalam et al., 2000] S.G. Ponnambalam, P. Aravindan e S.V. Rajesh (2000), “A Tabu Search Algorithm for Job Shop Scheduling”, *Springer – Advanced Manufacturing Technology* 16 pp. 765-771.
- [Tsujimura et al., 2001] Yasuhiro Tsujimura, Yuichiro Mafune e Mitsuo Gen (2001), “Effects of Symbiotic Evolution in Genetic Algorithms for Job shop-Shop Scheduling”, *IEEE Computer Society*.
- [Binato et al, 2002] S. Binato, W.J. Hery, D.M. Loewenstern e M.G.C. Resende (2002), "A GRASP for Job shop Scheduling", *Kluwer - Essays and Surveys on Metaheuristics*, pp. 58-79.
- [Mahmood e Hyasat, 2002] A. Mahmood e K. Hyasat (2002), “Scheduling Distributed Real-Time Tasks with Tabu Search Algorithm”, *International Journal of Computer System Signal*, Vol. 3 Nº 2 .
- [Talbi, 2002] E.G. Talbi (2002), “A Taxonomy of Hybrid Metaheuristics”, *Kluwer Academic Publishers - Journal of Heuristics*, 8: pp. 541–564.
- [Wang e Zheng, 2002] L. Wang e D.-Z. Zheng (2002), "A modified Genetic Algorithm for Job shop Sheduling", *Springer - Advanced Manufacturing Technology*, 20 pp. 72-76.

- [Bael e Rijckaert, 2002] P. Van Bael e M. Rijckaert (2002), “Specific Knowledge of the Job shop Scheduling Problem Incorporated in Local Search, How Good is it?”, K.U. Leuven – Chemical Engineering Department.
- [Cigolini e Staudacher, 2002] R. Cigolini e A. Portioli-Staudacher (2002), “An experimental investigation on workload limiting methods within ORR policies in a job shop environment”, *Production Planning & Control*, Vol. 13 N° 7 pp. 602-613.
- [Han e Kendall, 2003] Limin Han e Graham Kendall (2003), “Guided Operators for a Hyper-Heuristic Genetic Algorithm”, *Springer*, Vol. 2903 pp. 807-820.
- [Glibovets e Medvid, 2003] N. N. Glibovets e S. A. Medvid (2003), "Genetic Algorithms used to solve Scheduling Problems", *Cybernetics and System Analysis*, Vol. 39 N°. 1.
- [Glover e Kochenberger, 2003] Fred Glover e Gary A. Kochenberger (2003), “Handbook of Metaheuristics”, Kluwer Academic Publishers.
- [Tijms, 2003] Arjan Tijms (2003), “Attacking the schedule problem with a Genetic Algorithm”, *University of Leiden - Inst. of Adv. Computer Science*.
- [Park et al., 2003] Byung Joo Park, Hyung Rim Choi e Hyun Soo Kim (2003), "A hybrid genetic algorithm for the job shop scheduling problems", *Elsevier - Computers & Industrial Engineering*, 45 pp. 597-613.
- [Nearchou, 2004] Andreas C. Nearchou (2004), "The effect of various operators on the genetic search for large scheduling problems", *Elsevier-International journal of Production Economics*, 88 pp. 191-203.
- [Fontes e Moreira, 2004-a] Dalila B.M.M. Fontes e M. Rosário Moreira (2004-a), “Apresentação de Projecto de Tese: Simulação de um processo produtivo job shop”, UP-FEP.

- [Fontes e Moreira, 2004-b] Dalila B.M.M. Fontes e M. Rosário Moreira (2004-b), “Apresentação de Projecto de Tese: Simulação de um processo produtivo flow shop”, UP-FEP.
- [Geyik e Cedimoglu, 2004] F. Geyik e I. H. Cedimoglu (2004), “The strategies and parameters of tabu search for job-shop scheduling”, Kluwer Academic Publishers - Journal of Intelligent Manufacturing, 15 pp. 439-448.
- [Mattfeld e Bierwirth, 2004] Dirk C. Mattfeld e Christian Bierwirth (2004), "Na efficient genetic algorithm for job shop scheduling with tardiness objectives", Elsevier-European Journal of Operational Research, 155 pp. 616-630.
- [Ombuki e Ventresca, 2004] Beatrice M. Ombuki e Mario Ventresca (2004), "Local Search Genetic Algorithm for the Job shop Scheduling Problem", Kluwer - Applied Intelligence, 21 pp. 99-109.
- [Scrich et al., 2004] Cintia Rigão Scrich, Vinícius Amaral Armentano e Manuel Laguna (2004), “Tardiness minimization in a flexible job shop: A tabu search approach”, Kluwer Academic Publishers – Journal of Intelligent Manufacturing, 15 pp. 103-115.
- [Soares, 2004] Carlos Soares (2004), “Computação Evolutiva e Algoritmos Genéticos”, B18 MADSD.
- [Wang e Wu, 2004] Hsiao-Fan Wang e Kuang-Yao Wu (2004), "Hybrid genetic algorithm for optimization problems with permutation property", Elsevier-Computers & Operations Research, 31 pp. 2453-2471.
- [Azimi, 2005] Zahra Naji Azimi (2005), “Hybrid heuristic for Examination Timetabling problem”, Elsevier – Applied Mathematics and Computation, 163 pp. 705-733.
- [Gonçalves et al., 2005] José Fernando Gonçalves, Jorge José de Magalhães Mendes e Maurício J. C. Resende (2005), "A hybrid genetic algorithm for the job shop scheduling problem", Elsevier - European Journal of Operational Research, 167

pp. 77-95.

- [Nowicki e Smutnicki, 2005] Eugeniusz Nowicki e Czeslaw Smutnicki (2005), “An advanced tabu search algorithm for the job shop problem”, Springer – Journal of Scheduling, 8 pp. 145-159.
- [Watanabe et al., 2005] Masato Watanabe, Kenichi Ida e Mitsuo Gen (2005), “A genetic algorithm with modified crossover operator and search area adaptation for the job-shop scheduling problem”, Elsevier - Computers & Industrial Engineering 48 pp. 743–752.
- [Hasan et al., 2007] S. M. Kamrul Hasan, Ruhul Sarker e David Cornforth (2007), “Hybrid Genetic Algorithm for Solving Job-Shop Scheduling Problem”, IEEE Computer Society - 6th IEEE/ACIS International Conference on Computer and Information Science.
- [Zhang et al., 2007] ChaoYong Zhang, PeiGen Li, ZaiLin Guan e YunQing Rao (2007), “A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem”, Elsevier – Computers & Operations Research 34 pp. 3229 – 3242.
- [Vilcot e Billaut, 2008] Geoffrey Vilcot e Jean-Charles Billaut (2008), “A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem ”, Elsevier – European Journal of Operational Research, 190 pp. 398-411.

## **Anexos**

**Anexo 1 – Resultados iniciais dos parâmetros de controlo do algoritmo genético**

| Benchmark Problem | Máquinas | Jobs | Nº Sol. Testadas | popSize | Pc  | Pm   | Nº Repetições | Média  | Max  | Min  |
|-------------------|----------|------|------------------|---------|-----|------|---------------|--------|------|------|
| 3x3               | 3        | 3    | 20000            | 50      | 0.8 | 0.10 | 10            | 26     | 26   | 26   |
| 3x3               | 3        | 3    | 20000            | 50      | 0.8 | 0.01 | 10            | 26     | 26   | 26   |
| 3x3               | 3        | 3    | 20000            | 50      | 1   | 0.10 | 10            | 26     | 26   | 26   |
| 3x3               | 3        | 3    | 20000            | 100     | 0.8 | 0.10 | 10            | 26     | 26   | 26   |
| 3x3               | 3        | 3    | 20000            | 100     | 0.8 | 0.01 | 10            | 26     | 26   | 26   |
| 3x3               | 3        | 3    | 20000            | 100     | 1   | 0.10 | 10            | 26     | 26   | 26   |
| tai4x4            | 4        | 4    | 20000            | 50      | 0.8 | 0.10 | 10            | 274    | 286  | 272  |
| tai4x4            | 4        | 4    | 20000            | 50      | 0.8 | 0.01 | 10            | 273    | 286  | 272  |
| tai4x4            | 4        | 4    | 20000            | 50      | 1   | 0.10 | 10            | 272    | 272  | 272  |
| tai4x4            | 4        | 4    | 20000            | 100     | 0.8 | 0.10 | 10            | 272    | 272  | 272  |
| tai4x4            | 4        | 4    | 20000            | 100     | 0.8 | 0.01 | 10            | 272    | 272  | 272  |
| tai4x4            | 4        | 4    | 20000            | 100     | 1   | 0.10 | 10            | 272    | 272  | 272  |
| tai5x5            | 5        | 5    | 20000            | 50      | 0.8 | 0.10 | 10            | 457    | 485  | 428  |
| tai5x5            | 5        | 5    | 20000            | 50      | 0.8 | 0.01 | 10            | 441.5  | 466  | 428  |
| tai5x5            | 5        | 5    | 20000            | 50      | 1   | 0.10 | 10            | 456    | 484  | 428  |
| tai5x5            | 5        | 5    | 20000            | 100     | 0.8 | 0.10 | 10            | 442.8  | 475  | 428  |
| tai5x5            | 5        | 5    | 20000            | 100     | 0.8 | 0.01 | 10            | 433.8  | 466  | 428  |
| tai5x5            | 5        | 5    | 20000            | 100     | 1   | 0.10 | 10            | 442.8  | 475  | 428  |
| ft6x6             | 6        | 6    | 20000            | 50      | 0.8 | 0.10 | 10            | 58.9   | 64   | 55   |
| ft6x6             | 6        | 6    | 20000            | 50      | 0.8 | 0.01 | 10            | 59.3   | 64   | 58   |
| ft6x6             | 6        | 6    | 20000            | 50      | 1   | 0.10 | 10            | 58.6   | 59   | 55   |
| ft6x6             | 6        | 6    | 20000            | 100     | 0.8 | 0.10 | 10            | 59.2   | 60   | 58   |
| ft6x6             | 6        | 6    | 20000            | 100     | 0.8 | 0.01 | 10            | 58     | 60   | 55   |
| ft6x6             | 6        | 6    | 20000            | 100     | 1   | 0.10 | 10            | 57.2   | 60   | 55   |
| tai7x7            | 7        | 7    | 20000            | 50      | 0.8 | 0.10 | 10            | 641.3  | 684  | 608  |
| tai7x7            | 7        | 7    | 20000            | 50      | 0.8 | 0.01 | 10            | 647.5  | 682  | 628  |
| tai7x7            | 7        | 7    | 20000            | 50      | 1   | 0.10 | 10            | 630.6  | 659  | 590  |
| tai7x7            | 7        | 7    | 20000            | 100     | 0.8 | 0.10 | 10            | 640.1  | 664  | 623  |
| tai7x7            | 7        | 7    | 20000            | 100     | 0.8 | 0.01 | 10            | 635.5  | 655  | 603  |
| tai7x7            | 7        | 7    | 20000            | 100     | 1   | 0.10 | 10            | 620.9  | 633  | 608  |
| tai10x10          | 10       | 10   | 20000            | 50      | 0.8 | 0.10 | 10            | 1017.9 | 1107 | 944  |
| tai10x10          | 10       | 10   | 20000            | 50      | 0.8 | 0.01 | 10            | 969.2  | 1056 | 893  |
| tai10x10          | 10       | 10   | 20000            | 50      | 1   | 0.10 | 10            | 944.6  | 1028 | 877  |
| tai10x10          | 10       | 10   | 20000            | 100     | 0.8 | 0.10 | 10            | 986.8  | 1034 | 931  |
| tai10x10          | 10       | 10   | 20000            | 100     | 0.8 | 0.01 | 10            | 978.7  | 1038 | 943  |
| tai10x10          | 10       | 10   | 20000            | 100     | 1   | 0.10 | 10            | 933.5  | 1004 | 854  |
| ft10x10           | 10       | 10   | 20000            | 50      | 0.8 | 0.10 | 10            | 1187.2 | 1220 | 1120 |
| ft10x10           | 10       | 10   | 20000            | 50      | 0.8 | 0.01 | 10            | 1133.9 | 1182 | 1071 |
| ft10x10           | 10       | 10   | 20000            | 50      | 1   | 0.10 | 10            | 1151.2 | 1337 | 1029 |
| ft10x10           | 10       | 10   | 20000            | 100     | 0.8 | 0.10 | 10            | 1201   | 1250 | 1127 |
| ft10x10           | 10       | 10   | 20000            | 100     | 0.8 | 0.01 | 10            | 1200.3 | 1257 | 1108 |
| ft10x10           | 10       | 10   | 20000            | 100     | 1   | 0.10 | 10            | 1131.4 | 1197 | 1077 |
| tai15x15          | 15       | 15   | 20000            | 50      | 0.8 | 0.10 | 10            | 1748.9 | 1892 | 1642 |
| tai15x15          | 15       | 15   | 20000            | 50      | 0.8 | 0.01 | 10            | 1650.1 | 1760 | 1530 |
| tai15x15          | 15       | 15   | 20000            | 50      | 1   | 0.10 | 10            | 1640.4 | 1744 | 1579 |
| tai15x15          | 15       | 15   | 20000            | 100     | 0.8 | 0.10 | 10            | 1750.1 | 1812 | 1688 |
| tai15x15          | 15       | 15   | 20000            | 100     | 0.8 | 0.01 | 10            | 1767.5 | 1865 | 1671 |
| tai15x15          | 15       | 15   | 20000            | 100     | 1   | 0.10 | 10            | 1694.4 | 1798 | 1613 |

## Anexo 2 – Codificação do algoritmo PT1

```
avalia.solucao<-function(individuo,process,tempo,e1,Solucoes,ficheiro2)
{
  job.fim<-rep(0, nrow(process)) # Fim da última operação alocada de cada job
  mach.free<-rep(0, nrow(process)) # Fim da última operação alocada a cada máquina
  operacao.job<-rep(0, nrow(process)) # Indicar qual a próxima operação do job a executar
  nSolucoes<-get("a", env=e1)
  if (nSolucoes<Solucoes)
  {
    # Percorre o cromossoma, gene a gene, para determinar o makespan time
    for (j in 1:length(individuo))
    {
      gene <- individuo[j]
      #Atualiza a próxima operação do job a processar
      operacao.job[gene]<-operacao.job[gene]+1
      # Como uma máquina só pode processar um job de cada vez, e um job só pode processar um
      operação de cada vez
      ## Verifica o que fica livre primeiro, a máquina ou o job, e actualiza os tempos
      if (job.fim[gene]<mach.free[process[gene,operacao.job[gene]]])
      {
        mach.free[process[gene,operacao.job[gene]]]<-
        mach.free[process[gene,operacao.job[gene]]+tempo[gene,operacao.job[gene]]
        job.fim[gene]<-mach.free[process[gene,operacao.job[gene]]
      }
      else
      {
        job.fim[gene]<-job.fim[gene]+tempo[gene,operacao.job[gene]]
        mach.free[process[gene,operacao.job[gene]]<-job.fim[gene]
      }
    }
    nSolucoes<-nSolucoes+1
    assign("a", nSolucoes, env=e1)
    cat(max(mach.free),"n",file = ficheiro2) # Guarda a fitness da solução no ficheiro 2
    # Verifica qual a máquina com o tempo maior para terminar o processamento->makespan time
    return(max(mach.free))
  }
  else
  {
    nSolucoes<-nSolucoes+1
    assign("a", nSolucoes, env=e1)
    cat("9999","n",file = ficheiro2)
    return(9999)
  }
}

#-----
# Função principal Tabu Search
TS<-function(ficheiro1,ficheiro2,pred,tempo,Solucoes)
{
  # Ler problema
  process.df<- read.table(pred, header = FALSE, sep = ";")
  tempo.df<- read.table(tempo, header = FALSE, sep = ";")
  process <- matrix(unlist(process.df), nrow=nrow(process.df))
  tempo <- matrix(unlist(tempo.df), nrow=nrow(tempo.df))
  inicio.proc<-proc.time()[3]
  environment()
```

```

e1 <- new.env(TRUE, NULL)
assign("a", 0, env=e1)
nSolucoes<-0
# Avaliar solução inicial
solucao.cur<-
t(sapply(1,function(dummy){rep(1:nrow(process),ncol(process))[order(runif(nrow(process)*ncol(process)
))]})))
  valor.cur <- avalia.solucao(solucão.cur, process ,tempos,e1,Solucoes,ficheiro2)
  nSolucoes<-get("a", env=e1)
# Inicializar melhor solução
solucao.best <- solucao.cur
valor.best <- valor.cur
cat(proc.time()[3]-inicio.proc,";",valor.best,";",valor.cur,";",solucao.cur,"\n",file = ficheiro1)
# Criar a estrutura que guarda os movimentos tabu
lista.tabu <-rep(0, nrow(process)*ncol(process))
while (nSolucoes<Solucoes)
{
  # Actualiza lista tabu
  lista.tabu[which(lista.tabu>0)]<-lista.tabu[which(lista.tabu>0)]-1
  # Construir e avaliar todos os vizinhos
  fitness.vizinhos <- sapply(1:(length(solucão.cur)-1), function(i, sc, p, t,e1,Solucoes,ficheiro2)
  {
    v <- replace(sc, i:(i+1), sc[(i+1):i])
    avalia.solucao(v, p, t,e1,Solucoes,ficheiro2)
  }, solucao.cur, process, tempos,e1,Solucoes,ficheiro2)

  tabus <- sapply(1:(length(solucão.cur)-1), function(i, lt, sc) {lt[i]}, lista.tabu, solucao.cur)

  # Próxima solução corrente é o vizinho que seja melhor que o best ou o melhor vizinho (não-tabu) ou,
  se não houver nenhum, o que tiver menor tabu ou que for melhor que best
  i.melhor.vizinho <-
  ifelse(sum(which(fitness.vizinhos<valor.best))>0,which(fitness.vizinhos<valor.best),ifelse(all(lista.tabu >
0),
  (1:(length(solucão.cur)-1))[tabus <= 0][which.min(fitness.vizinhos[tabus <= 0])],
  (1:(length(solucão.cur)-1))[which.min(tabus)]))
  #Actualiza a solução corrente
  solucao.cur <- replace(solucão.cur, i.melhor.vizinho:(i.melhor.vizinho+1),
solucao.cur[(i.melhor.vizinho+1):i.melhor.vizinho])
  valor.cur <- fitness.vizinhos[i.melhor.vizinho]
  lista.tabu[i.melhor.vizinho] <- sqrt(length(solucão.cur))
  # Actualiza melhor solução
  if (valor.cur < valor.best)
  {
    valor.best <- valor.cur
    solucao.best <- solucao.cur
  }
  # Guarda dados da iteração no ficheiro de configurações
  cat(proc.time()[3]-inicio.proc,";",valor.best,";",valor.cur,";",solucao.cur,"\n",file = ficheiro1)
  nSolucoes<-get("a", env=e1)
}
return(valor.best)
}

obtem.resultados<-function(problemas,parametros,repeticoes)
{
  fim.problemas<-FALSE
  hist.problemas<-as.matrix(read.table(problemas, header = FALSE, sep = ";"))
  while (fim.problemas==FALSE)

```

```

{
  fim.parametros<-FALSE
  hist.parametros<-as.matrix(read.table(parametros, header = FALSE, sep = ";"))
  while (fim.parametros==FALSE)
  {
    fim.repeticoes<-FALSE
    hist.repeticoes<-as.matrix(read.table(repeticoes, header = FALSE, sep = ";"))
    while (fim.repeticoes==FALSE)
    {
      ficheiro1<-
file(paste("res1.",hist.problemas[which.min(hist.problemas[,2]),1],".param",which.min(hist.parametros[,4
]),".rep",which.min(hist.repeticoes[,2]),".txt",sep=""),"w")
      ficheiro2<-
file(paste("res2.",hist.problemas[which.min(hist.problemas[,2]),1],".param",which.min(hist.parametros[,4
]),".rep",which.min(hist.repeticoes[,2]),".txt",sep=""),"w")
      TS(ficheiro1,ficheiro2,paste("pred",hist.problemas[which.min(hist.problemas[,2]),1],".txt",sep=""),paste(
"tempos",hist.problemas[which.min(hist.problemas[,2]),1],".txt",sep=""),hist.parametros[which.min(hist.
parametros[,4]),3])
      close(ficheiro1)
      close(ficheiro2)
      if (which.min(hist.repeticoes[,2])==nrow(hist.repeticoes))
      {
        hist.repeticoes[,2]<-0
        fim.repeticoes<-TRUE
      }
      else hist.repeticoes[which.min(hist.repeticoes[,2]),2]<-1
      write.table(hist.repeticoes, file = "hist.repeticoes.txt", sep = ";",row.names = FALSE, col.names =
FALSE)
    }
    if (which.min(hist.parametros[,4])==nrow(hist.parametros))
    {
      hist.parametros[,4]<-0
      fim.parametros<-TRUE
    }
    else hist.parametros[which.min(hist.parametros[,4]),4]<-1
    write.table(hist.parametros, file = "hist.parametros.txt", sep = ";",row.names = FALSE, col.names =
FALSE)
  }
  if (which.min(hist.problemas[,2])==nrow(hist.problemas)) fim.problemas<-TRUE
  else hist.problemas[which.min(hist.problemas[,2]),2]<-1
  write.table(hist.problemas, file = "hist.problemas.txt", sep = ";",row.names = FALSE, col.names =
FALSE)
}
}

obtem.resultados("hist.problemas.txt","hist.parametros.txt","hist.repeticoes.txt")

```

### **Anexo 3 – Codificação do algoritmo PT2**

```
#-----
# Função que avalia cada elemento que constitui a população (valor de fitness)
avalia.populacao <-function(ficheiro2,nSolucoes,Solucoes,pop,inicio,process, tempos)
{
  apply(pop[inicio:nrow(pop),], 1, avalia.solucao, process, tempos,nSolucoes,Solucoes,ficheiro2)
}

#-----
avalia.solucao<-function(individuo,process,tempos,e1,Solucoes,ficheiro2)
{
  job.fim<-rep(0, nrow(process)) # Fim da última operação alocada de cada job
  mach.free<-rep(0, nrow(process)) # Fim da última operação alocada a cada máquina
  operacao.job<-rep(0, nrow(process)) # Indicar qual a próxima operação do job a executar
  nSolucoes<-get("a", env=e1)
  if (nSolucoes<Solucoes)
  {
    # Percorre o cromossoma, gene a gene, para determinar o makespan time
    for (j in 1:length(individuo))
    {
      gene <- individuo[j]
      #Actualiza a próxima operação do job a processar
      operacao.job[gene]<-operacao.job[gene]+1
      # Como uma máquina só pode processar um job de cada vez, e um job só pode processar um
      operação de cada vez
      ## Verifica o que fica livre primeiro, a máquina ou o job, e actualiza os tempos
      if (job.fim[gene]<mach.free[process[gene,operacao.job[gene]]])
      {
        mach.free[process[gene,operacao.job[gene]]]<-
        mach.free[process[gene,operacao.job[gene]]+tempos[gene,operacao.job[gene]]
        job.fim[gene]<-mach.free[process[gene,operacao.job[gene]]]
      }
      else
      {
        job.fim[gene]<-job.fim[gene]+tempos[gene,operacao.job[gene]]
        mach.free[process[gene,operacao.job[gene]]]<-job.fim[gene]
      }
    }
    nSolucoes<-nSolucoes+1
    assign("a", nSolucoes, env=e1)
    cat(max(mach.free),"\\n",file = ficheiro2) # Guarda a fitness da solução no ficheiro 2
    # Verifica qual a máquina com o tempo maior para terminar o processamento->makespan time
    return(max(mach.free))
  }
  else
  {
    nSolucoes<-nSolucoes+1
    assign("a", nSolucoes, env=e1)
    cat("9999","\\n",file = ficheiro2)
    return(9999)
  }
}

#-----
```

```

# Função principal Tabu Search
TS<-function(ficheiro1,ficheiro2,pred,tempos,Solucoes)
{
  # Ler problema
  process.df<- read.table(pred, header = FALSE, sep = ";")
  tempos.df<- read.table(tempos, header = FALSE, sep = ";")
  process <- matrix(unlist(process.df), nrow=nrow(process.df))
  tempos <- matrix(unlist(tempos.df), nrow=nrow(tempos.df))
  environment()
  e1 <- new.env(TRUE, NULL)
  assign("a", 0, env=e1)

  # Inicializar população
  solucoes.ini <- rep(1:nrow(process),ncol(process))[order(runif(nrow(process)*ncol(process)))]
  fitness.ini <- avalia.populacao(ficheiro2,e1,Solucoes,solucoes.ini,1,process, tempos)
  inicio.proc<-proc.time()[3]
  nSolucoes<-500
  # Avaliar solução inicial
  solucao.cur<-solucoes.ini[which.min(fitness.ini),]
  valor.cur <- min(fitness.ini)
  # Inicializar melhor solução
  solucao.best <- solucao.cur
  valor.best <- valor.cur
  cat(proc.time()[3]-inicio.proc,";",valor.best,";",valor.cur,";",solucao.cur,"\n",file = ficheiro1)
  # Criar a estrutura que guarda os movimentos tabu
  lista.tabu <-rep(0, nrow(process)*ncol(process))
  while (nSolucoes<Solucoes)
  {
    # Actualiza lista tabu
    lista.tabu[which(lista.tabu>0)]<-lista.tabu[which(lista.tabu>0)]-1
    # Construir e avaliar todos os vizinhos
    fitness.vizinhos <- sapply(1:(length(solucao.cur)-1), function(i, sc, p, t,e1,Solucoes,ficheiro2)
    {
      v <- replace(sc, i:(i+1), sc[(i+1):i])
      avalia.solucao(v, p, t,e1,Solucoes,ficheiro2)
    }, solucao.cur, process, tempos,e1,Solucoes,ficheiro2)

    tabus <- sapply(1:(length(solucao.cur)-1), function(i, lt, sc) {lt[i]}, lista.tabu, solucao.cur)

    # Próxima solução corrente é o vizinho que seja melhor que o best ou o melhor vizinho (não-tabu) ou,
    # se não houver nenhum, o que tiver menor tabu ou que for melhor que best
    i.melhor.vizinho <- ifelse(sum(which(fitness.vizinhos<valor.best))>0,which(fitness.vizinhos<valor.best),ifelse(all(lista.tabu > 0),
    (1:(length(solucao.cur)-1))[tabus <= 0][which.min(fitness.vizinhos[tabus <= 0 ])],
    (1:(length(solucao.cur)-1))[which.min(tabus)]))

    #Actualiza a solução corrente
    solucao.cur <- replace(solucao.cur, i.melhor.vizinho:(i.melhor.vizinho+1),
    solucao.cur[(i.melhor.vizinho+1):i.melhor.vizinho])
    valor.cur <- fitness.vizinhos[i.melhor.vizinho]
    lista.tabu[i.melhor.vizinho] <- sqrt(length(solucao.cur))
    # Actualiza melhor solução
    if (valor.cur < valor.best)
    {
      valor.best <- valor.cur
      solucao.best <- solucao.cur
    }
  }
}

```

```

    }
    # Guarda dados da iteração no ficheiro de configurações
    cat(proc.time()[3]-inicio.proc, ";", valor.best, ";", valor.cur, ";", solucao.cur, "\n", file = ficheiro1)
    nSolucoes<-get("a", env=e1)
  }

  return(valor.best)
}

obtem.resultados<-function(problemas,parametros,repeticoes)
{
  fim.problemas<-FALSE
  hist.problemas<-as.matrix(read.table(problemas, header = FALSE, sep = ";"))
  while (fim.problemas==FALSE)
  {
    fim.parametros<-FALSE
    hist.parametros<-as.matrix(read.table(parametros, header = FALSE, sep = ";"))
    while (fim.parametros==FALSE)
    {
      fim.repeticoes<-FALSE
      hist.repeticoes<-as.matrix(read.table(repeticoes, header = FALSE, sep = ";"))
      while (fim.repeticoes==FALSE)
      {
        ficheiro1<-
file(paste("res1.",hist.problemas[which.min(hist.problemas[,2]),1], ".param",which.min(hist.parametros[,4
]),".rep",which.min(hist.repeticoes[,2]),".txt",sep=""),"w")
        ficheiro2<-
file(paste("res2.",hist.problemas[which.min(hist.problemas[,2]),1], ".param",which.min(hist.parametros[,4
]),".rep",which.min(hist.repeticoes[,2]),".txt",sep=""),"w")

TS(ficheiro1,ficheiro2,paste("pred",hist.problemas[which.min(hist.problemas[,2]),1],".txt",sep=""),paste(
"tempos",hist.problemas[which.min(hist.problemas[,2]),1],".txt",sep=""),hist.parametros[which.min(hist.
parametros[,4]),3])
        close(ficheiro1)
        close(ficheiro2)
        if (which.min(hist.repeticoes[,2])==nrow(hist.repeticoes))
        {
          hist.repeticoes[,2]<-0
          fim.repeticoes<-TRUE
        }
        else hist.repeticoes[which.min(hist.repeticoes[,2]),2]<-1
        write.table(hist.repeticoes, file = "hist.repeticoes.txt", sep = ";",row.names = FALSE, col.names =
FALSE)
      }
      if (which.min(hist.parametros[,4])==nrow(hist.parametros))
      {
        hist.parametros[,4]<-0
        fim.parametros<-TRUE
      }
      else hist.parametros[which.min(hist.parametros[,4]),4]<-1
      write.table(hist.parametros, file = "hist.parametros.txt", sep = ";",row.names = FALSE, col.names =
FALSE)
    }
    if (which.min(hist.problemas[,2])==nrow(hist.problemas)) fim.problemas<-TRUE
    else hist.problemas[which.min(hist.problemas[,2]),2]<-1
    write.table(hist.problemas, file = "hist.problemas.txt", sep = ";",row.names = FALSE, col.names =
FALSE)
  }
}

```

```
}
```

```
obtem.resultados("hist.problemas.txt","hist.parametros.txt","hist.repeticoes.txt")
```

## ***Anexo 4 – Codificação do algoritmo AG***

```
#-----
# Função para gerar a população
gera.populacao<-function(popSize,jobs,machines)
{
  t(sapply(1:popSize,function(dummy){rep(1:jobs,machines)[order(runif(jobs*machines))]})))
}

#-----
# Função que faz mutação a cada elemento que constitui a população
mutacao.populacao <-function(pop, Pm)
{
  t(apply(pop[(round(nrow(pop)*0.1) + 1):nrow(pop),], 1, mutacao.individuo, Pm))
}

#-----
# Função de mutação de individuos
mutacao.individuo<-function(individuo,Pm)
{
  # Faz mutação com probabilidade Pm
  if (runif(1)<Pm)
  {
    # Shift mutation
    # Selecciona um gene aleatoriamente
    genes <- sample(1:length(individuo), 2)
    guardaGene<-individuo[genes[1]]
    # Vamos mover o gene seleccionado para a posição seleccionada
    if (genes[1]<genes[2])
    {
      individuo <- replace(individuo, genes[1]:(genes[2]-1), individuo[(genes[1]+1):genes[2]])
    }
    else
    {
      individuo <- replace(individuo, (genes[2]+1):genes[1], individuo[genes[2]:(genes[1]-1)])
    }
    individuo[genes[2]]<-guardaGene
  }

  return(individuo)
}

#-----
# Função de cruzamento de individuos
cruzamento.individuos<-function(pop,npop,fitness)
{
  insere<-round(nrow(pop)*0.1) + 1
  while (insere<=nrow(pop))
  {
    # Operador de selecção: Binary Tournament
    # selecciona 2 individuos de forma aleatória, sendo o melhor deles o 1º pai
    presel <- sample(1:nrow(pop), 4)
    pai1 <- presel[which.max(fitness[presel[1:2]])]
    pai2 <- presel[which.max(fitness[presel[3:4]])+2]

    # Operador de Cruzamento: Two-Point Crossover
    # Definem-se dois pontos de corte
```

```

points <- sort(sample(1:ncol(pop), 2))
# Gerar primeiro filho
npop[inserere, points[1]:points[2]] <- pop[pai1, points[1]:points[2]]
npop[inserere,-(points[1]:points[2])] <- rev(rev(pop[pai2,])[-
(sort(pmatch(as.character(pop[pai1,points[1]:points[2]]),as.character(rev(pop[pai2,])),dup=FALSE))))])
if ((inserere+1)<=nrow(pop))      # se ainda há espaço na população
{
  # Gerar segundo filho
  npop[inserere+1, points[1]:points[2]] <- pop[pai2, points[1]:points[2]]
  npop[inserere+1,-(points[1]:points[2])] <- rev(rev(pop[pai1,])[-
(sort(pmatch(as.character(pop[pai2,points[1]:points[2]]),as.character(rev(pop[pai1,])),dup=FALSE))))])
}
inserere<-inserere+2
}
return(npop)
}

#-----
# Função que avalia cada elemento que constitui a população (valor de fitness)
avalia.populacao <-function(ficheiro2,nSolucoes,Solucoes,pop,inicio,process, tempos)
{
  apply(pop[inicio:nrow(pop),], 1, avalia.solucao, process, tempos,nSolucoes,Solucoes,ficheiro2)
}

#-----
avalia.solucao<-function(individuo,process,tempos,e1,Solucoes,ficheiro2)
{
  job.fim<-rep(0, nrow(process)) # Fim da última operação alocada de cada job
  mach.free<-rep(0, nrow(process)) # Fim da última operação alocada a cada máquina
  operacao.job<-rep(0, nrow(process)) # Indicar qual a próxima operação do job a executar
  nSolucoes<-get("a", env=e1)
  if (nSolucoes<Solucoes)
  {
    # Percorre o cromossoma, gene a gene, para determinar o makespan time
    for (j in 1:length(individuo))
    {
      gene <- individuo[j]
      #Actualiza a próxima operação do job a processar
      operacao.job[gene]<-operacao.job[gene]+1
      # Como uma máquina só pode processar um job de cada vez, e um job só pode processar um
      operação de cada vez
      ## Verifica o que fica livre primeiro, a máquina ou o job, e actualiza os tempos
      if (job.fim[gene]<mach.free[process[gene,operacao.job[gene]]])
      {
        mach.free[process[gene,operacao.job[gene]]]<-
        mach.free[process[gene,operacao.job[gene]]]+tempos[gene,operacao.job[gene]]
        job.fim[gene]<-mach.free[process[gene,operacao.job[gene]]]
      }
      else
      {
        job.fim[gene]<-job.fim[gene]+tempos[gene,operacao.job[gene]]
        mach.free[process[gene,operacao.job[gene]]]<-job.fim[gene]
      }
    }
  }
  nSolucoes<-nSolucoes+1
  assign("a", nSolucoes, env=e1)
  cat(max(mach.free),"n",file = ficheiro2)      # Guarda a fitness da solução no ficheiro 2
  # Verifica qual a máquina com o tempo maior para terminar o processamento->makespan time

```

```

    return(max(mach.free))
  }
else
{
  nSolucoes<-nSolucoes+1
  assign("a", nSolucoes, env=e1)
  cat("9999","\n",file = ficheiro2)
  return(9999)
}
}

#-----
# Função principal do algoritmo genético
AG<-function(ficheiro1,ficheiro2,pred,tempos,Solucoes,popSize, Pm)
{
  # Ler problema
  process.df<- read.table(pred, header = FALSE, sep = ";")
  tempos.df<- read.table(tempos, header = FALSE, sep = ";")
  process <- matrix(unlist(process.df), nrow=nrow(process.df))
  tempos <- matrix(unlist(tempos.df), nrow=nrow(tempos.df))
  inicio.proc<-proc.time()[3]
  environment()
  e1 <- new.env(TRUE, NULL)
  assign("a", 0, env=e1)
  # Inicializar população
  pop <- gera.populacao(popSize,nrow(process),ncol(process))
  fitness <- avalia.populacao(ficheiro2,e1,Solucoes,pop,1,process, tempos)
  nSolucoes<-get("a", env=e1)
  while (nSolucoes<Solucoes)
  {
    #Verifica a ordem dos elementos que compõe o vector de valores de fitness dos filhos gerados
    # e aplica a ordem à matriz de individuos e de fitness
    ordem<-order(fitness)
    pop<-pop[ordem,]
    fitness<-fitness[ordem]
    # Cria as estruturas de dados para a nova população
    npop <- matrix(0, nrow=nrow(pop), ncol=ncol(pop))
    # Faz o survival selection: selecciona os 10% melhores individuos da geração anterior
    npop[1:round(nrow(pop)*0.1),]<-pop[1:round(nrow(pop)*0.1),]
    npop <- cruzamento.individuos(pop,npop, fitness) # Selecção de pais e Crossover
    npop[(round(nrow(pop)*0.1) + 1):nrow(pop),]<-mutacao.populacao(npop,Pm)
    i<-1 # Guarda a fitness das soluções que vêm da geração anterior no ficheiro 2
    while (i<=round(nrow(pop)*0.1))
    {
      cat(fitness[i],"\\n",file = ficheiro2)
      i<-i+1
    }
    nSolucoes<-nSolucoes+i-1
    assign("a", nSolucoes, env=e1)
    fitness[(round(nrow(pop)*0.1) + 1):nrow(pop)]<-
    avalia.populacao(ficheiro2,e1,Solucoes,npop,(round(nrow(pop)*0.1) + 1),process, tempos) #
    Avaliar nova população
    pop<-npop
    nSolucoes<-get("a", env=e1)
    # Guarda dados da iteração no ficheiro de configurações
    cat(proc.time()[3]-inicio.proc,";",min(fitness),";",pop[which.min(fitness),],"\\n",file = ficheiro1)
  }
}

```

```

    return(min(fitness))
}

obtem.resultados<-function(problemas,parametros,repeticoes)
{
  fim.problemas<-FALSE
  hist.problemas<-as.matrix(read.table(problemas, header = FALSE, sep = ";"))
  while (fim.problemas==FALSE)
  {
    fim.parametros<-FALSE
    hist.parametros<-as.matrix(read.table(parametros, header = FALSE, sep = ";"))
    while (fim.parametros==FALSE)
    {
      fim.repeticoes<-FALSE
      hist.repeticoes<-as.matrix(read.table(repeticoes, header = FALSE, sep = ";"))
      while (fim.repeticoes==FALSE)
      {
        arquivo1<-
file(paste("res1.",hist.problemas[which.min(hist.problemas[,2]),1],".param",which.min(hist.parametros[,4
]),".rep",which.min(hist.repeticoes[,2]),".txt",sep=""),"w")
        arquivo2<-
file(paste("res2.",hist.problemas[which.min(hist.problemas[,2]),1],".param",which.min(hist.parametros[,4
]),".rep",which.min(hist.repeticoes[,2]),".txt",sep=""),"w")

AG(arquivo1,arquivo2,paste("pred",hist.problemas[which.min(hist.problemas[,2]),1],".txt",sep=""),paste(
"tempos",hist.problemas[which.min(hist.problemas[,2]),1],".txt",sep=""),hist.parametros[which.min(hist.
parametros[,4]),3],hist.parametros[which.min(hist.parametros[,4]),2],hist.parametros[which.min(hist.para
metros[,4]),1])
        close(arquivo1)
        close(arquivo2)
        if (which.min(hist.repeticoes[,2])==nrow(hist.repeticoes))
        {
          hist.repeticoes[,2]<-0
          fim.repeticoes<-TRUE
        }
        else hist.repeticoes[which.min(hist.repeticoes[,2]),2]<-1
        write.table(hist.repeticoes, file = "hist.repeticoes.txt", sep = ";",row.names = FALSE, col.names =
FALSE)
      }
      if (which.min(hist.parametros[,4])==nrow(hist.parametros))
      {
        hist.parametros[,4]<-0
        fim.parametros<-TRUE
      }
      else hist.parametros[which.min(hist.parametros[,4]),4]<-1
      write.table(hist.parametros, file = "hist.parametros.txt", sep = ";",row.names = FALSE, col.names =
FALSE)
    }
    if (which.min(hist.problemas[,2])==nrow(hist.problemas)) fim.problemas<-TRUE
    else hist.problemas[which.min(hist.problemas[,2]),2]<-1
    write.table(hist.problemas, file = "hist.problemas.txt", sep = ";",row.names = FALSE, col.names =
FALSE)
  }
}

obtem.resultados("hist.problemas.txt","hist.parametros.txt","hist.repeticoes.txt")

```

## ***Anexo 5 – Codificação do algoritmo AGH1***

```
#-----
# Função para gerar a população
gera.populacao<-function(popSize,jobs,machines)
{
  t(sapply(1:popSize,function(dummy){rep(1:jobs,machines)[order(runif(jobs*machines))]}))
}

#-----
# Função que faz mutação a cada elemento que constitui a população
mutacao.populacao <-function(pop, Pm)
{
  t(apply(pop[(round(nrow(pop)*0.1) + 1):nrow(pop),], 1, mutacao.individuo, Pm))
}

#-----
# Função de mutação de individuos
mutacao.individuo<-function(individuo,Pm)
{
  # Faz mutação com probabilidade Pm
  if (runif(1)<Pm)
  {
    # Shift mutation
    # Selecciona um gene aleatoriamente
    genes <- sample(1:length(individuo), 2)
    guardaGene<-individuo[genes[1]]
    # Vamos mover o gene seleccionado para a posição seleccionada
    if (genes[1]<genes[2])
    {
      individuo <- replace(individuo, genes[1]:(genes[2]-1), individuo[(genes[1]+1):genes[2]])
    }
    else
    {
      individuo <- replace(individuo, (genes[2]+1):genes[1], individuo[genes[2]:(genes[1]-1)])
    }
    individuo[genes[2]]<-guardaGene
  }

  return(individuo)
}

#-----
# Função de cruzamento de individuos
cruzamento.individuos<-function(pop,npop,fitness)
{
  insere<-round(nrow(pop)*0.1) + 1
  while (insere<=nrow(pop))
  {
    # Operador de selecção: Binary Tournament
    # selecciona 2 individuos de forma aleatória, sendo o melhor deles o 1º pai
    presel <- sample(1:nrow(pop), 4)
    pai1 <- presel[which.max(fitness[presel[1:2]])]
    pai2 <- presel[which.max(fitness[presel[3:4]])+2]
    # Operador de Cruzamento: Two-Point Crossover
    # Definem-se dois pontos de corte
    points <- sort(sample(1:ncol(pop), 2))
```

```

# Gerar primeiro filho
npop[inserere, points[1]:points[2]] <- pop[pai1, points[1]:points[2]]
npop[inserere,-(points[1]:points[2])<-rev(rev(pop[pai2,])[-
(sort(pmatch(as.character(pop[pai1,points[1]:points[2]]),as.character(rev(pop[pai2,])),dup=FALSE))))))
if ((inserere+1)<=nrow(pop)) # se ainda há espaço na população
{
  # Gerar segundo filho
  npop[inserere+1, points[1]:points[2]] <- pop[pai2, points[1]:points[2]]
  npop[inserere+1,-(points[1]:points[2])<-rev(rev(pop[pai1,])[-
(sort(pmatch(as.character(pop[pai2,points[1]:points[2]]),as.character(rev(pop[pai1,])),dup=FALSE))))))
}
inserere<-inserere+2
}
return(npop)
}

#-----
# Função que avalia cada elemento que constitui a população (valor de fitness)
avalia.populacao <-function(ficheiro2,nSolucoes,Solucoes,pop,inicio,process, tempos)
{
  apply(pop[inicio:nrow(pop),], 1, avalia.solucao, process, tempos,nSolucoes,Solucoes,ficheiro2)
}

#-----
avalia.solucao<-function(individuo,process,tempos,e1,Solucoes,ficheiro2)
{
  job.fim<-rep(0, nrow(process)) # Fim da última operação alocada de cada job
  mach.free<-rep(0, nrow(process)) # Fim da última operação alocada a cada máquina
  operacao.job<-rep(0, nrow(process)) # Indicar qual a próxima operação do job a executar
  nSolucoes<-get("a", env=e1)
  if (nSolucoes<Solucoes)
  {
    # Percorre o cromossoma, gene a gene, para determinar o makespan time
    for (j in 1:length(individuo))
    {
      gene <- individuo[j]
      #Atualiza a próxima operação do job a processar
      operacao.job[gene]<-operacao.job[gene]+1
      # Como uma máquina só pode processar um job de cada vez, e um job só pode processar um
      operação de cada vez
      ## Verifica o que fica livre primeiro, a máquina ou o job, e actualiza os tempos
      if (job.fim[gene]<mach.free[process[gene,operacao.job[gene]]])
      {
        mach.free[process[gene,operacao.job[gene]]]<-
        mach.free[process[gene,operacao.job[gene]]+tempos[gene,operacao.job[gene]]
        job.fim[gene]<-mach.free[process[gene,operacao.job[gene]]
      }
      else
      {
        job.fim[gene]<-job.fim[gene]+tempos[gene,operacao.job[gene]]
        mach.free[process[gene,operacao.job[gene]]<-job.fim[gene]
      }
    }
  }
  nSolucoes<-nSolucoes+1
  assign("a", nSolucoes, env=e1)
  cat(max(mach.free),"\\n",file = ficheiro2) # Guarda a fitness da solução no ficheiro 2
  # Verifica qual a máquina com o tempo maior para terminar o processamento->makespan time
  return(max(mach.free))
}

```

```

    }
  else
  {
    nSolucoes<-nSolucoes+1
    assign("a", nSolucoes, env=e1)
    cat("9999","\n",file = ficheiro2)
    return(9999)
  }
}

#-----
avalia.solucaoTS<-function(individuo,process,tempos)
{
  job.fim<-rep(0, nrow(process)) # Fim da última operação alocada de cada job
  mach.free<-rep(0, nrow(process)) # Fim da última operação alocada a cada máquina
  operacao.job<-rep(0, nrow(process)) # Indicar qual a próxima operação do job a executar
  # Percorre o cromossoma, gene a gene, para determinar o makespan time
  for (j in 1:length(individuo))
  {
    gene <- individuo[j]
    #Actualiza a próxima operação do job a processar
    operacao.job[gene]<-operacao.job[gene]+1
    # Como uma máquina só pode processar um job de cada vez, e um job só pode processar um operação
    de cada vez
    ## Verifica o que fica livre primeiro, a máquina ou o job, e actualiza os tempos
    if (job.fim[gene]<mach.free[process[gene,operacao.job[gene]]])
    {
      mach.free[process[gene,operacao.job[gene]]]<-
      mach.free[process[gene,operacao.job[gene]]]+tempos[gene,operacao.job[gene]]
      job.fim[gene]<-mach.free[process[gene,operacao.job[gene]]]
    }
    else
    {
      job.fim[gene]<-job.fim[gene]+tempos[gene,operacao.job[gene]]
      mach.free[process[gene,operacao.job[gene]]]<-job.fim[gene]
    }
  }
  # Verifica qual a máquina com o tempo maior para terminar o processamento->makespan time
  return(max(mach.free))
}

#-----
# Função principal Tabu Search
TS<-function(solucaoCur,fitValue,e1,numSoluc,ficheiro2,process,tempos,Solucoes)
{
  nSolucoes<-numSoluc
  i<-nSolucoes
  # Avaliar solução inicial
  solucao.cur<-solucaoCur
  valor.cur <- fitValue
  # Inicializar melhor solução
  solucao.best <- solucao.cur
  valor.best <- valor.cur
  # Criar a estrutura que guarda os movimentos tabu
  lista.tabu <-rep(0, nrow(process)*ncol(process))
  while (nSolucoes<Solucoes && nSolucoes<(i+(Solucoes/1000)))
  {
    # Actualiza lista tabu

```

```

lista.tabu[which(lista.tabu>0)]<-lista.tabu[which(lista.tabu>0)]-1

# Construir e avaliar todos os vizinhos
fitness.vizinhos <- sapply(1:(length(solucao.cur)-1), function(i, sc, p, t,e1,Solucoes,ficheiro2)
{
  v <- replace(sc, i:(i+1), sc[(i+1):i])
  avalia.solucao(v, p, t,e1,Solucoes,ficheiro2)
}, solucao.cur, process, tempos,e1,Solucoes,ficheiro2)
tabus <- sapply(1:(length(solucao.cur)-1), function(i, lt, sc) {lt[i]}, lista.tabu, solucao.cur)
# Próxima solução corrente é o vizinho que seja melhor que o best ou o melhor vizinho (não-tabu) ou,
se não houver nenhum, o que tiver menor tabu ou que for melhor que best
i.melhor.vizinho <-
ifelse(sum(which(fitness.vizinhos<valor.best))>0,which(fitness.vizinhos<valor.best),ifelse(all(lista.tabu >
0),
(1:(length(solucao.cur)-1))[tabus <= 0][which.min(fitness.vizinhos[tabus <= 0 ])],
(1:(length(solucao.cur)-1))[which.min(tabus)]))
#Atualiza a solução corrente
solucao.cur <- replace(solucao.cur, i.melhor.vizinho:(i.melhor.vizinho+1),
solucao.cur[(i.melhor.vizinho+1):i.melhor.vizinho])
valor.cur <- fitness.vizinhos[i.melhor.vizinho]
lista.tabu[i.melhor.vizinho] <- sqrt(length(solucao.cur))
# Atualiza melhor solução
if (valor.cur < valor.best)
{
  valor.best <- valor.cur
  solucao.best <- solucao.cur
}

nSolucoes<-get("a", env=e1)
}
return(solucao.best)
}

#-----
# Função principal do algoritmo genético
AG<-function(ficheiro1,ficheiro2,pred,tempos,Solucoes,popSize, Pm)
{
  # Ler problema
  process.df<- read.table(pred, header = FALSE, sep = ";")
  tempos.df<- read.table(tempos, header = FALSE, sep = ";")
  process <- matrix(unlist(process.df), nrow=nrow(process.df))
  tempos <- matrix(unlist(tempos.df), nrow=nrow(tempos.df))
  inicio.proc<-proc.time()[3]
  environment()
  e1 <- new.env(TRUE, NULL)
  assign("a", 0, env=e1)
  # Inicializar população
  pop <- gera.populacao(popSize,nrow(process),ncol(process))
  fitness <- avalia.populacao(ficheiro2,e1,Solucoes,pop,1,process, tempos)
  nSolucoes<-get("a", env=e1)
  while (nSolucoes<Solucoes)
  {
    #Verifica a ordem dos elementos que compõe o vector de valores de fitness dos filhos gerados
    # e aplica a ordem à matriz de individuos e de fitness
    ordem<-order(fitness)
    pop<-pop[ordem,]
    fitness<-fitness[ordem]
  }
}

```

```

# Cria as estruturas de dados para a nova população
npop <- matrix(0, nrow=nrow(pop), ncol=ncol(pop))
# Faz o survival selection: selecciona os 10% melhores individuos da geração anterior
npop[1:round(nrow(pop)*0.1),]<-pop[1:round(nrow(pop)*0.1),]
npop <- cruzamento.individuos(pop,npop, fitness) # Selecção de pais e Crossover
npop[(round(nrow(pop)*0.1) + 1):nrow(pop),]<-mutacao.populacao(npop,Pm)
i<-1 # Guarda a fitness das soluções que vêm da geração anterior no ficheiro 2
while (i<=round(nrow(pop)*0.1))
{
  cat(fitness[i],"\n",file = ficheiro2)
  i<-i+1
}
nSolucoes<-nSolucoes+i-1
assign("a", nSolucoes, env=e1)

fitness[(round(nrow(pop)*0.1) + 1):nrow(pop)]<-
avalia.populacao(ficheiro2,e1,Solucoes,npop,(round(nrow(pop)*0.1) + 1),process, tempos) #
Avaliar nova população
melhor<-which.min(fitness[(round(nrow(pop)*0.1) + 1):nrow(pop)])+(round(nrow(pop)*0.1))
npop[melhor,]<-TS(npop[melhor,],fitness[melhor,],e1,nSolucoes,ficheiro2,process,tempos,Solucoes)
#cat("SOLUCAO TS: ",npop[which.min(fitness[(round(nrow(pop)*0.1) + 1):nrow(pop)])+(round(nrow(pop)*0.1)),,"\n")
pop<-npop
fitness[melhor]<-avalia.solucaoTS(pop[melhor,],process,tempos)
nSolucoes<-get("a", env=e1)
# Guarda dados da iteração no ficheiro de configurações
cat(proc.time()[3]-inicio.proc,";",min(fitness),";",pop[which.min(fitness),],"\n",file = ficheiro1)
}
return(min(fitness))
}

obtem.resultados<-function(problemas,parametros,repeticoes)
{
  fim.problemas<-FALSE
  hist.problemas<-as.matrix(read.table(problemas, header = FALSE, sep = ";"))
  while (fim.problemas==FALSE)
  {
    fim.parametros<-FALSE
    hist.parametros<-as.matrix(read.table(parametros, header = FALSE, sep = ";"))
    while (fim.parametros==FALSE)
    {
      fim.repeticoes<-FALSE
      hist.repeticoes<-as.matrix(read.table(repeticoes, header = FALSE, sep = ";"))
      while (fim.repeticoes==FALSE)
      {
        ficheiro1<-
file(paste("res1.",hist.problemas[which.min(hist.problemas[,2]),1],".param",which.min(hist.parametros[,4]),".rep",which.min(hist.repeticoes[,2]),".txt",sep=""),"w")
        ficheiro2<-
file(paste("res2.",hist.problemas[which.min(hist.problemas[,2]),1],".param",which.min(hist.parametros[,4]),".rep",which.min(hist.repeticoes[,2]),".txt",sep=""),"w")
        AG(ficheiro1,ficheiro2,paste("pred",hist.problemas[which.min(hist.problemas[,2]),1],".txt",sep=""),paste("tempos",hist.problemas[which.min(hist.problemas[,2]),1],".txt",sep=""),hist.parametros[which.min(hist.parametros[,4]),3],hist.parametros[which.min(hist.parametros[,4]),2],hist.parametros[which.min(hist.parametros[,4]),1])
        close(ficheiro1)
        close(ficheiro2)
        if (which.min(hist.repeticoes[,2])==nrow(hist.repeticoes))

```

```

    {
      hist.repeticoes[,2]<-0
      fim.repeticoes<-TRUE
    }
    else hist.repeticoes[which.min(hist.repeticoes[,2]),2]<-1
    write.table(hist.repeticoes, file = "hist.repeticoes.txt", sep = ";",row.names = FALSE, col.names =
FALSE)
  }
  if (which.min(hist.parametros[,4])==nrow(hist.parametros))
  {
    hist.parametros[,4]<-0
    fim.parametros<-TRUE
  }
  else hist.parametros[which.min(hist.parametros[,4]),4]<-1
  write.table(hist.parametros, file = "hist.parametros.txt", sep = ";",row.names = FALSE, col.names =
FALSE)
}
if (which.min(hist.problemas[,2])==nrow(hist.problemas)) fim.problemas<-TRUE
else hist.problemas[which.min(hist.problemas[,2]),2]<-1
write.table(hist.problemas, file = "hist.problemas.txt", sep = ";",row.names = FALSE, col.names =
FALSE)
}
}

obtem.resultados("hist.problemas.txt","hist.parametros.txt","hist.repeticoes.txt")

```

## ***Anexo 6 – Codificação do algoritmo AGH2***

```
#-----  
# Função para gerar a população  
gera.populacao<-function(popSize,jobs,machines)  
{  
  t(sapply(1:popSize,function(dummy){rep(1:jobs,machines)[order(runif(jobs*machines))]}))  
}  
  
#-----  
# Função que faz mutação a cada elemento que constitui a população  
mutacao.populacao <-function(pop, Pm)  
{  
  t(apply(pop[(round(nrow(pop)*0.1) + 1):nrow(pop),], 1, mutacao.individuo, Pm))  
}  
  
#-----  
# Função de mutação de individuos  
mutacao.individuo<-function(individuo,Pm)  
{  
  # Faz mutação com probabilidade Pm  
  if (runif(1)<Pm)  
  {  
    # Shift mutation  
    # Selecciona um gene aleatoriamente  
    genes <- sample(1:length(individuo), 2)  
    guardaGene<-individuo[genes[1]]  
    # Vamos mover o gene seleccionado para a posição seleccionada  
    if (genes[1]<genes[2])  
    {  
      individuo <- replace(individuo, genes[1]:(genes[2]-1), individuo[(genes[1]+1):genes[2]])  
    }  
    else  
    {  
      individuo <- replace(individuo, (genes[2]+1):genes[1], individuo[genes[2]:(genes[1]-1)])  
    }  
    individuo[genes[2]]<-guardaGene  
  }  
}
```

```

    return(individuo)
}
#-----
# Função de cruzamento de individuos
cruzamento.individuos<-function(pop,npop,fitness)
{
  insere<-round(nrow(pop)*0.1) + 1
  while (insere<=nrow(pop))
  {
    # Operador de selecção: Binary Tournament
    # selecciona 2 individuos de forma aleatória, sendo o melhor deles o 1º pai
    presel <- sample(1:nrow(pop), 4)
    pai1 <- presel[which.max(fitness[presel[1:2]])]
    pai2 <- presel[which.max(fitness[presel[3:4]])+2]

    # Operador de Cruzamento: Two-Point Crossover
    # Definem-se dois pontos de corte
    points <- sort(sample(1:ncol(pop), 2))
    # Gerar primeiro filho
    npop[insere, points[1]:points[2]] <- pop[pai1, points[1]:points[2]]
    npop[insere,-(points[1]:points[2])]<-rev(rev(pop[pai2,])[-
(sort(pmatch(as.character(pop[pai1,points[1]:points[2]]),as.character(rev(pop[pai2,])),dup=FALSE)))]))
    if ((insere+1)<=nrow(pop))      # se ainda há espaço na população
    {
      # Gerar segundo filho
      npop[insere+1, points[1]:points[2]] <- pop[pai2, points[1]:points[2]]
      npop[insere+1,-(points[1]:points[2])<-rev(rev(pop[pai1,])[-
(sort(pmatch(as.character(pop[pai2,points[1]:points[2]]),as.character(rev(pop[pai1,])),dup=FALSE)))]))
    }
    insere<-insere+2
  }
  return(npop)
}

#-----
# Função que avalia cada elemento que constitui a população (valor de fitness)
avalia.populacao <-function(ficheiro2,nSolucoes,Solucoes,pop,inicio,process, tempos)
{
  apply(pop[inicio:nrow(pop),], 1, avalia.solucao, process, tempos,nSolucoes,Solucoes,ficheiro2)

```

```

}

#-----
avalia.solucao<-function(individuo,process,tempo,e1,Solucoes,ficheiro2)
{
  job.fim<-rep(0, nrow(process)) # Fim da última operação alocada de cada job
  mach.free<-rep(0, nrow(process)) # Fim da última operação alocada a cada máquina
  operacao.job<-rep(0, nrow(process)) # Indicar qual a próxima operação do job a executar
  nSolucoes<-get("a", env=e1)
  if (nSolucoes<Solucoes)
  {
    # Percorre o cromossoma, gene a gene, para determinar o makespan time
    for (j in 1:length(individuo))
    {
      gene <- individuo[j]
      #Atualiza a próxima operação do job a processar
      operacao.job[gene]<-operacao.job[gene]+1
      # Como uma máquina só pode processar um job de cada vez, e um job só pode processar um
      operação de cada vez
      ## Verifica o que fica livre primeiro, a máquina ou o job, e actualiza os tempos
      if (job.fim[gene]<mach.free[process[gene,operacao.job[gene]]])
      {
        mach.free[process[gene,operacao.job[gene]]]<-
        mach.free[process[gene,operacao.job[gene]]]+tempo[gene,operacao.job[gene]]
        job.fim[gene]<-mach.free[process[gene,operacao.job[gene]]]
      }
      else
      {
        job.fim[gene]<-job.fim[gene]+tempo[gene,operacao.job[gene]]
        mach.free[process[gene,operacao.job[gene]]]<-job.fim[gene]
      }
    }
    nSolucoes<-nSolucoes+1
    assign("a", nSolucoes, env=e1)
    cat(max(mach.free),"n",file = ficheiro2) # Guarda a fitness da solução no ficheiro 2
    # Verifica qual a máquina com o tempo maior para terminar o processamento->makespan time
    return(max(mach.free))
  }
  else

```

```

{
#   nSolucoes<-nSolucoes+1
#   assign("a", nSolucoes, env=e1)
  cat("9999","\n",file = ficheiro2)
  return(9999)
}
}

#-----
# Função principal do algoritmo genético
AG<-function(ficheiro1,ficheiro2,e1,process,tempo,Solucoes,popSize, Pm)
{
  inicio.proc<-proc.time()[3]
  # Inicializar população
  pop <- gera.populacao(popSize,nrow(process),ncol(process))
  fitness <- avalia.populacao(ficheiro2,e1,Solucoes,pop,1,process, tempo)
  nSolucoes<-get("a", env=e1)
  while (nSolucoes<Solucoes)
  {
    #Verifica a ordem dos elementos que compõe o vector de valores de fitness dos filhos gerados
    # e aplica a ordem à matriz de individuos e de fitness
    ordem<-order(fitness)
    pop<-pop[ordem,]
    fitness<-fitness[ordem]
    # Cria as estruturas de dados para a nova população
    npop <- matrix(0, nrow=nrow(pop), ncol=ncol(pop))
    # Faz o survival selection: selecciona os 10% melhores individuos da geração anterior
    npop[1:round(nrow(pop)*0.1),]<-pop[1:round(nrow(pop)*0.1),]
    npop <- cruzamento.individuos(pop,npop, fitness) # Selecção de pais e Crossover
    npop[(round(nrow(pop)*0.1) + 1):nrow(pop),]<-mutacao.populacao(npop,Pm)
    i<-1 # Guarda a fitness das soluções que vêm da geração anterior no ficheiro 2
    while (i<=round(nrow(pop)*0.1))
    {
      cat(fitness[i],"\n",file = ficheiro2)
      i<-i+1
    }
    nSolucoes<-nSolucoes+i-1
    assign("a", nSolucoes, env=e1)
  }
}

```

```

        fitness[(round(nrow(pop)*0.1) + 1):nrow(pop)]<-
avalia.populacao(ficheiro2,e1,Solucoes,npop,(round(nrow(pop)*0.1) + 1),process, tempos) #
Avaliar nova população
    pop<-npop
    nSolucoes<-get("a", env=e1)
    # Guarda dados da iteração no ficheiro de configurações
    cat(proc.time()[3]-inicio.proc,";",min(fitness)," ", " ", " ",pop[which.min(fitness)],"\n",file = ficheiro1)
  }
  return(pop[which.min(fitness),])
}

#-----
# Função principal Tabu Search
TS<-function(ficheiro1,ficheiro2,pred,tempos,Solucoes,popSize,Pm)
{
  # Ler problema
  process.df<- read.table(pred, header = FALSE, sep = ";")
  tempos.df<- read.table(tempos, header = FALSE, sep = ";")
  process <- matrix(unlist(process.df), nrow=nrow(process.df))
  tempos <- matrix(unlist(tempos.df), nrow=nrow(tempos.df))
  inicio.proc<-proc.time()[3]
  environment()
  e1 <- new.env(TRUE, NULL)
  assign("a", 0, env=e1)
  nSolucoes<-0
  # Avaliar solução inicial
  solucao.cur<-AG(ficheiro1,ficheiro2,e1,process,tempos,Solucoes/2,popSize, Pm)
  valor.cur <- avalia.solucao(solucao.cur, process ,tempos,e1,Solucoes,ficheiro2)
  nSolucoes<-get("a", env=e1)
  # Inicializar melhor solução
  solucao.best <- solucao.cur
  valor.best <- valor.cur
  cat(proc.time()[3]-inicio.proc,";",valor.best,";",valor.cur,";",solucao.cur,"\n",file = ficheiro1)
  # Criar a estrutura que guarda os movimentos tabu
  lista.tabu <-rep(0, nrow(process)*ncol(process))
  while (nSolucoes<Solucoes)
  {
    # Actualiza lista tabu
    lista.tabu[which(lista.tabu>0)]<-lista.tabu[which(lista.tabu>0)]-1

```

```

# Construir e avaliar todos os vizinhos
fitness.vizinhos <- sapply(1:(length(solucao.cur)-1), function(i, sc, p, t,e1,Solucoes,ficheiro2)
{
  v <- replace(sc, i:(i+1), sc[(i+1):i])
  avalia.solucao(v, p, t,e1,Solucoes,ficheiro2)
}, solucao.cur, process, tempos,e1,Solucoes,ficheiro2)

tabus <- sapply(1:(length(solucao.cur)-1), function(i, lt, sc) {lt[i]}, lista.tabu, solucao.cur)
# Próxima solução corrente é o vizinho que seja melhor que o best ou o melhor vizinho (não-tabu) ou,
se não houver nenhum, o que tiver menor tabu ou que for melhor que best
i.melhor.vizinho <-
ifelse(sum(which(fitness.vizinhos<valor.best))>0,which(fitness.vizinhos<valor.best),ifelse(all(lista.tabu >
0),
(1:(length(solucao.cur)-1))[tabus <= 0][which.min(fitness.vizinhos[tabus <= 0 ])],
(1:(length(solucao.cur)-1))[which.min(tabus)]))
#Actualiza a solução corrente
solucao.cur <- replace(solucao.cur, i.melhor.vizinho:(i.melhor.vizinho+1),
solucao.cur[(i.melhor.vizinho+1):i.melhor.vizinho])
valor.cur <- fitness.vizinhos[i.melhor.vizinho]
lista.tabu[i.melhor.vizinho] <- sqrt(length(solucao.cur))
# Actualiza melhor solução
if (valor.cur < valor.best)
{
  valor.best <- valor.cur
  solucao.best <- solucao.cur
}
# Guarda dados da iteração no ficheiro de configurações
cat(proc.time()[3]-inicio.proc,";",valor.best,";",valor.cur,";",solucao.cur,"\n",file = ficheiro1)
nSolucoes<-get("a", env=e1)
}
return(valor.best)
}
obtem.resultados<-function(problemas,parametros,repeticoes)
{
  fim.problemas<-FALSE
  hist.problemas<-as.matrix(read.table(problemas, header = FALSE, sep = ";"))
  while (fim.problemas==FALSE)
  {
    fim.parametros<-FALSE

```

```

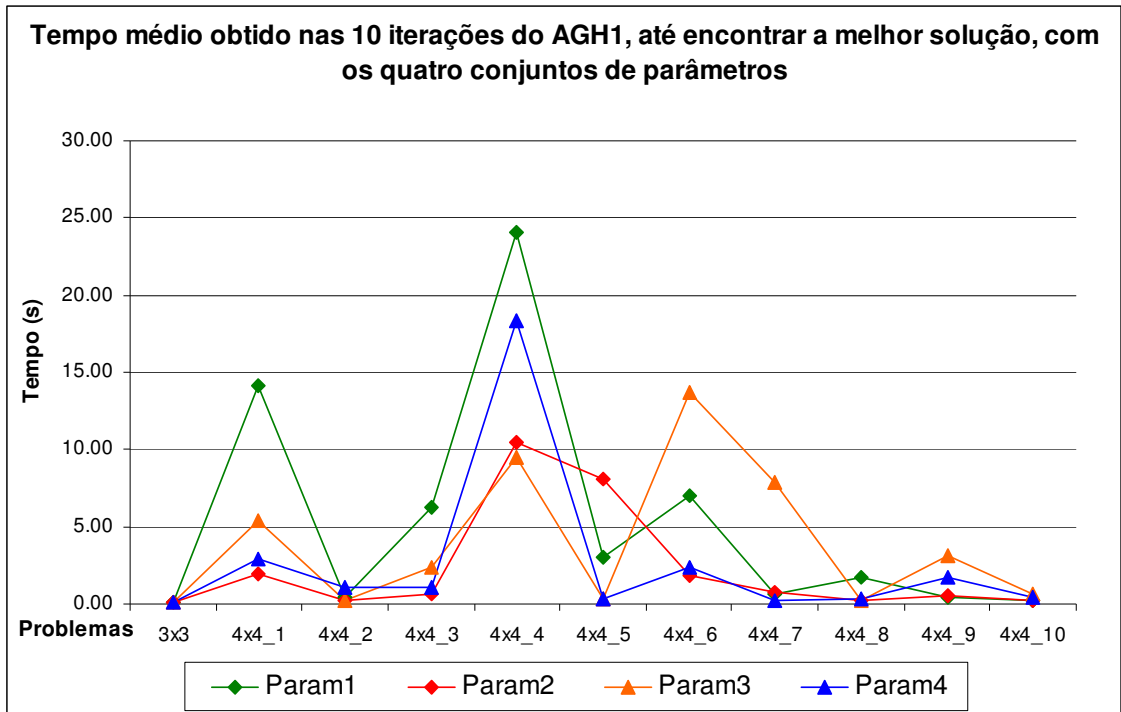
hist.parametros<-as.matrix(read.table(parametros, header = FALSE, sep = ";"))
while (fim.parametros==FALSE)
{
  fim.repeticoes<-FALSE
  hist.repeticoes<-as.matrix(read.table(repeticoes, header = FALSE, sep = ";"))
  while (fim.repeticoes==FALSE)
  {
    ficheiro1<-
file(paste("res1.",hist.problemas[which.min(hist.problemas[,2]),1],".param",which.min(hist.parametros[,4
]),".rep",which.min(hist.repeticoes[,2]),".txt",sep=""),"w")
    ficheiro2<-
file(paste("res2.",hist.problemas[which.min(hist.problemas[,2]),1],".param",which.min(hist.parametros[,4
]),".rep",which.min(hist.repeticoes[,2]),".txt",sep=""),"w")
    TS(ficheiro1,ficheiro2,paste("pred",hist.problemas[which.min(hist.problemas[,2]),1],".txt",sep=""),paste(
"tempos",hist.problemas[which.min(hist.problemas[,2]),1],".txt",sep=""),hist.parametros[which.min(hist.
parametros[,4]),3],hist.parametros[which.min(hist.parametros[,4]),2],hist.parametros[which.min(hist.para
metros[,4]),1])
    close(ficheiro1)
    close(ficheiro2)
    if (which.min(hist.repeticoes[,2])==nrow(hist.repeticoes))
    {
      hist.repeticoes[,2]<-0
      fim.repeticoes<-TRUE
    }
    else hist.repeticoes[which.min(hist.repeticoes[,2]),2]<-1
    write.table(hist.repeticoes, file = "hist.repeticoes.txt", sep = ";",row.names = FALSE, col.names =
FALSE)
  }
  if (which.min(hist.parametros[,4])==nrow(hist.parametros))
  {
    hist.parametros[,4]<-0
    fim.parametros<-TRUE
  }
  else hist.parametros[which.min(hist.parametros[,4]),4]<-1
  write.table(hist.parametros, file = "hist.parametros.txt", sep = ";",row.names = FALSE, col.names =
FALSE)
}
if (which.min(hist.problemas[,2])==nrow(hist.problemas)) fim.problemas<-TRUE
else hist.problemas[which.min(hist.problemas[,2]),2]<-1

```

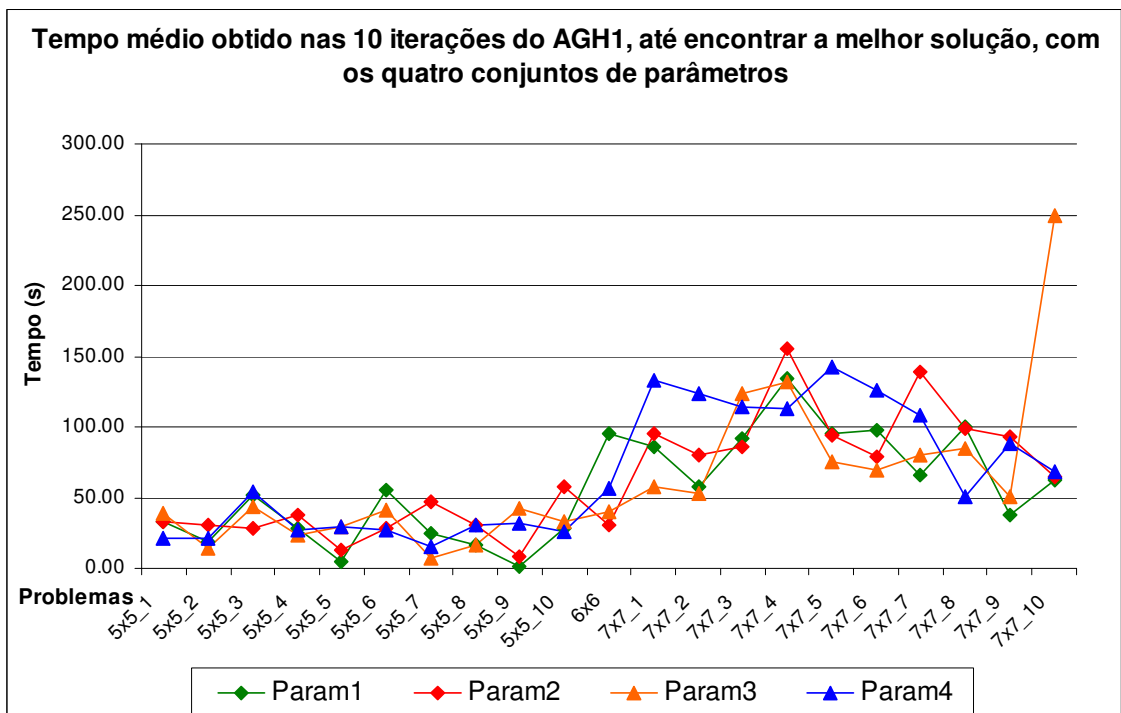
```
write.table(hist.problemas, file = "hist.problemas.txt", sep = ";",row.names = FALSE, col.names =  
FALSE)  
}  
}
```

```
obtem.resultados("hist.problemas.txt","hist.parametros.txt","hist.repeticoes.txt")
```

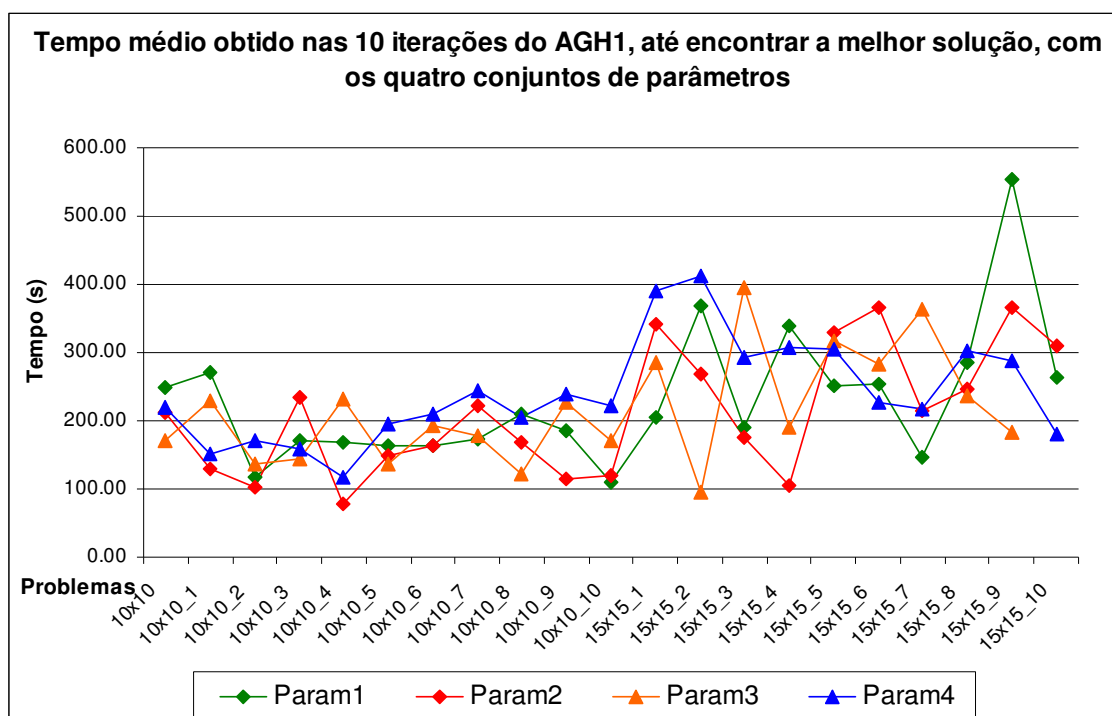
## Anexo 7 – Resultado dos tempos médios obtidos pelo algoritmo AGH1



Resultado dos tempos médios necessários para encontrar a melhor solução possível, quando usados os quatro conjuntos de parâmetros no AGH1, para os problemas do tipo 3x3 e 4x4.



Resultado dos tempos médios necessários para encontrar a melhor solução possível, quando usados os quatro conjuntos de parâmetros no AGH1, para os problemas do tipo 5x5, 6x6 e 7x7.



Resultado dos tempos médios necessários para encontrar a melhor solução possível, quando usados os quatro conjuntos de parâmetros no AGH1, para os problemas do tipo 10x10 e 15x15.

## Anexo 8 - Resultados experimentais obtidos nas 10 execuções de cada algoritmo

- Resultados obtidos quando executado o algoritmo AG com o 4º conjunto de parâmetros:

| Problema                 | Rep1 | Rep2 | Rep3 | Rep4 | Rep5 | Rep6 | Rep7 | Rep8 | Rep9 | Rep10 | Média  | Desvio | MAX  | MIN  |
|--------------------------|------|------|------|------|------|------|------|------|------|-------|--------|--------|------|------|
| res1.3x3.param4.txt      | 26   | 26   | 26   | 26   | 26   | 26   | 26   | 26   | 26   | 26    | 26.0   | 0.0    | 26   | 26   |
| res1.6x6.param4.txt      | 60   | 61   | 60   | 59   | 60   | 59   | 62   | 59   | 59   | 60    | 59.9   | 1.0    | 62   | 59   |
| res1.10x10.param4.txt    | 1262 | 1308 | 1337 | 1334 | 1322 | 1356 | 1321 | 1316 | 1344 | 1303  | 1320.3 | 26.2   | 1356 | 1262 |
| res1.4x4_1.param4.txt    | 272  | 272  | 272  | 272  | 272  | 272  | 272  | 286  | 272  | 272   | 273.4  | 4.4    | 286  | 272  |
| res1.4x4_2.param4.txt    | 312  | 312  | 312  | 312  | 312  | 312  | 312  | 312  | 312  | 312   | 312.0  | 0.0    | 312  | 312  |
| res1.4x4_3.param4.txt    | 309  | 309  | 309  | 309  | 309  | 309  | 309  | 309  | 309  | 309   | 309.0  | 0.0    | 309  | 309  |
| res1.4x4_4.param4.txt    | 391  | 390  | 390  | 390  | 390  | 391  | 390  | 394  | 390  | 391   | 390.7  | 1.3    | 394  | 390  |
| res1.4x4_5.param4.txt    | 356  | 356  | 356  | 356  | 356  | 356  | 356  | 356  | 356  | 356   | 356.0  | 0.0    | 356  | 356  |
| res1.4x4_6.param4.txt    | 274  | 274  | 274  | 274  | 274  | 274  | 274  | 274  | 274  | 274   | 274.0  | 0.0    | 274  | 274  |
| res1.4x4_7.param4.txt    | 252  | 252  | 252  | 252  | 252  | 252  | 252  | 252  | 252  | 252   | 252.0  | 0.0    | 252  | 252  |
| res1.4x4_8.param4.txt    | 267  | 267  | 267  | 267  | 267  | 267  | 267  | 267  | 267  | 267   | 267.0  | 0.0    | 267  | 267  |
| res1.4x4_9.param4.txt    | 336  | 336  | 344  | 336  | 342  | 336  | 336  | 336  | 336  | 336   | 337.4  | 3.0    | 344  | 336  |
| res1.4x4_10.param4.txt   | 285  | 285  | 285  | 285  | 285  | 285  | 285  | 285  | 285  | 285   | 285.0  | 0.0    | 285  | 285  |
| res1.5x5_1.param4.txt    | 469  | 449  | 449  | 464  | 437  | 485  | 438  | 469  | 428  | 473   | 456.1  | 18.6   | 485  | 428  |
| res1.5x5_2.param4.txt    | 361  | 333  | 352  | 368  | 333  | 333  | 333  | 363  | 355  | 333   | 346.4  | 14.8   | 368  | 333  |
| res1.5x5_3.param4.txt    | 510  | 511  | 527  | 506  | 527  | 506  | 507  | 510  | 508  | 524   | 513.6  | 8.8    | 527  | 506  |
| res1.5x5_4.param4.txt    | 471  | 471  | 471  | 486  | 470  | 470  | 470  | 479  | 470  | 465   | 472.3  | 5.9    | 486  | 465  |
| res1.5x5_5.param4.txt    | 474  | 461  | 432  | 420  | 420  | 449  | 439  | 440  | 420  | 420   | 437.5  | 19.1   | 474  | 420  |
| res1.5x5_6.param4.txt    | 443  | 419  | 427  | 443  | 460  | 445  | 393  | 427  | 443  | 427   | 432.7  | 18.5   | 460  | 393  |
| res1.5x5_7.param4.txt    | 475  | 494  | 467  | 478  | 475  | 516  | 489  | 475  | 476  | 475   | 482.0  | 14.2   | 516  | 467  |
| res1.5x5_8.param4.txt    | 392  | 408  | 397  | 392  | 392  | 397  | 392  | 408  | 411  | 392   | 398.1  | 7.8    | 411  | 392  |
| res1.5x5_9.param4.txt    | 514  | 502  | 509  | 502  | 502  | 502  | 502  | 502  | 522  | 502   | 505.9  | 7.0    | 522  | 502  |
| res1.5x5_10.param4.txt   | 482  | 480  | 493  | 487  | 482  | 480  | 489  | 482  | 493  | 482   | 485.0  | 5.1    | 493  | 480  |
| res1.7x7_1.param4.txt    | 659  | 706  | 656  | 688  | 689  | 681  | 680  | 681  | 685  | 717   | 684.2  | 18.4   | 717  | 656  |
| res1.7x7_2.param4.txt    | 727  | 727  | 715  | 675  | 680  | 691  | 695  | 732  | 731  | 686   | 705.9  | 22.7   | 732  | 675  |
| res1.7x7_3.param4.txt    | 836  | 835  | 857  | 872  | 834  | 827  | 855  | 888  | 834  | 844   | 848.2  | 19.6   | 888  | 827  |
| res1.7x7_4.param4.txt    | 761  | 799  | 806  | 796  | 774  | 779  | 789  | 746  | 784  | 749   | 778.3  | 20.8   | 806  | 746  |
| res1.7x7_5.param4.txt    | 660  | 670  | 689  | 678  | 666  | 674  | 628  | 687  | 663  | 688   | 670.3  | 18.2   | 689  | 628  |
| res1.7x7_6.param4.txt    | 690  | 698  | 681  | 712  | 677  | 695  | 685  | 653  | 717  | 679   | 688.7  | 18.4   | 717  | 653  |
| res1.7x7_7.param4.txt    | 661  | 652  | 674  | 678  | 712  | 699  | 703  | 658  | 658  | 654   | 674.9  | 22.3   | 712  | 652  |
| res1.7x7_8.param4.txt    | 653  | 624  | 686  | 669  | 678  | 673  | 662  | 661  | 662  | 644   | 661.2  | 17.8   | 686  | 624  |
| res1.7x7_9.param4.txt    | 696  | 705  | 655  | 668  | 678  | 680  | 677  | 656  | 724  | 719   | 685.8  | 24.4   | 724  | 655  |
| res1.7x7_10.param4.txt   | 620  | 613  | 614  | 593  | 596  | 634  | 625  | 624  | 609  | 594   | 612.2  | 14.2   | 634  | 593  |
| res1.10x10_1.param4.txt  | 1126 | 1105 | 1046 | 1103 | 1074 | 1108 | 1072 | 1119 | 1159 | 1090  | 1100.2 | 31.8   | 1159 | 1046 |
| res1.10x10_2.param4.txt  | 1130 | 1041 | 1019 | 1067 | 1098 | 1045 | 1019 | 1090 | 1082 | 1093  | 1068.4 | 36.7   | 1130 | 1019 |
| res1.10x10_3.param4.txt  | 1052 | 1074 | 1006 | 1117 | 1032 | 1078 | 1050 | 1049 | 1069 | 1067  | 1059.4 | 29.6   | 1117 | 1006 |
| res1.10x10_4.param4.txt  | 1079 | 1074 | 1049 | 1044 | 1084 | 1089 | 1014 | 1048 | 1134 | 1073  | 1068.8 | 32.5   | 1134 | 1014 |
| res1.10x10_5.param4.txt  | 1203 | 1136 | 1157 | 1205 | 1195 | 1140 | 1151 | 1163 | 1200 | 1191  | 1174.1 | 27.4   | 1205 | 1136 |
| res1.10x10_6.param4.txt  | 1038 | 1007 | 953  | 974  | 997  | 943  | 1038 | 1032 | 955  | 961   | 989.8  | 37.4   | 1038 | 943  |
| res1.10x10_7.param4.txt  | 1209 | 1180 | 1103 | 1112 | 1181 | 1139 | 1203 | 1165 | 1179 | 1206  | 1167.7 | 38.0   | 1209 | 1103 |
| res1.10x10_8.param4.txt  | 1034 | 1082 | 1075 | 1086 | 1064 | 1110 | 1068 | 1113 | 1064 | 1045  | 1074.1 | 25.2   | 1113 | 1034 |
| res1.10x10_9.param4.txt  | 1037 | 1055 | 1020 | 1076 | 1031 | 1025 | 1029 | 1066 | 1069 | 1062  | 1047.0 | 20.7   | 1076 | 1020 |
| res1.10x10_10.param4.txt | 1056 | 1105 | 1105 | 1130 | 1128 | 1007 | 1129 | 1011 | 1093 | 1085  | 1084.9 | 46.0   | 1130 | 1007 |
| res1.15x15_1.param4.txt  | 1945 | 1997 | 1948 | 1943 | 1921 | 1913 | 1903 | 1965 | 1972 | 1915  | 1942.2 | 29.9   | 1997 | 1903 |
| res1.15x15_2.param4.txt  | 1898 | 1922 | 1845 | 1942 | 1935 | 1957 | 1988 | 1946 | 1965 | 1874  | 1927.2 | 43.6   | 1988 | 1845 |
| res1.15x15_3.param4.txt  | 1849 | 1835 | 1897 | 1966 | 1890 | 1928 | 1893 | 1937 | 1954 | 1861  | 1901.0 | 44.6   | 1966 | 1835 |
| res1.15x15_4.param4.txt  | 1929 | 1918 | 1955 | 1904 | 1831 | 1887 | 1879 | 1890 | 1861 | 1959  | 1901.3 | 40.4   | 1959 | 1831 |
| res1.15x15_5.param4.txt  | 2000 | 1938 | 1848 | 1968 | 1864 | 1967 | 1899 | 1987 | 1937 | 1910  | 1931.8 | 51.1   | 2000 | 1848 |
| res1.15x15_6.param4.txt  | 1892 | 1878 | 1859 | 1816 | 1833 | 1894 | 1906 | 1939 | 1866 | 1892  | 1877.5 | 35.8   | 1939 | 1816 |
| res1.15x15_7.param4.txt  | 1956 | 1948 | 1942 | 1941 | 1947 | 1825 | 1946 | 1865 | 1776 | 1980  | 1912.6 | 66.9   | 1980 | 1776 |
| res1.15x15_8.param4.txt  | 2004 | 1952 | 1932 | 1911 | 1926 | 1951 | 1861 | 1984 | 1969 | 1974  | 1946.4 | 41.1   | 2004 | 1861 |
| res1.15x15_9.param4.txt  | 2081 | 1942 | 2034 | 2017 | 1925 | 2058 | 2016 | 1966 | 1968 | 2073  | 2008.0 | 55.3   | 2081 | 1925 |
| res1.15x15_10.param4.txt | 1912 | 1848 | 1925 | 1923 | 1804 | 1936 | 1861 | 1897 | 1838 | 1904  | 1884.8 | 44.2   | 1936 | 1804 |

- Resultados obtidos quando executado o algoritmo PT1:

| Problema                 | Rep1 | Rep2 | Rep3 | Rep4 | Rep5 | Rep6 | Rep7 | Rep8 | Rep9 | Rep10 | Média  | Desvio | MAX  | MIN  |
|--------------------------|------|------|------|------|------|------|------|------|------|-------|--------|--------|------|------|
| res1.3x3.param1.txt      | 26   | 29   | 29   | 26   | 26   | 26   | 35   | 26   | 26   | 26    | 27.5   | 2.8    | 35   | 26   |
| res1.6x6.param1.txt      | 68   | 77   | 97   | 79   | 64   | 76   | 71   | 75   | 92   | 84    | 78.3   | 9.8    | 97   | 64   |
| res1.10x10.param1.txt    | 1525 | 1477 | 1400 | 1572 | 1485 | 1432 | 1416 | 1600 | 1516 | 1274  | 1469.7 | 89.7   | 1600 | 1274 |
| res1.4x4_1.param1.txt    | 339  | 272  | 432  | 337  | 410  | 344  | 466  | 307  | 366  | 341   | 361.4  | 55.7   | 466  | 272  |
| res1.4x4_2.param1.txt    | 350  | 356  | 387  | 350  | 312  | 391  | 363  | 350  | 362  | 351   | 357.2  | 20.8   | 391  | 312  |
| res1.4x4_3.param1.txt    | 458  | 340  | 342  | 309  | 438  | 354  | 360  | 471  | 489  | 384   | 394.5  | 60.6   | 489  | 309  |
| res1.4x4_4.param1.txt    | 402  | 391  | 465  | 418  | 413  | 413  | 438  | 465  | 437  | 478   | 432.0  | 28.0   | 478  | 391  |
| res1.4x4_5.param1.txt    | 438  | 526  | 397  | 385  | 408  | 450  | 552  | 411  | 505  | 549   | 462.1  | 61.6   | 552  | 385  |
| res1.4x4_6.param1.txt    | 274  | 380  | 315  | 373  | 376  | 406  | 422  | 376  | 352  | 322   | 359.6  | 42.4   | 422  | 274  |
| res1.4x4_7.param1.txt    | 343  | 287  | 292  | 252  | 313  | 254  | 266  | 296  | 286  | 256   | 284.5  | 27.5   | 343  | 252  |
| res1.4x4_8.param1.txt    | 337  | 281  | 396  | 304  | 376  | 308  | 267  | 344  | 457  | 337   | 340.7  | 54.1   | 457  | 267  |
| res1.4x4_9.param1.txt    | 394  | 465  | 408  | 431  | 482  | 336  | 430  | 471  | 447  | 372   | 423.6  | 44.1   | 482  | 336  |
| res1.4x4_10.param1.txt   | 308  | 309  | 285  | 384  | 325  | 286  | 286  | 290  | 337  | 308   | 311.8  | 29.3   | 384  | 285  |
| res1.5x5_1.param1.txt    | 506  | 553  | 567  | 645  | 498  | 632  | 505  | 457  | 617  | 493   | 547.3  | 62.5   | 645  | 457  |
| res1.5x5_2.param1.txt    | 435  | 566  | 429  | 454  | 451  | 417  | 502  | 509  | 411  | 529   | 470.3  | 50.1   | 566  | 411  |
| res1.5x5_3.param1.txt    | 730  | 533  | 581  | 740  | 533  | 597  | 537  | 560  | 630  | 665   | 610.6  | 74.4   | 740  | 533  |
| res1.5x5_4.param1.txt    | 528  | 521  | 651  | 536  | 517  | 476  | 621  | 616  | 651  | 552   | 566.9  | 59.2   | 651  | 476  |
| res1.5x5_5.param1.txt    | 530  | 598  | 646  | 442  | 526  | 523  | 566  | 512  | 483  | 636   | 546.2  | 61.9   | 646  | 442  |
| res1.5x5_6.param1.txt    | 602  | 592  | 635  | 521  | 520  | 510  | 517  | 603  | 449  | 437   | 538.6  | 63.7   | 635  | 437  |
| res1.5x5_7.param1.txt    | 721  | 558  | 645  | 522  | 579  | 562  | 555  | 594  | 609  | 531   | 587.6  | 56.4   | 721  | 522  |
| res1.5x5_8.param1.txt    | 581  | 563  | 497  | 499  | 586  | 479  | 460  | 551  | 650  | 460   | 532.6  | 60.0   | 650  | 460  |
| res1.5x5_9.param1.txt    | 565  | 692  | 568  | 676  | 679  | 588  | 616  | 587  | 698  | 596   | 626.5  | 50.9   | 698  | 565  |
| res1.5x5_10.param1.txt   | 674  | 565  | 659  | 573  | 600  | 582  | 582  | 692  | 690  | 657   | 627.4  | 48.9   | 692  | 565  |
| res1.7x7_1.param1.txt    | 943  | 721  | 898  | 1014 | 885  | 982  | 927  | 857  | 665  | 797   | 868.9  | 106.1  | 1014 | 665  |
| res1.7x7_2.param1.txt    | 954  | 832  | 894  | 905  | 1017 | 1012 | 939  | 1068 | 907  | 898   | 942.6  | 67.4   | 1068 | 832  |
| res1.7x7_3.param1.txt    | 1143 | 938  | 1133 | 1154 | 943  | 964  | 1015 | 1006 | 1016 | 1055  | 1036.7 | 77.7   | 1154 | 938  |
| res1.7x7_4.param1.txt    | 857  | 885  | 1002 | 947  | 853  | 995  | 1102 | 998  | 962  | 853   | 945.4  | 78.6   | 1102 | 853  |
| res1.7x7_5.param1.txt    | 759  | 785  | 977  | 739  | 885  | 868  | 800  | 735  | 784  | 968   | 830.0  | 85.2   | 977  | 735  |
| res1.7x7_6.param1.txt    | 965  | 861  | 800  | 929  | 843  | 944  | 948  | 906  | 853  | 796   | 884.5  | 58.9   | 965  | 796  |
| res1.7x7_7.param1.txt    | 826  | 938  | 880  | 859  | 826  | 886  | 1052 | 911  | 819  | 704   | 870.1  | 86.0   | 1052 | 704  |
| res1.7x7_8.param1.txt    | 855  | 1000 | 658  | 984  | 863  | 937  | 870  | 737  | 797  | 874   | 857.5  | 100.5  | 1000 | 658  |
| res1.7x7_9.param1.txt    | 933  | 858  | 837  | 741  | 1155 | 763  | 808  | 853  | 1081 | 790   | 881.9  | 129.7  | 1155 | 741  |
| res1.7x7_10.param1.txt   | 853  | 813  | 835  | 842  | 861  | 839  | 631  | 657  | 814  | 759   | 790.4  | 78.2   | 861  | 631  |
| res1.10x10_1.param1.txt  | 1425 | 1312 | 1603 | 1147 | 1330 | 1356 | 1399 | 1214 | 1263 | 1451  | 1350.0 | 123.0  | 1603 | 1147 |
| res1.10x10_2.param1.txt  | 1312 | 1246 | 1168 | 1298 | 1094 | 1275 | 1199 | 1412 | 1351 | 1277  | 1263.2 | 87.1   | 1412 | 1094 |
| res1.10x10_3.param1.txt  | 1274 | 1338 | 1047 | 1333 | 1351 | 1303 | 1226 | 1159 | 1239 | 1130  | 1240.0 | 95.8   | 1351 | 1047 |
| res1.10x10_4.param1.txt  | 1564 | 1126 | 1343 | 1239 | 1055 | 1270 | 1322 | 1503 | 1353 | 1257  | 1303.2 | 145.8  | 1564 | 1055 |
| res1.10x10_5.param1.txt  | 1598 | 1360 | 1327 | 1476 | 1355 | 1385 | 1553 | 1440 | 1504 | 1241  | 1423.9 | 104.7  | 1598 | 1241 |
| res1.10x10_6.param1.txt  | 1268 | 1169 | 1115 | 1206 | 1277 | 1246 | 1160 | 1258 | 1066 | 1224  | 1198.9 | 66.6   | 1277 | 1066 |
| res1.10x10_7.param1.txt  | 1295 | 1443 | 1315 | 1657 | 1272 | 1447 | 1376 | 1309 | 1433 | 1760  | 1430.7 | 153.3  | 1760 | 1272 |
| res1.10x10_8.param1.txt  | 1221 | 1107 | 1525 | 1619 | 1390 | 1145 | 1129 | 1456 | 1330 | 1400  | 1332.2 | 168.0  | 1619 | 1107 |
| res1.10x10_9.param1.txt  | 1291 | 1319 | 1357 | 1389 | 1205 | 1166 | 1166 | 1514 | 1512 | 1231  | 1315.0 | 122.4  | 1514 | 1166 |
| res1.10x10_10.param1.txt | 1380 | 1298 | 1152 | 1260 | 1163 | 1350 | 1213 | 1199 | 1280 | 1481  | 1277.6 | 98.5   | 1481 | 1152 |
| res1.15x15_1.param1.txt  | 2202 | 2199 | 2244 | 1949 | 2138 | 2518 | 2161 | 1923 | 2028 | 2389  | 2175.1 | 175.3  | 2518 | 1923 |
| res1.15x15_2.param1.txt  | 2339 | 2330 | 2452 | 2197 | 2271 | 2255 | 2399 | 2053 | 2523 | 2479  | 2329.8 | 135.2  | 2523 | 2053 |
| res1.15x15_3.param1.txt  | 2050 | 2436 | 2088 | 2374 | 2149 | 2242 | 2137 | 2559 | 2330 | 2268  | 2263.3 | 154.8  | 2559 | 2050 |
| res1.15x15_4.param1.txt  | 2407 | 2181 | 2127 | 2301 | 2175 | 2380 | 2177 | 2421 | 2166 | 2574  | 2290.9 | 141.3  | 2574 | 2127 |
| res1.15x15_5.param1.txt  | 2415 | 2146 | 2251 | 2138 | 2218 | 2179 | 2205 | 2130 | 2060 | 2528  | 2227.0 | 134.6  | 2528 | 2060 |
| res1.15x15_6.param1.txt  | 2334 | 2245 | 1866 | 2146 | 2100 | 2207 | 1928 | 2130 | 2177 | 2112  | 2124.5 | 132.2  | 2334 | 1866 |
| res1.15x15_7.param1.txt  | 2106 | 2434 | 2444 | 2146 | 2397 | 1978 | 2513 | 2311 | 2569 | 2038  | 2293.6 | 199.8  | 2569 | 1978 |
| res1.15x15_8.param1.txt  | 2164 | 2364 | 2320 | 2259 | 2259 | 2513 | 2295 | 2047 | 2101 | 2366  | 2268.8 | 130.4  | 2513 | 2047 |
| res1.15x15_9.param1.txt  | 2310 | 2339 | 2229 | 2323 | 1989 | 2206 | 2540 | 2208 | 2297 | 2079  | 2252.0 | 143.0  | 2540 | 1989 |
| res1.15x15_10.param1.txt | 2199 | 2186 | 2275 | 2235 | 2480 | 1999 | 2205 | 2145 | 2673 | 2311  | 2270.8 | 177.4  | 2673 | 1999 |

- Resultados obtidos quando executado o algoritmo PT2:

| Problema                 | Rep1 | Rep2 | Rep3 | Rep4 | Rep5 | Rep6 | Rep7 | Rep8 | Rep9 | Rep10 | Média  | Desvio | MAX  | MIN  |
|--------------------------|------|------|------|------|------|------|------|------|------|-------|--------|--------|------|------|
| res1.3x3.param1.txt      | 26   | 26   | 26   | 26   | 26   | 26   | 26   | 26   | 26   | 26    | 26.0   | 0.0    | 26   | 26   |
| res1.6x6.param1.txt      | 60   | 60   | 59   | 60   | 63   | 62   | 60   | 58   | 63   | 59    | 60.4   | 1.6    | 63   | 58   |
| res1.10x10.param1.txt    | 1341 | 1184 | 1276 | 1320 | 1343 | 1312 | 1279 | 1352 | 1330 | 1198  | 1293.5 | 56.7   | 1352 | 1184 |
| res1.4x4_1.param1.txt    | 272  | 281  | 272  | 272  | 281  | 272  | 272  | 272  | 272  | 272   | 273.8  | 3.6    | 281  | 272  |
| res1.4x4_2.param1.txt    | 312  | 312  | 312  | 312  | 312  | 312  | 312  | 312  | 312  | 312   | 312.0  | 0.0    | 312  | 312  |
| res1.4x4_3.param1.txt    | 309  | 309  | 309  | 309  | 309  | 309  | 309  | 309  | 309  | 309   | 309.0  | 0.0    | 309  | 309  |
| res1.4x4_4.param1.txt    | 391  | 391  | 398  | 398  | 390  | 402  | 396  | 402  | 390  | 391   | 394.9  | 4.6    | 402  | 390  |
| res1.4x4_5.param1.txt    | 356  | 356  | 356  | 356  | 356  | 356  | 356  | 356  | 356  | 356   | 356.0  | 0.0    | 356  | 356  |
| res1.4x4_6.param1.txt    | 274  | 274  | 274  | 274  | 274  | 274  | 274  | 274  | 274  | 274   | 274.0  | 0.0    | 274  | 274  |
| res1.4x4_7.param1.txt    | 252  | 252  | 252  | 252  | 252  | 252  | 252  | 252  | 252  | 252   | 252.0  | 0.0    | 252  | 252  |
| res1.4x4_8.param1.txt    | 267  | 267  | 267  | 267  | 267  | 267  | 267  | 267  | 267  | 267   | 267.0  | 0.0    | 267  | 267  |
| res1.4x4_9.param1.txt    | 336  | 336  | 336  | 336  | 336  | 336  | 336  | 336  | 342  | 336   | 336.6  | 1.8    | 342  | 336  |
| res1.4x4_10.param1.txt   | 285  | 285  | 285  | 285  | 285  | 285  | 285  | 285  | 285  | 285   | 285.0  | 0.0    | 285  | 285  |
| res1.5x5_1.param1.txt    | 476  | 446  | 428  | 474  | 484  | 466  | 471  | 492  | 446  | 459   | 464.2  | 18.6   | 492  | 428  |
| res1.5x5_2.param1.txt    | 346  | 333  | 349  | 333  | 333  | 333  | 350  | 333  | 349  | 333   | 339.2  | 7.7    | 350  | 333  |
| res1.5x5_3.param1.txt    | 552  | 533  | 512  | 534  | 525  | 547  | 528  | 537  | 534  | 524   | 532.6  | 10.9   | 552  | 512  |
| res1.5x5_4.param1.txt    | 465  | 470  | 465  | 486  | 471  | 486  | 471  | 470  | 472  | 508   | 476.4  | 12.7   | 508  | 465  |
| res1.5x5_5.param1.txt    | 420  | 432  | 421  | 420  | 432  | 461  | 420  | 459  | 420  | 467   | 435.2  | 18.4   | 467  | 420  |
| res1.5x5_6.param1.txt    | 457  | 431  | 443  | 442  | 437  | 431  | 391  | 443  | 441  | 443   | 435.9  | 16.5   | 457  | 391  |
| res1.5x5_7.param1.txt    | 493  | 491  | 479  | 495  | 495  | 478  | 479  | 480  | 480  | 468   | 483.8  | 8.6    | 495  | 468  |
| res1.5x5_8.param1.txt    | 398  | 408  | 397  | 426  | 404  | 408  | 408  | 397  | 392  | 392   | 403.0  | 9.7    | 426  | 392  |
| res1.5x5_9.param1.txt    | 502  | 502  | 502  | 502  | 509  | 502  | 514  | 502  | 516  | 509   | 506.0  | 5.3    | 516  | 502  |
| res1.5x5_10.param1.txt   | 487  | 487  | 487  | 482  | 480  | 491  | 482  | 480  | 482  | 482   | 484.0  | 3.5    | 491  | 480  |
| res1.7x7_1.param1.txt    | 693  | 655  | 698  | 710  | 637  | 675  | 694  | 687  | 649  | 678   | 677.6  | 22.5   | 710  | 637  |
| res1.7x7_2.param1.txt    | 665  | 718  | 702  | 683  | 725  | 739  | 742  | 698  | 698  | 684   | 705.4  | 23.9   | 742  | 665  |
| res1.7x7_3.param1.txt    | 863  | 821  | 897  | 848  | 844  | 882  | 826  | 856  | 866  | 865   | 856.8  | 22.2   | 897  | 821  |
| res1.7x7_4.param1.txt    | 784  | 780  | 740  | 701  | 781  | 727  | 779  | 729  | 699  | 784   | 750.4  | 33.3   | 784  | 699  |
| res1.7x7_5.param1.txt    | 671  | 678  | 694  | 706  | 667  | 669  | 646  | 694  | 643  | 651   | 671.9  | 20.4   | 706  | 643  |
| res1.7x7_6.param1.txt    | 718  | 653  | 691  | 688  | 684  | 717  | 670  | 742  | 723  | 686   | 697.2  | 25.7   | 742  | 653  |
| res1.7x7_7.param1.txt    | 676  | 652  | 675  | 670  | 714  | 631  | 713  | 721  | 665  | 700   | 681.7  | 28.0   | 721  | 631  |
| res1.7x7_8.param1.txt    | 709  | 671  | 676  | 708  | 700  | 714  | 695  | 700  | 692  | 702   | 696.7  | 13.2   | 714  | 671  |
| res1.7x7_9.param1.txt    | 635  | 698  | 703  | 649  | 710  | 701  | 709  | 682  | 648  | 670   | 680.5  | 26.7   | 710  | 635  |
| res1.7x7_10.param1.txt   | 612  | 635  | 593  | 547  | 604  | 613  | 650  | 639  | 649  | 625   | 616.7  | 29.4   | 650  | 547  |
| res1.10x10_1.param1.txt  | 1046 | 1090 | 1081 | 997  | 1054 | 1106 | 1060 | 1029 | 1067 | 1093  | 1062.3 | 31.2   | 1106 | 997  |
| res1.10x10_2.param1.txt  | 1109 | 1009 | 1114 | 1117 | 1055 | 1040 | 1091 | 1068 | 1079 | 1015  | 1069.7 | 37.5   | 1117 | 1009 |
| res1.10x10_3.param1.txt  | 1060 | 1031 | 1045 | 1085 | 1088 | 1040 | 933  | 1049 | 1086 | 1000  | 1041.7 | 44.8   | 1088 | 933  |
| res1.10x10_4.param1.txt  | 977  | 1031 | 1069 | 1012 | 988  | 1020 | 1042 | 1075 | 1040 | 1081  | 1033.5 | 33.6   | 1081 | 977  |
| res1.10x10_5.param1.txt  | 1183 | 1220 | 1191 | 1213 | 1192 | 1159 | 1151 | 1114 | 1183 | 1119  | 1172.5 | 34.3   | 1220 | 1114 |
| res1.10x10_6.param1.txt  | 944  | 862  | 882  | 992  | 1000 | 955  | 1015 | 896  | 992  | 951   | 948.9  | 50.6   | 1015 | 862  |
| res1.10x10_7.param1.txt  | 1123 | 1127 | 1051 | 1195 | 1130 | 1139 | 1149 | 1149 | 1153 | 1133  | 1134.9 | 34.1   | 1195 | 1051 |
| res1.10x10_8.param1.txt  | 1025 | 1046 | 990  | 1044 | 1046 | 1017 | 992  | 1097 | 1062 | 1044  | 1036.3 | 30.4   | 1097 | 990  |
| res1.10x10_9.param1.txt  | 968  | 1026 | 1007 | 1049 | 1047 | 1046 | 965  | 1080 | 997  | 990   | 1017.5 | 36.3   | 1080 | 965  |
| res1.10x10_10.param1.txt | 1093 | 1102 | 1113 | 1026 | 1085 | 1072 | 1069 | 1059 | 1038 | 1119  | 1077.6 | 29.2   | 1119 | 1026 |
| res1.15x15_1.param1.txt  | 1824 | 1785 | 1908 | 1814 | 1963 | 1905 | 1790 | 1839 | 1927 | 1866  | 1862.1 | 58.1   | 1963 | 1785 |
| res1.15x15_2.param1.txt  | 1858 | 1859 | 1819 | 1990 | 1905 | 1918 | 1950 | 1858 | 1819 | 1901  | 1887.7 | 52.7   | 1990 | 1819 |
| res1.15x15_3.param1.txt  | 1862 | 1925 | 1902 | 1940 | 1822 | 1893 | 1855 | 1805 | 1931 | 1770  | 1870.5 | 54.8   | 1940 | 1770 |
| res1.15x15_4.param1.txt  | 1809 | 1851 | 1936 | 1880 | 1844 | 1803 | 1951 | 1904 | 1919 | 1769  | 1866.6 | 58.2   | 1951 | 1769 |
| res1.15x15_5.param1.txt  | 1931 | 1981 | 1943 | 1817 | 1965 | 1903 | 1766 | 1911 | 1977 | 1811  | 1900.5 | 72.4   | 1981 | 1766 |
| res1.15x15_6.param1.txt  | 1802 | 1906 | 1862 | 1949 | 1871 | 1907 | 1859 | 1746 | 1812 | 1857  | 1857.1 | 55.6   | 1949 | 1746 |
| res1.15x15_7.param1.txt  | 1907 | 1837 | 1830 | 1742 | 1883 | 1914 | 1931 | 1773 | 1852 | 1764  | 1843.3 | 63.3   | 1931 | 1742 |
| res1.15x15_8.param1.txt  | 2012 | 1935 | 1935 | 1877 | 1880 | 1797 | 1948 | 1863 | 1823 | 1881  | 1895.1 | 60.2   | 2012 | 1797 |
| res1.15x15_9.param1.txt  | 1843 | 1870 | 1908 | 1911 | 1884 | 1916 | 1842 | 1948 | 2003 | 1955  | 1908.0 | 48.5   | 2003 | 1842 |
| res1.15x15_10.param1.txt | 1803 | 1944 | 1984 | 1841 | 1930 | 1926 | 1908 | 1866 | 1949 | 1925  | 1907.6 | 52.1   | 1984 | 1803 |

- Resultados obtidos quando executado o algoritmo AGH1 com o 4º conjunto de parâmetros:

| Problema                 | Rep1 | Rep2 | Rep3 | Rep4 | Rep5 | Rep6 | Rep7 | Rep8 | Rep9 | Rep10 | Média  | Desvio | MAX  | MIN  |
|--------------------------|------|------|------|------|------|------|------|------|------|-------|--------|--------|------|------|
| res1.3x3.param4.txt      | 26   | 26   | 26   | 26   | 26   | 26   | 26   | 26   | 26   | 26    | 26.0   | 0.0    | 26   | 26   |
| res1.6x6.param4.txt      | 59   | 60   | 59   | 58   | 58   | 60   | 60   | 58   | 59   | 58    | 58.9   | 0.8    | 60   | 58   |
| res1.10x10.param4.txt    | 1310 | 1269 | 1304 | 1293 | 1299 | 1230 | 1268 | 1273 | 1289 | 1295  | 1283.0 | 22.4   | 1310 | 1230 |
| res1.4x4_1.param4.txt    | 272  | 272  | 272  | 272  | 272  | 272  | 272  | 272  | 272  | 272   | 272.0  | 0.0    | 272  | 272  |
| res1.4x4_2.param4.txt    | 312  | 312  | 312  | 312  | 312  | 312  | 312  | 312  | 312  | 312   | 312.0  | 0.0    | 312  | 312  |
| res1.4x4_3.param4.txt    | 309  | 309  | 309  | 309  | 309  | 309  | 311  | 309  | 309  | 309   | 309.2  | 0.6    | 311  | 309  |
| res1.4x4_4.param4.txt    | 390  | 390  | 390  | 390  | 390  | 390  | 390  | 390  | 390  | 390   | 390.0  | 0.0    | 390  | 390  |
| res1.4x4_5.param4.txt    | 356  | 356  | 356  | 356  | 356  | 356  | 356  | 356  | 356  | 356   | 356.0  | 0.0    | 356  | 356  |
| res1.4x4_6.param4.txt    | 275  | 274  | 274  | 274  | 274  | 274  | 274  | 274  | 274  | 274   | 274.1  | 0.3    | 275  | 274  |
| res1.4x4_7.param4.txt    | 252  | 252  | 252  | 252  | 252  | 252  | 252  | 252  | 252  | 252   | 252.0  | 0.0    | 252  | 252  |
| res1.4x4_8.param4.txt    | 267  | 267  | 267  | 267  | 267  | 267  | 267  | 267  | 267  | 267   | 267.0  | 0.0    | 267  | 267  |
| res1.4x4_9.param4.txt    | 336  | 336  | 336  | 336  | 336  | 336  | 336  | 336  | 336  | 336   | 336.0  | 0.0    | 336  | 336  |
| res1.4x4_10.param4.txt   | 285  | 285  | 285  | 285  | 285  | 285  | 285  | 285  | 285  | 285   | 285.0  | 0.0    | 285  | 285  |
| res1.5x5_1.param4.txt    | 442  | 486  | 449  | 474  | 474  | 466  | 449  | 457  | 428  | 437   | 456.2  | 17.6   | 486  | 428  |
| res1.5x5_2.param4.txt    | 333  | 333  | 333  | 333  | 333  | 333  | 349  | 333  | 333  | 333   | 334.6  | 4.8    | 349  | 333  |
| res1.5x5_3.param4.txt    | 510  | 510  | 510  | 506  | 506  | 528  | 518  | 506  | 510  | 506   | 511.0  | 6.6    | 528  | 506  |
| res1.5x5_4.param4.txt    | 470  | 461  | 465  | 465  | 461  | 472  | 471  | 461  | 465  | 461   | 465.2  | 4.2    | 472  | 461  |
| res1.5x5_5.param4.txt    | 420  | 420  | 432  | 442  | 420  | 420  | 440  | 420  | 420  | 420   | 425.4  | 8.6    | 442  | 420  |
| res1.5x5_6.param4.txt    | 437  | 437  | 390  | 443  | 391  | 427  | 439  | 445  | 427  | 435   | 427.1  | 19.1   | 445  | 390  |
| res1.5x5_7.param4.txt    | 479  | 475  | 456  | 495  | 465  | 455  | 467  | 471  | 488  | 482   | 473.3  | 12.4   | 495  | 455  |
| res1.5x5_8.param4.txt    | 392  | 392  | 392  | 398  | 392  | 392  | 392  | 392  | 392  | 392   | 392.6  | 1.8    | 398  | 392  |
| res1.5x5_9.param4.txt    | 502  | 502  | 502  | 502  | 502  | 502  | 502  | 502  | 502  | 502   | 502.0  | 0.0    | 502  | 502  |
| res1.5x5_10.param4.txt   | 487  | 480  | 487  | 482  | 487  | 480  | 482  | 479  | 487  | 480   | 483.1  | 3.3    | 487  | 479  |
| res1.7x7_1.param4.txt    | 684  | 630  | 667  | 652  | 669  | 683  | 686  | 678  | 673  | 650   | 667.2  | 17.2   | 686  | 630  |
| res1.7x7_2.param4.txt    | 705  | 742  | 683  | 663  | 701  | 712  | 699  | 706  | 719  | 715   | 704.5  | 20.0   | 742  | 663  |
| res1.7x7_3.param4.txt    | 840  | 799  | 825  | 818  | 832  | 814  | 778  | 844  | 809  | 841   | 820.0  | 19.9   | 844  | 778  |
| res1.7x7_4.param4.txt    | 779  | 781  | 737  | 758  | 772  | 749  | 731  | 753  | 749  | 742   | 755.1  | 16.4   | 781  | 731  |
| res1.7x7_5.param4.txt    | 664  | 651  | 646  | 654  | 652  | 635  | 686  | 648  | 647  | 643   | 652.6  | 13.2   | 686  | 635  |
| res1.7x7_6.param4.txt    | 708  | 687  | 696  | 687  | 660  | 670  | 664  | 696  | 690  | 691   | 684.9  | 14.6   | 708  | 660  |
| res1.7x7_7.param4.txt    | 660  | 693  | 665  | 638  | 680  | 716  | 688  | 652  | 665  | 629   | 668.6  | 25.0   | 716  | 629  |
| res1.7x7_8.param4.txt    | 670  | 638  | 624  | 649  | 645  | 650  | 633  | 691  | 697  | 662   | 655.9  | 22.8   | 697  | 624  |
| res1.7x7_9.param4.txt    | 705  | 706  | 699  | 650  | 694  | 681  | 697  | 674  | 666  | 685   | 685.7  | 17.3   | 706  | 650  |
| res1.7x7_10.param4.txt   | 617  | 598  | 629  | 613  | 604  | 595  | 625  | 630  | 598  | 635   | 614.4  | 14.2   | 635  | 595  |
| res1.10x10_1.param4.txt  | 1057 | 1104 | 1128 | 1048 | 1131 | 1063 | 1097 | 1113 | 1125 | 1097  | 1096.3 | 28.9   | 1131 | 1048 |
| res1.10x10_2.param4.txt  | 1101 | 1081 | 1075 | 1091 | 1106 | 1106 | 1041 | 1093 | 1143 | 1076  | 1091.3 | 25.2   | 1143 | 1041 |
| res1.10x10_3.param4.txt  | 1053 | 1033 | 1096 | 1034 | 1057 | 1030 | 1055 | 995  | 1069 | 1025  | 1044.7 | 26.2   | 1096 | 995  |
| res1.10x10_4.param4.txt  | 1040 | 989  | 1050 | 957  | 1015 | 1046 | 1015 | 1029 | 1054 | 1046  | 1024.1 | 29.5   | 1054 | 957  |
| res1.10x10_5.param4.txt  | 1232 | 1174 | 1152 | 1135 | 1133 | 1185 | 1173 | 1257 | 1131 | 1132  | 1170.4 | 41.9   | 1257 | 1131 |
| res1.10x10_6.param4.txt  | 1004 | 916  | 958  | 989  | 962  | 983  | 952  | 986  | 969  | 1012  | 973.1  | 26.6   | 1012 | 916  |
| res1.10x10_7.param4.txt  | 1139 | 1177 | 1152 | 1188 | 1131 | 1217 | 1133 | 1082 | 1170 | 1143  | 1153.2 | 35.2   | 1217 | 1082 |
| res1.10x10_8.param4.txt  | 1064 | 1106 | 1024 | 1040 | 1058 | 1050 | 1033 | 1001 | 1006 | 1018  | 1040.0 | 29.7   | 1106 | 1001 |
| res1.10x10_9.param4.txt  | 1064 | 1029 | 1065 | 961  | 1048 | 1063 | 1052 | 1023 | 1063 | 1045  | 1041.3 | 30.2   | 1065 | 961  |
| res1.10x10_10.param4.txt | 1068 | 1105 | 1025 | 1054 | 1108 | 1099 | 1089 | 1055 | 1056 | 1105  | 1076.4 | 27.2   | 1108 | 1025 |
| res1.15x15_1.param4.txt  | 1933 | 1912 | 1710 | 1927 | 1927 | 1900 | 1913 | 1961 | 1921 | 1890  | 1899.4 | 65.7   | 1961 | 1710 |
| res1.15x15_2.param4.txt  | 1878 | 1895 | 1997 | 1853 | 1967 | 1906 | 1802 | 1904 | 1944 | 1892  | 1903.8 | 53.0   | 1997 | 1802 |
| res1.15x15_3.param4.txt  | 1902 | 1928 | 1962 | 1771 | 1907 | 1911 | 1950 | 2013 | 1907 | 1954  | 1920.5 | 59.6   | 2013 | 1771 |
| res1.15x15_4.param4.txt  | 1946 | 1869 | 1924 | 1930 | 1904 | 1891 | 1907 | 1898 | 1882 | 1857  | 1900.8 | 26.2   | 1946 | 1857 |
| res1.15x15_5.param4.txt  | 1939 | 1906 | 1908 | 1864 | 1882 | 1903 | 1935 | 1945 | 2041 | 1955  | 1927.8 | 46.6   | 2041 | 1864 |
| res1.15x15_6.param4.txt  | 1945 | 1959 | 1966 | 1806 | 1919 | 1943 | 1880 | 2001 | 1927 | 1973  | 1931.9 | 52.3   | 2001 | 1806 |
| res1.15x15_7.param4.txt  | 1929 | 1934 | 1865 | 1940 | 1940 | 1925 | 1937 | 1855 | 1877 | 1935  | 1913.7 | 32.1   | 1940 | 1855 |
| res1.15x15_8.param4.txt  | 1885 | 2014 | 1957 | 1881 | 1966 | 1903 | 1919 | 1951 | 1975 | 2007  | 1945.8 | 45.0   | 2014 | 1881 |
| res1.15x15_9.param4.txt  | 2021 | 1997 | 2031 | 2098 | 1941 | 2029 | 1911 | 2005 | 1980 | 1946  | 1995.9 | 51.4   | 2098 | 1911 |
| res1.15x15_10.param4.txt | 1950 | 1954 | 1861 | 1967 | 1945 | 1981 | 1874 | 1921 | 2004 | 1977  | 1943.4 | 43.6   | 2004 | 1861 |

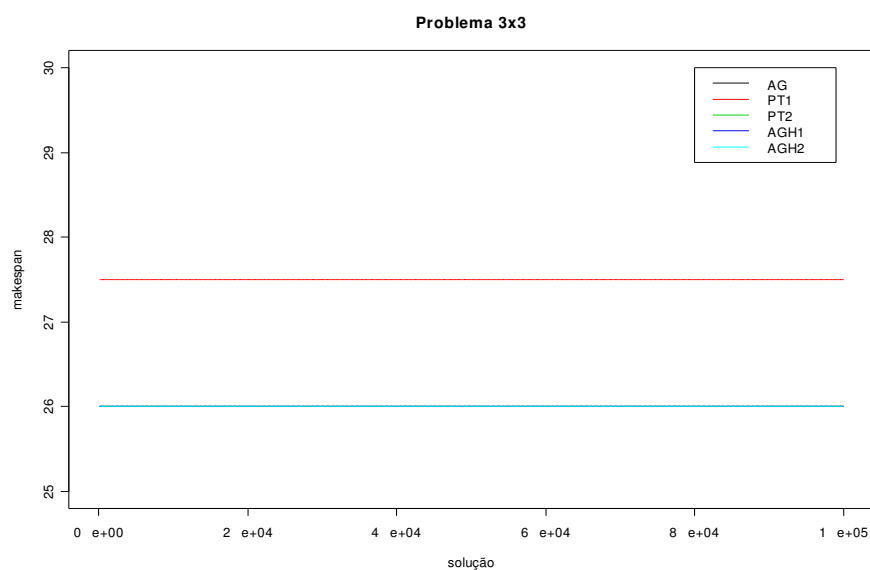
- Resultados obtidos quando executado o algoritmo AGH2 com o 4º conjunto de parâmetros:

| Problema                 | Rep1 | Rep2 | Rep3 | Rep4 | Rep5 | Rep6 | Rep7 | Rep8 | Rep9 | Rep10 | Média  | Desvio | MAX  | MIN  |
|--------------------------|------|------|------|------|------|------|------|------|------|-------|--------|--------|------|------|
| res1.3x3.param4.txt      | 26   | 26   | 26   | 26   | 26   | 26   | 26   | 26   | 26   | 26    | 26.0   | 0.0    | 26   | 26   |
| res1.6x6.param4.txt      | 59   | 64   | 61   | 61   | 61   | 58   | 60   | 60   | 60   | 61    | 60.5   | 1.5    | 64   | 58   |
| res1.10x10.param4.txt    | 1317 | 1172 | 1368 | 1268 | 1358 | 1260 | 1312 | 1299 | 1295 | 1294  | 1294.3 | 52.1   | 1368 | 1172 |
| res1.4x4_1.param4.txt    | 286  | 272  | 272  | 272  | 272  | 272  | 272  | 272  | 272  | 272   | 273.4  | 4.2    | 286  | 272  |
| res1.4x4_2.param4.txt    | 312  | 312  | 312  | 312  | 312  | 312  | 312  | 312  | 312  | 312   | 312.0  | 0.0    | 312  | 312  |
| res1.4x4_3.param4.txt    | 309  | 309  | 309  | 309  | 309  | 309  | 309  | 309  | 309  | 309   | 309.0  | 0.0    | 309  | 309  |
| res1.4x4_4.param4.txt    | 390  | 390  | 390  | 391  | 390  | 390  | 391  | 402  | 390  | 391   | 391.5  | 3.5    | 402  | 390  |
| res1.4x4_5.param4.txt    | 356  | 356  | 356  | 356  | 356  | 356  | 356  | 356  | 356  | 356   | 356.0  | 0.0    | 356  | 356  |
| res1.4x4_6.param4.txt    | 274  | 274  | 274  | 274  | 274  | 274  | 274  | 275  | 274  | 274   | 274.1  | 0.3    | 275  | 274  |
| res1.4x4_7.param4.txt    | 252  | 252  | 252  | 252  | 252  | 252  | 252  | 252  | 252  | 252   | 252.0  | 0.0    | 252  | 252  |
| res1.4x4_8.param4.txt    | 267  | 267  | 267  | 267  | 267  | 267  | 267  | 267  | 267  | 267   | 267.0  | 0.0    | 267  | 267  |
| res1.4x4_9.param4.txt    | 336  | 336  | 342  | 336  | 336  | 336  | 342  | 342  | 336  | 336   | 337.8  | 2.7    | 342  | 336  |
| res1.4x4_10.param4.txt   | 285  | 285  | 285  | 285  | 285  | 285  | 285  | 285  | 285  | 285   | 285.0  | 0.0    | 285  | 285  |
| res1.5x5_1.param4.txt    | 474  | 455  | 471  | 446  | 481  | 484  | 467  | 480  | 428  | 430   | 461.6  | 19.8   | 484  | 428  |
| res1.5x5_2.param4.txt    | 349  | 342  | 341  | 355  | 361  | 355  | 342  | 349  | 333  | 355   | 348.2  | 8.1    | 361  | 333  |
| res1.5x5_3.param4.txt    | 518  | 513  | 524  | 528  | 536  | 524  | 524  | 510  | 506  | 525   | 520.8  | 8.6    | 536  | 506  |
| res1.5x5_4.param4.txt    | 496  | 474  | 465  | 465  | 471  | 470  | 465  | 472  | 497  | 474   | 474.9  | 11.3   | 497  | 465  |
| res1.5x5_5.param4.txt    | 420  | 444  | 421  | 444  | 420  | 420  | 421  | 432  | 420  | 444   | 428.6  | 10.7   | 444  | 420  |
| res1.5x5_6.param4.txt    | 391  | 455  | 445  | 429  | 443  | 391  | 445  | 437  | 445  | 437   | 431.8  | 21.4   | 455  | 391  |
| res1.5x5_7.param4.txt    | 479  | 481  | 489  | 491  | 455  | 479  | 482  | 467  | 467  | 479   | 476.9  | 10.4   | 491  | 455  |
| res1.5x5_8.param4.txt    | 409  | 392  | 392  | 404  | 397  | 392  | 397  | 404  | 392  | 392   | 397.1  | 6.1    | 409  | 392  |
| res1.5x5_9.param4.txt    | 513  | 502  | 516  | 502  | 502  | 502  | 502  | 532  | 502  | 502   | 507.5  | 9.6    | 532  | 502  |
| res1.5x5_10.param4.txt   | 487  | 503  | 507  | 482  | 489  | 487  | 480  | 482  | 482  | 487   | 488.6  | 8.7    | 507  | 480  |
| res1.7x7_1.param4.txt    | 644  | 680  | 662  | 694  | 722  | 656  | 653  | 675  | 697  | 679   | 676.2  | 22.5   | 722  | 644  |
| res1.7x7_2.param4.txt    | 729  | 697  | 730  | 684  | 747  | 682  | 705  | 691  | 733  | 729   | 712.7  | 22.3   | 747  | 682  |
| res1.7x7_3.param4.txt    | 903  | 854  | 846  | 832  | 795  | 790  | 821  | 849  | 842  | 850   | 838.2  | 30.5   | 903  | 790  |
| res1.7x7_4.param4.txt    | 713  | 751  | 755  | 752  | 839  | 765  | 752  | 759  | 750  | 777   | 761.3  | 30.1   | 839  | 713  |
| res1.7x7_5.param4.txt    | 617  | 687  | 681  | 632  | 689  | 666  | 666  | 648  | 689  | 673   | 664.8  | 23.7   | 689  | 617  |
| res1.7x7_6.param4.txt    | 718  | 684  | 746  | 695  | 698  | 715  | 653  | 704  | 714  | 726   | 705.3  | 24.0   | 746  | 653  |
| res1.7x7_7.param4.txt    | 657  | 635  | 694  | 681  | 673  | 699  | 671  | 657  | 660  | 622   | 664.9  | 22.9   | 699  | 622  |
| res1.7x7_8.param4.txt    | 674  | 662  | 686  | 642  | 662  | 671  | 691  | 716  | 693  | 669   | 676.6  | 19.6   | 716  | 642  |
| res1.7x7_9.param4.txt    | 681  | 678  | 690  | 647  | 732  | 614  | 682  | 704  | 675  | 703   | 680.6  | 30.6   | 732  | 614  |
| res1.7x7_10.param4.txt   | 633  | 568  | 625  | 638  | 611  | 610  | 626  | 621  | 568  | 607   | 610.7  | 23.3   | 638  | 568  |
| res1.10x10_1.param4.txt  | 1050 | 1056 | 1138 | 1105 | 1179 | 1101 | 1110 | 1134 | 1075 | 1187  | 1113.5 | 44.5   | 1187 | 1050 |
| res1.10x10_2.param4.txt  | 1097 | 992  | 1140 | 1061 | 1017 | 1032 | 1089 | 1049 | 1151 | 1072  | 1070.0 | 48.4   | 1151 | 992  |
| res1.10x10_3.param4.txt  | 993  | 1085 | 997  | 1076 | 1041 | 1001 | 1109 | 1036 | 1066 | 1110  | 1051.4 | 42.4   | 1110 | 993  |
| res1.10x10_4.param4.txt  | 1052 | 966  | 1060 | 1055 | 1009 | 1056 | 1042 | 1089 | 1028 | 1039  | 1039.6 | 31.6   | 1089 | 966  |
| res1.10x10_5.param4.txt  | 1181 | 1252 | 1275 | 1227 | 1164 | 1182 | 1211 | 1295 | 1259 | 1138  | 1218.4 | 49.1   | 1295 | 1138 |
| res1.10x10_6.param4.txt  | 998  | 996  | 975  | 1015 | 1020 | 985  | 1050 | 971  | 1054 | 953   | 1001.7 | 31.5   | 1054 | 953  |
| res1.10x10_7.param4.txt  | 1186 | 1178 | 1177 | 1184 | 1181 | 1148 | 1159 | 1152 | 1187 | 1143  | 1169.5 | 16.2   | 1187 | 1143 |
| res1.10x10_8.param4.txt  | 1051 | 987  | 1042 | 1045 | 1088 | 1077 | 1113 | 1048 | 1006 | 1036  | 1049.3 | 35.1   | 1113 | 987  |
| res1.10x10_9.param4.txt  | 1036 | 1023 | 1093 | 1082 | 1027 | 1016 | 1032 | 1028 | 1012 | 1030  | 1037.9 | 25.8   | 1093 | 1012 |
| res1.10x10_10.param4.txt | 1098 | 1075 | 1054 | 1070 | 1093 | 1081 | 1109 | 1039 | 1098 | 1092  | 1080.9 | 20.7   | 1109 | 1039 |
| res1.15x15_1.param4.txt  | 1831 | 1845 | 2025 | 1951 | 1886 | 1897 | 1955 | 1857 | 1800 | 1935  | 1898.2 | 65.0   | 2025 | 1800 |
| res1.15x15_2.param4.txt  | 1919 | 1811 | 1762 | 1837 | 1824 | 1930 | 1897 | 1902 | 1834 | 1890  | 1860.6 | 51.9   | 1930 | 1762 |
| res1.15x15_3.param4.txt  | 1791 | 1939 | 1932 | 1797 | 1747 | 1836 | 1920 | 1780 | 2001 | 1960  | 1870.3 | 85.0   | 2001 | 1747 |
| res1.15x15_4.param4.txt  | 1767 | 1935 | 1818 | 1913 | 1807 | 1945 | 1771 | 1788 | 1805 | 1734  | 1828.3 | 71.3   | 1945 | 1734 |
| res1.15x15_5.param4.txt  | 1886 | 1900 | 1860 | 1992 | 1936 | 1864 | 1871 | 1853 | 1969 | 1884  | 1901.5 | 45.7   | 1992 | 1853 |
| res1.15x15_6.param4.txt  | 1903 | 1877 | 1887 | 1844 | 1920 | 1801 | 1862 | 1890 | 1775 | 1803  | 1856.2 | 46.3   | 1920 | 1775 |
| res1.15x15_7.param4.txt  | 1877 | 1833 | 1977 | 1886 | 1851 | 1820 | 1893 | 1961 | 1934 | 1856  | 1888.8 | 50.5   | 1977 | 1820 |
| res1.15x15_8.param4.txt  | 1891 | 1914 | 1906 | 1962 | 1884 | 1890 | 1863 | 1946 | 1844 | 1864  | 1896.4 | 35.1   | 1962 | 1844 |
| res1.15x15_9.param4.txt  | 1940 | 1869 | 1874 | 1963 | 1897 | 2055 | 1866 | 1947 | 1990 | 1875  | 1927.6 | 59.8   | 2055 | 1866 |
| res1.15x15_10.param4.txt | 1891 | 1950 | 1862 | 1845 | 1855 | 1778 | 1829 | 1840 | 1817 | 1938  | 1860.5 | 50.3   | 1950 | 1778 |

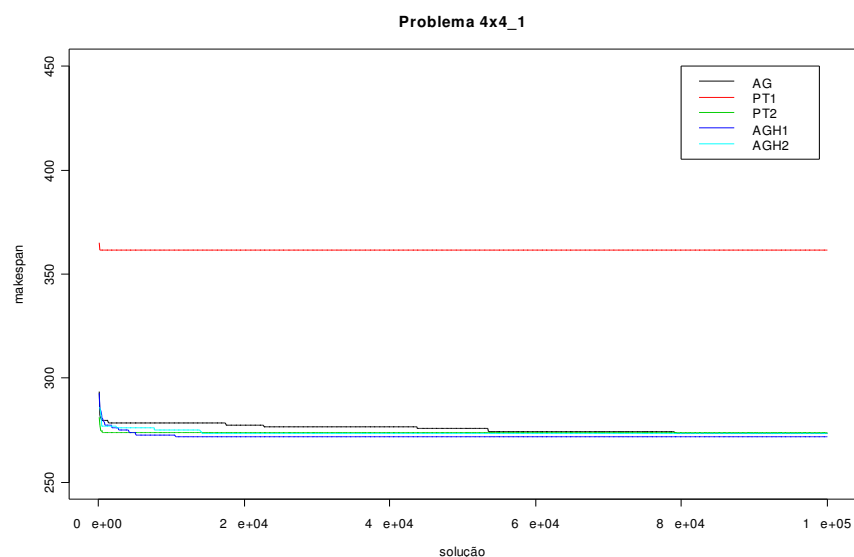
## ***Anexo 9 – Resultados médios da evolução da procura***

Estes foram os resultados obtidos pelos 5 algoritmos, para os 53 problemas testados, quando usado o quarto conjunto de parâmetros (populações de 100 indivíduos com probabilidade de mutação de 0.1)

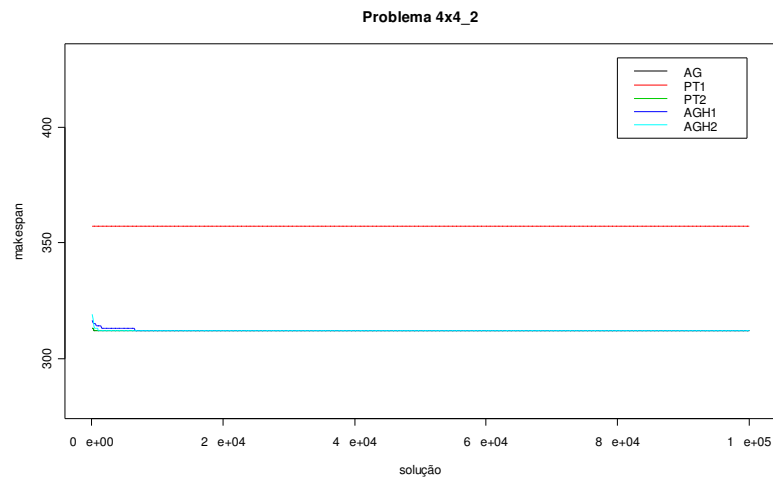
- Resultados médios da evolução da procura para o problema 3x3:



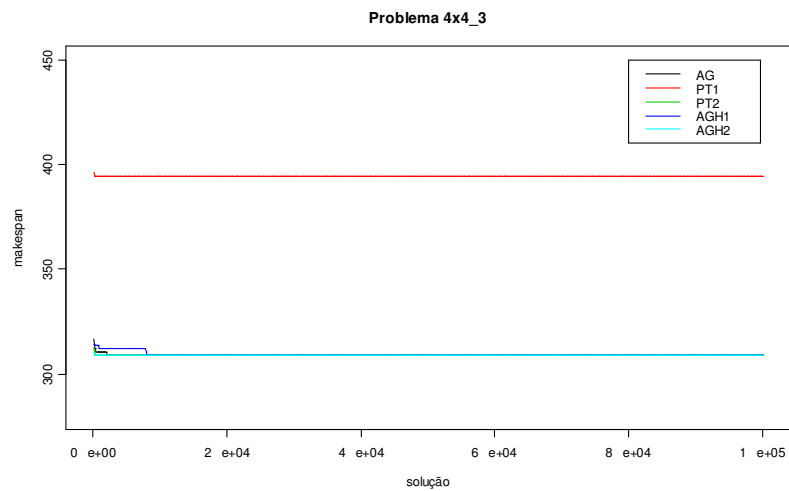
- Resultados médios da evolução da procura para o problema 4x4\_1:



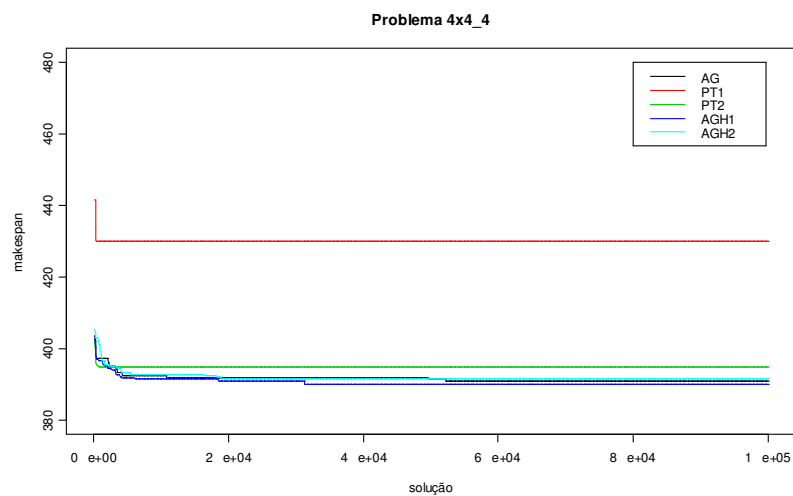
- Resultados médios da evolução da procura para o problema 4x4\_2:



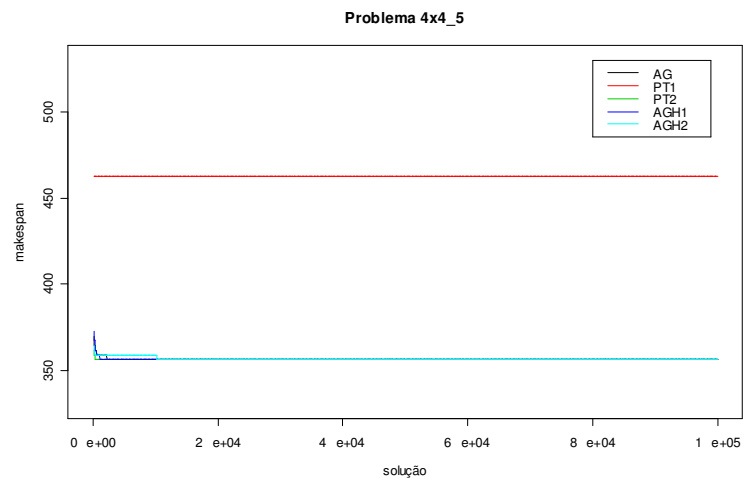
- Resultados médios da evolução da procura para o problema 4x4\_3:



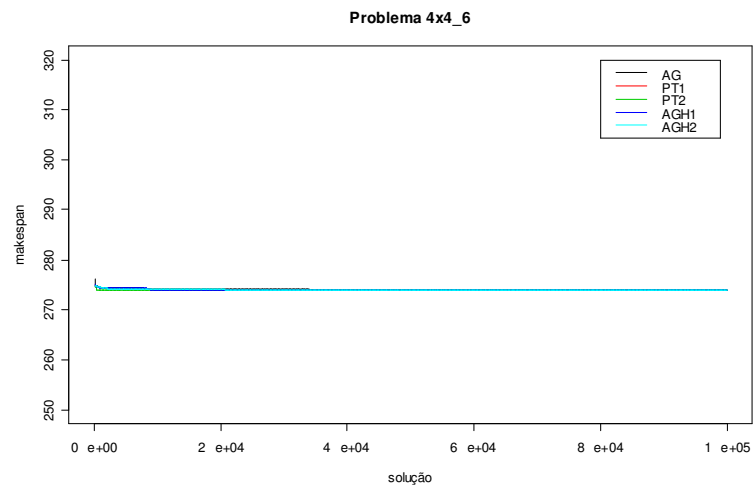
- Resultados médios da evolução da procura para o problema 4x4\_4:



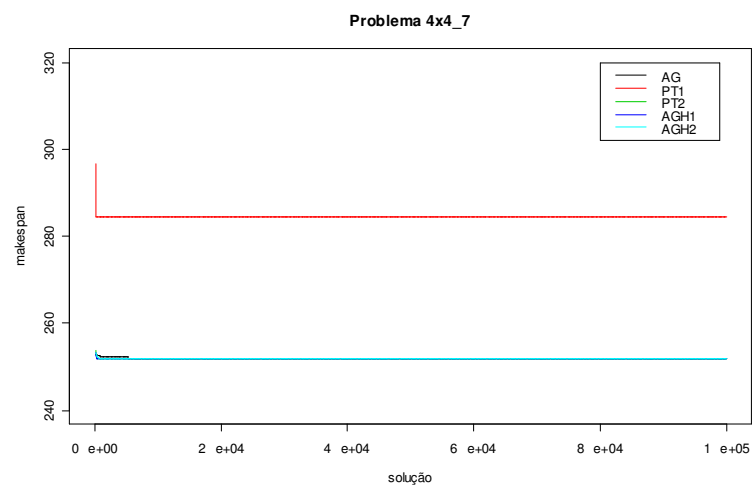
- Resultados médios da evolução da procura para o problema 4x4\_5:



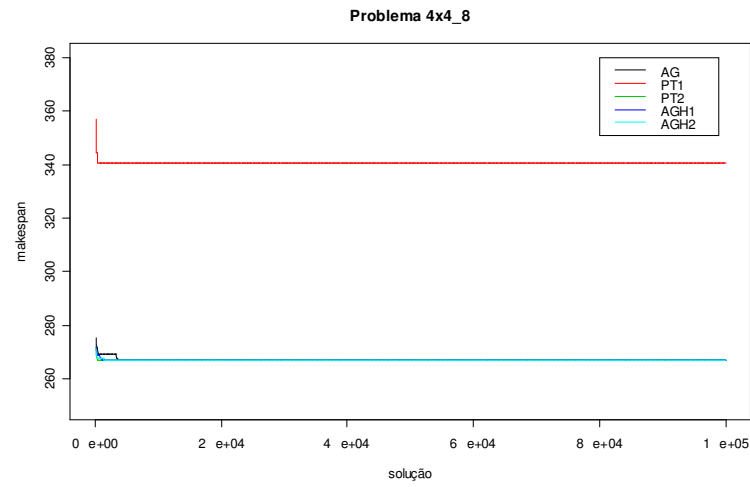
- Resultados médios da evolução da procura para o problema 4x4\_6:



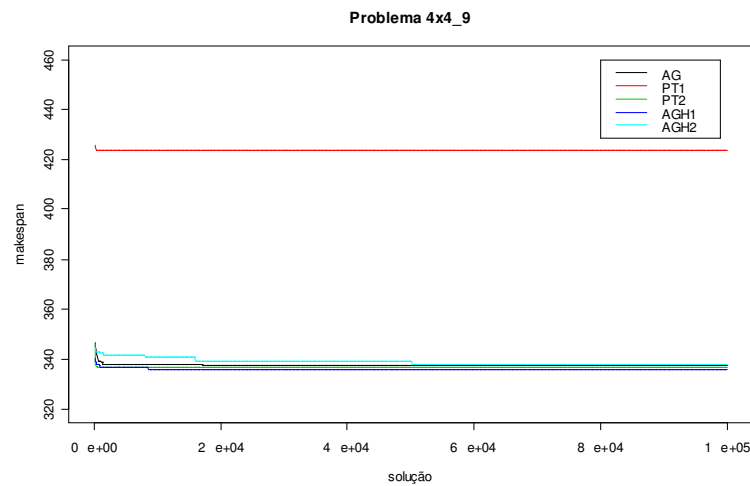
- Resultados médios da evolução da procura para o problema 4x4\_7:



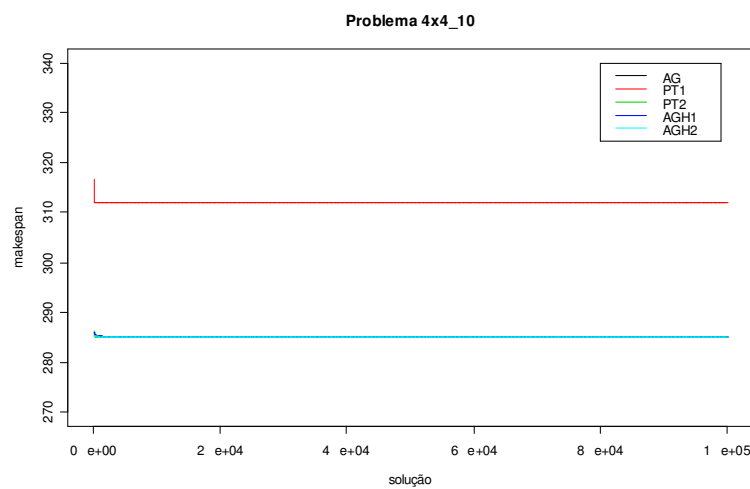
- Resultados médios da evolução da procura para o problema 4x4\_8:



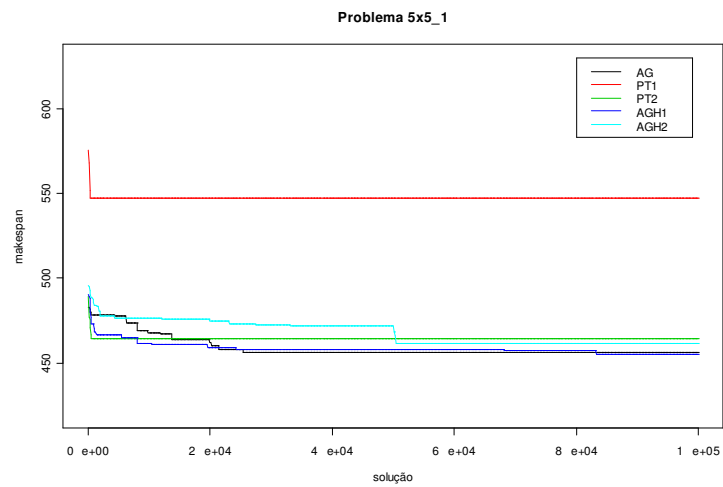
- Resultados médios da evolução da procura para o problema 4x4\_9:



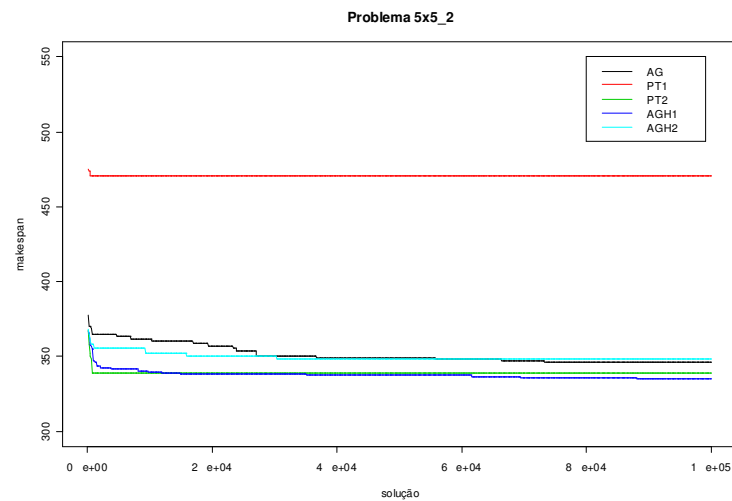
- Resultados médios da evolução da procura para o problema 4x4\_10:



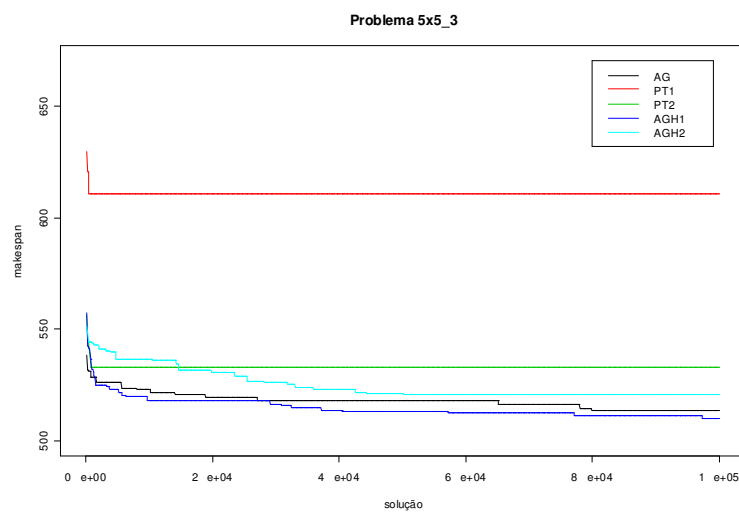
- Resultados médios da evolução da procura para o problema 5x5\_1:



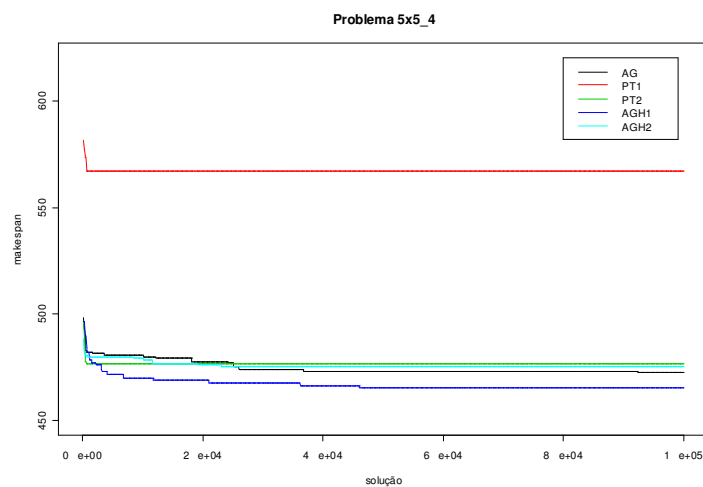
- Resultados médios da evolução da procura para o problema 5x5\_2:



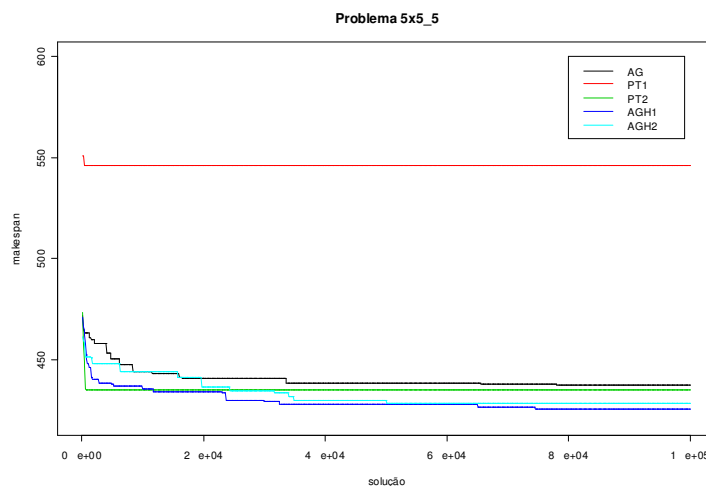
- Resultados médios da evolução da procura para o problema 5x5\_3:



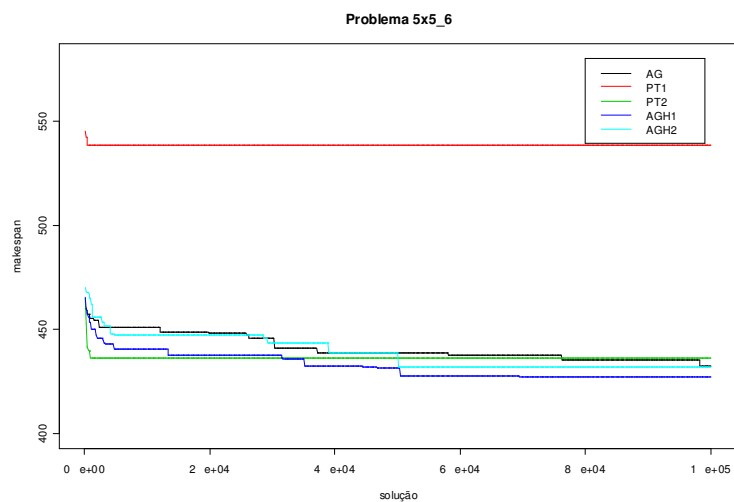
- Resultados médios da evolução da procura para o problema 5x5\_4:



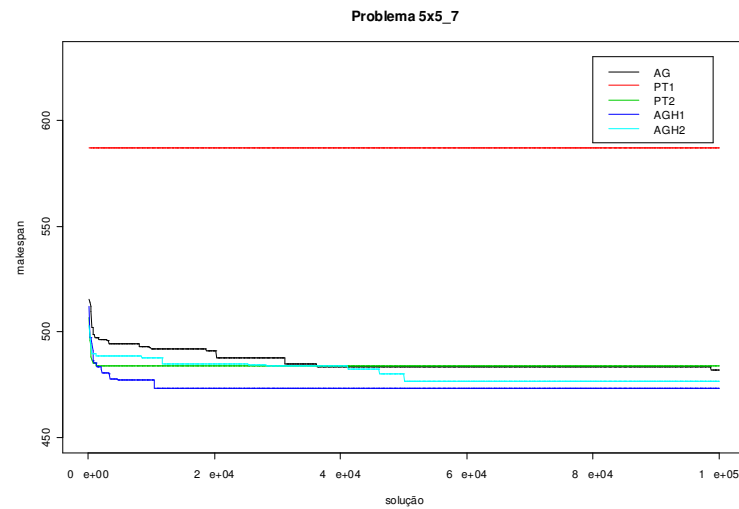
- Resultados médios da evolução da procura para o problema 5x5\_5:



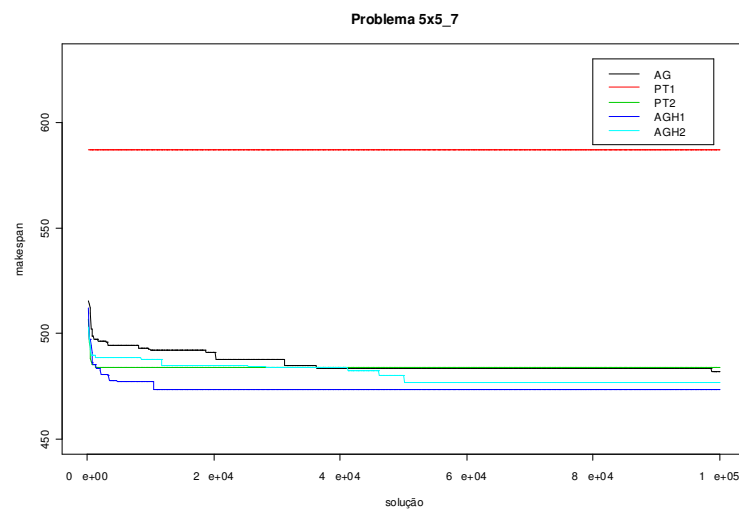
- Resultados médios da evolução da procura para o problema 5x5\_6:



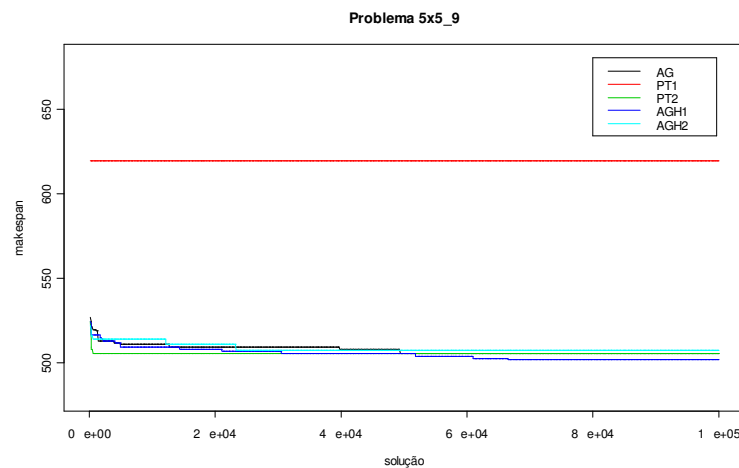
- Resultados médios da evolução da procura para o problema 5x5\_7:



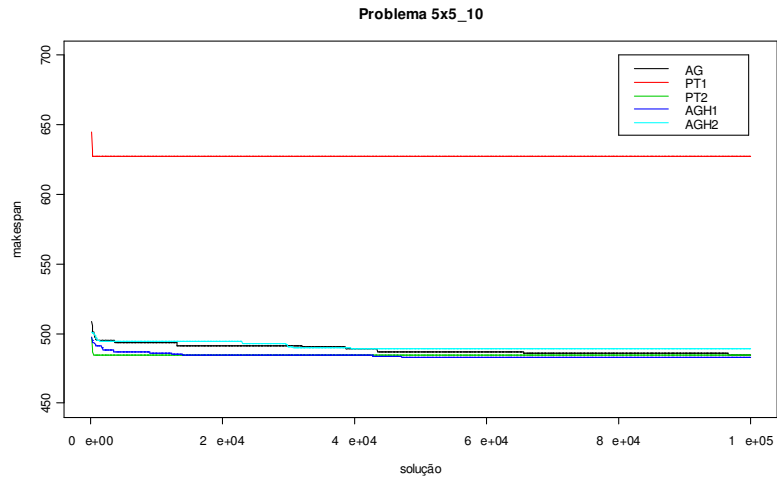
- Resultados médios da evolução da procura para o problema 5x5\_8:



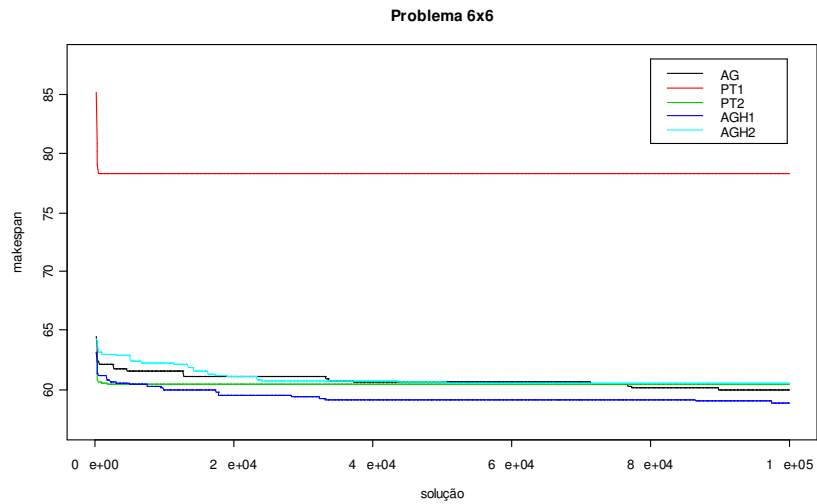
- Resultados médios da evolução da procura para o problema 5x5\_9:



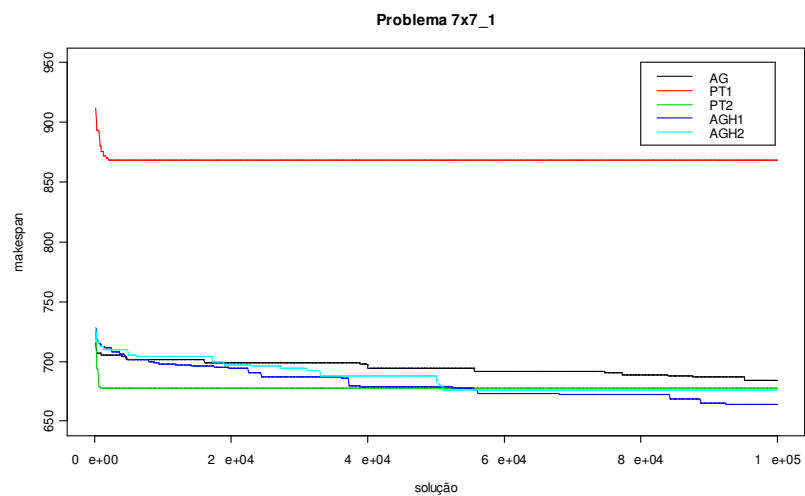
- Resultados médios da evolução da procura para o problema 5x5\_10:



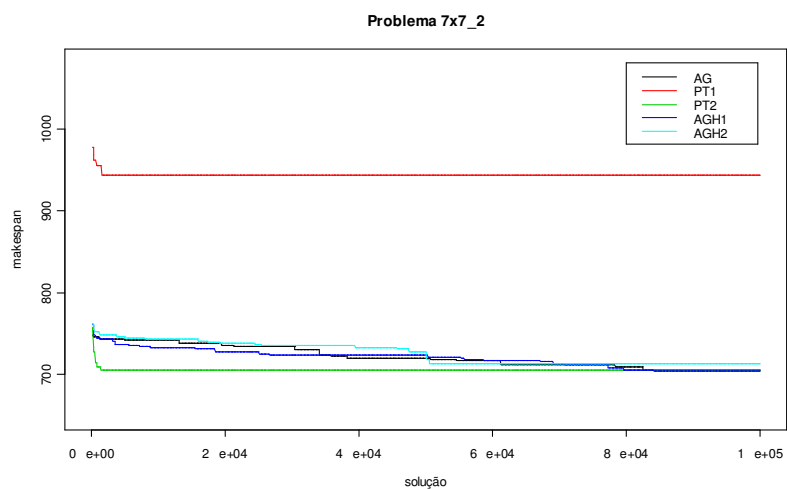
- Resultados médios da evolução da procura para o problema 6x6:



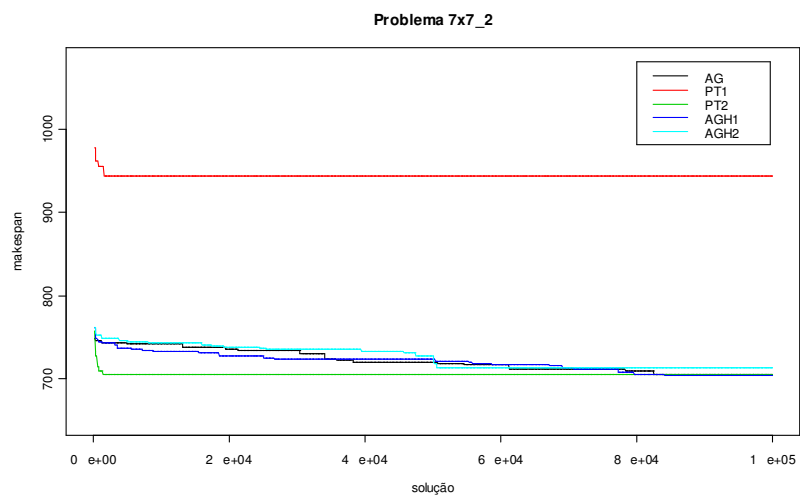
- Resultados médios da evolução da procura para o problema 7x7\_1:



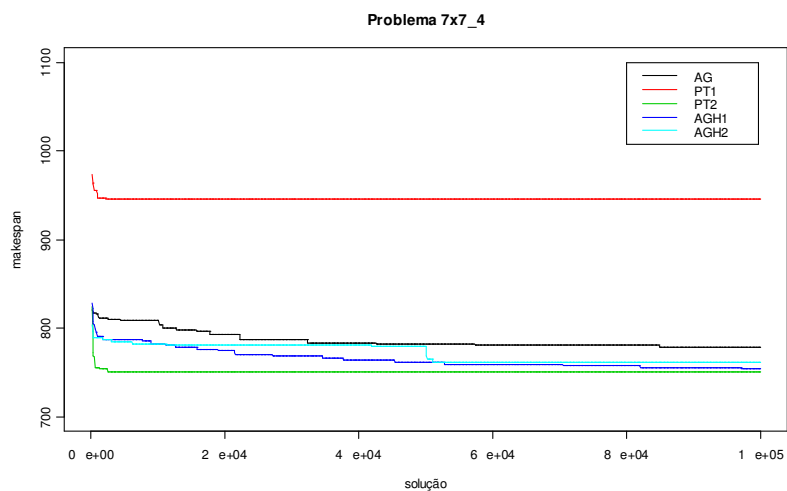
- Resultados médios da evolução da procura para o problema 7x7\_2:



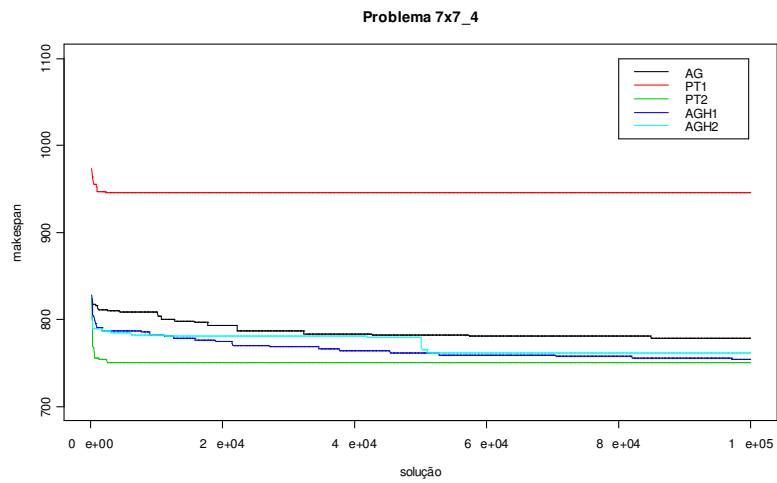
- Resultados médios da evolução da procura para o problema 7x7\_3:



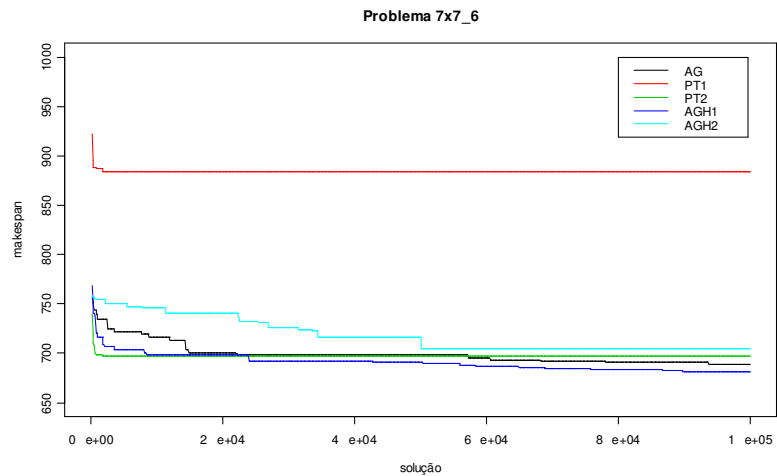
- Resultados médios da evolução da procura para o problema 7x7\_4:



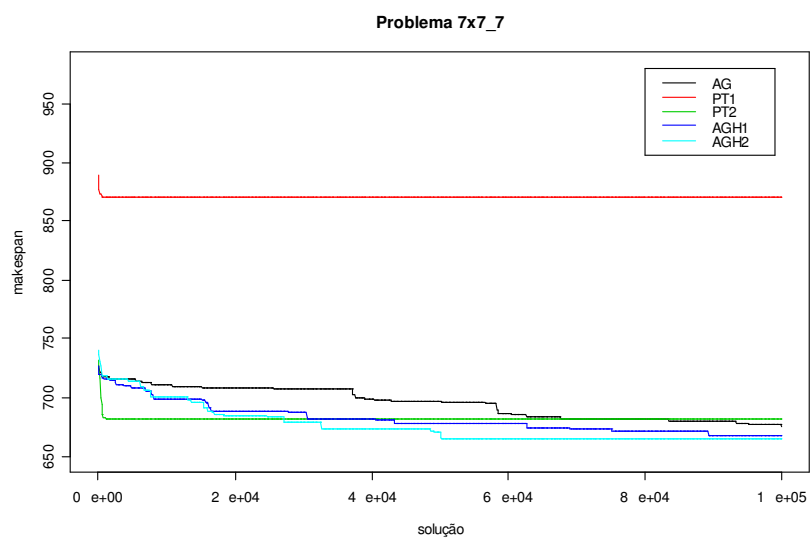
- Resultados médios da evolução da procura para o problema 7x7\_5:



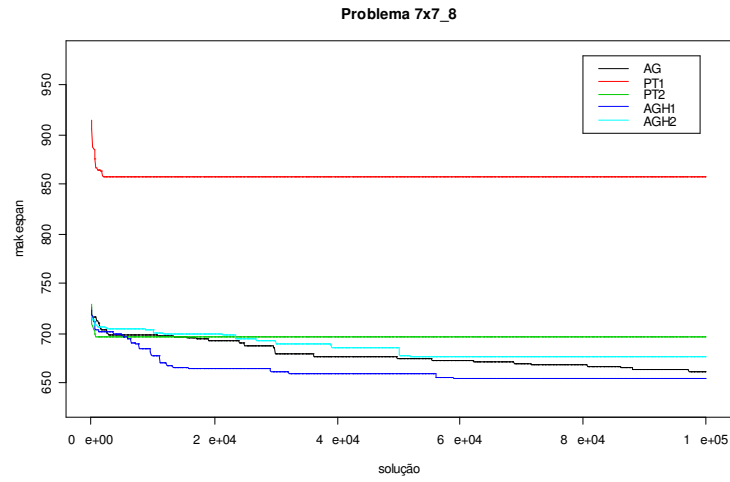
- Resultados médios da evolução da procura para o problema 7x7\_6:



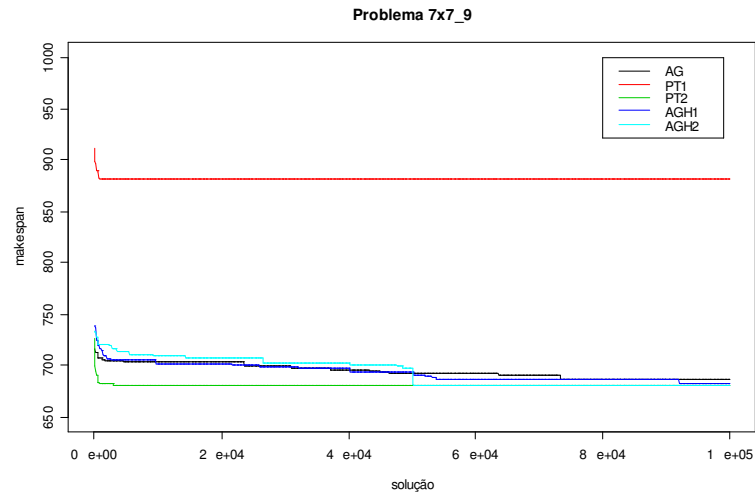
- Resultados médios da evolução da procura para o problema 7x7\_7:



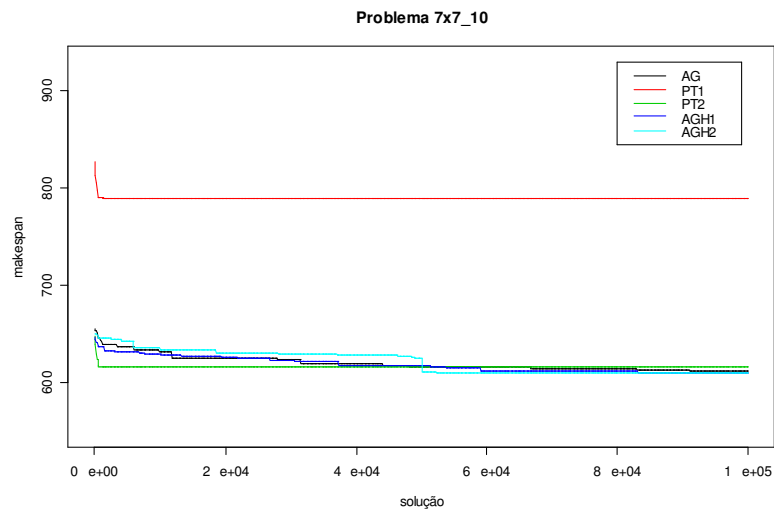
- Resultados médios da evolução da procura para o problema 7x7\_8:



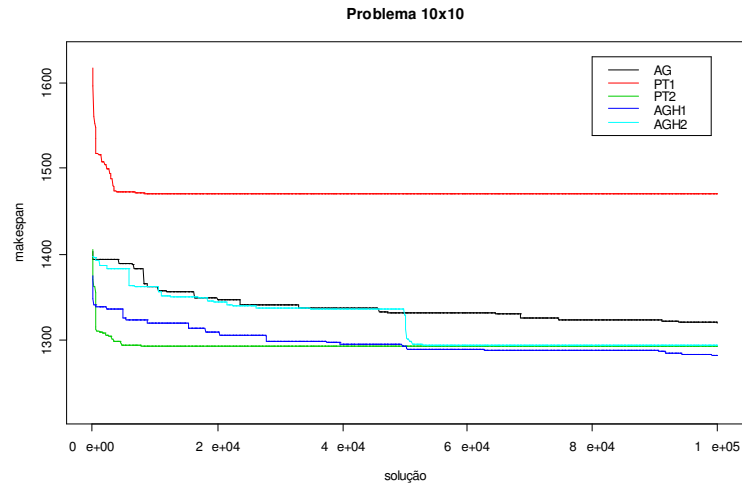
- Resultados médios da evolução da procura para o problema 7x7\_9:



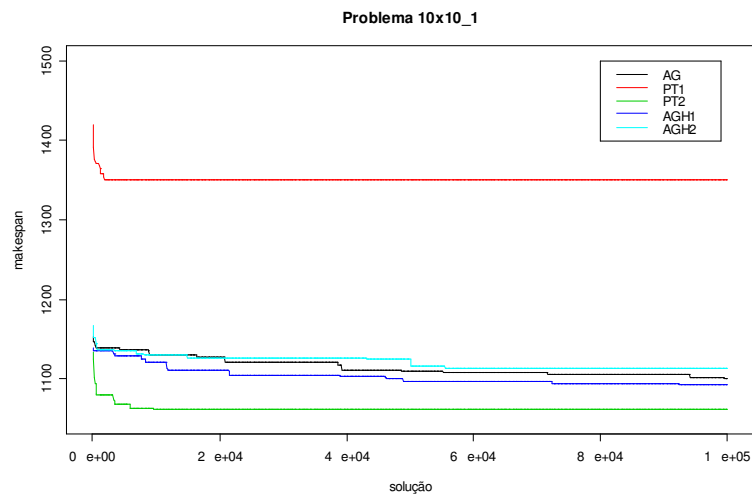
- Resultados médios da evolução da procura para o problema 7x7\_10:



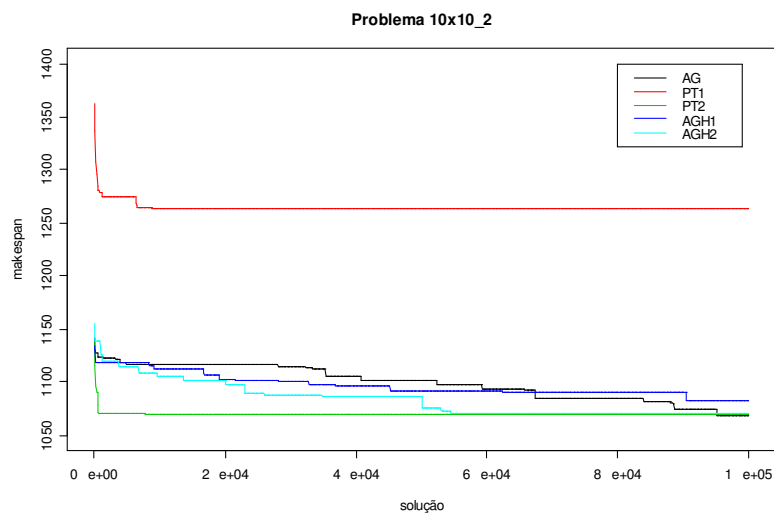
- Resultados médios da evolução da procura para o problema 10x10:



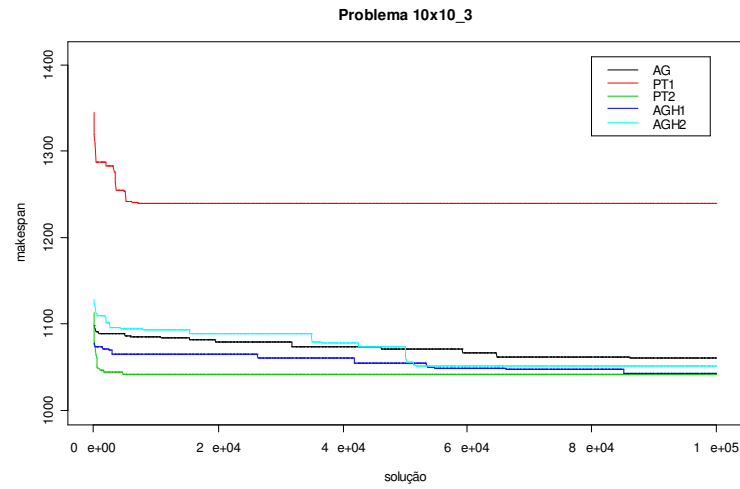
- Resultados médios da evolução da procura para o problema 10x10\_1:



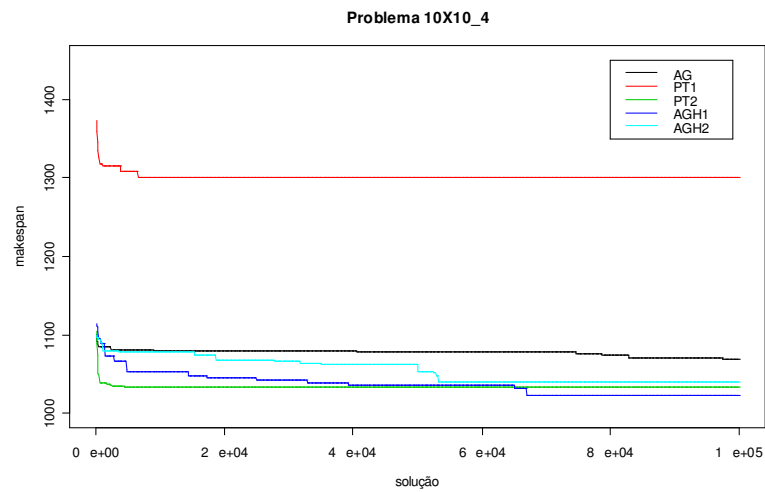
- Resultados médios da evolução da procura para o problema 10x10\_2:



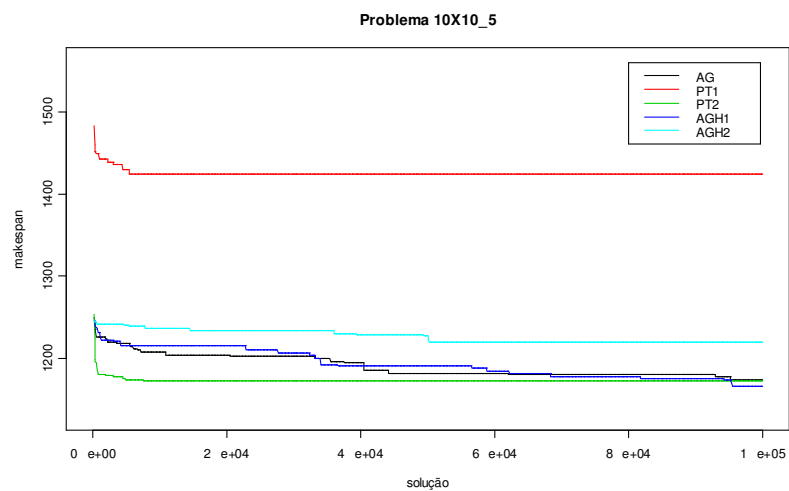
- Resultados médios da evolução da procura para o problema 10x10\_3:



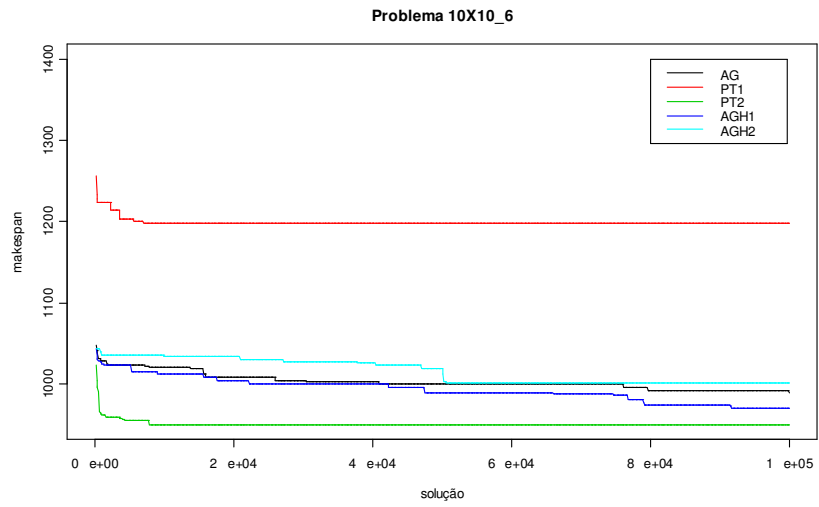
- Resultados médios da evolução da procura para o problema 10x10\_4:



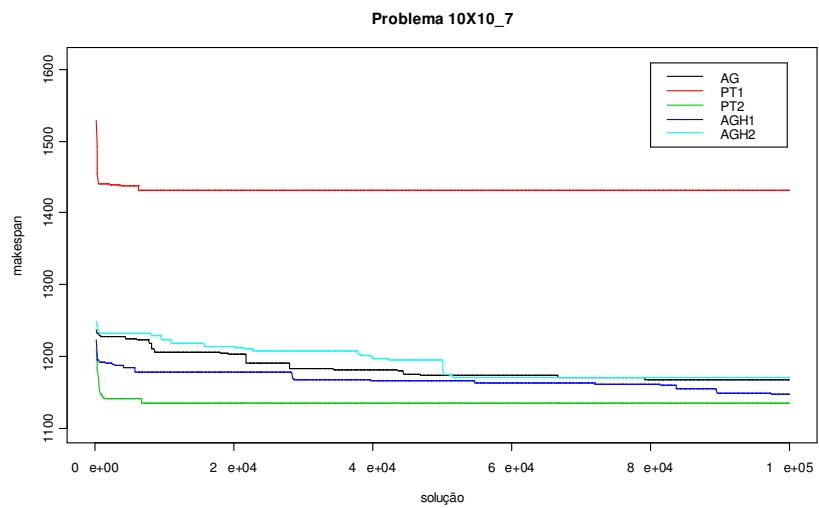
- Resultados médios da evolução da procura para o problema 10x10\_5:



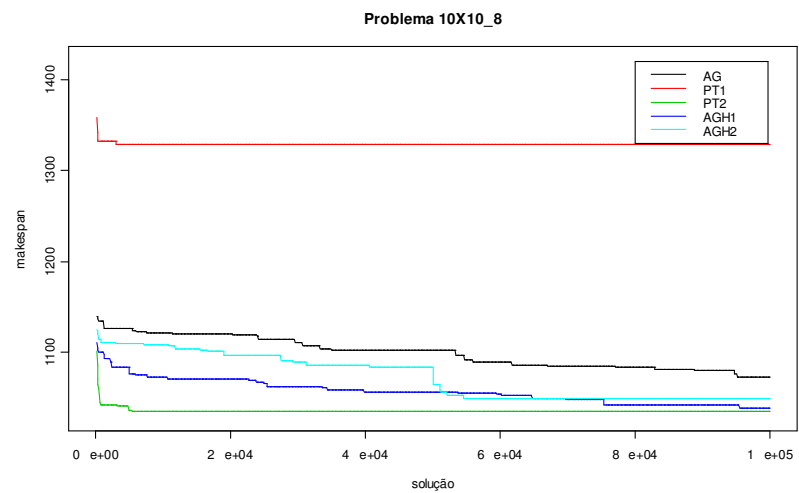
- Resultados médios da evolução da procura para o problema 10x10\_6:



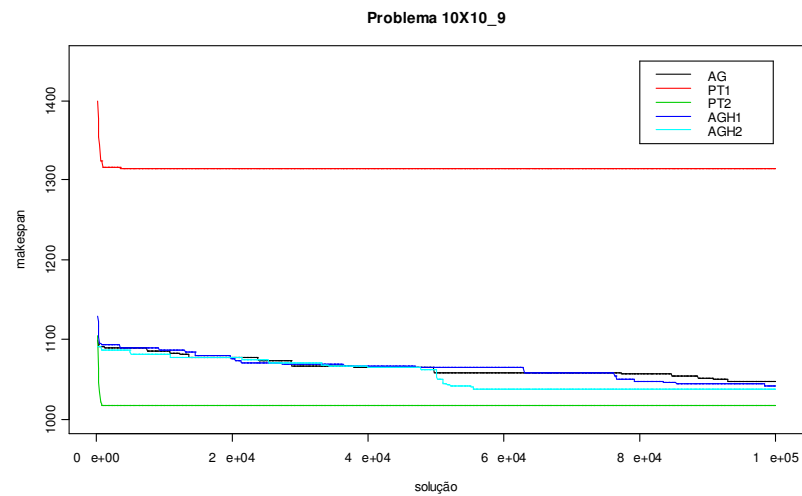
- Resultados médios da evolução da procura para o problema 10x10\_7:



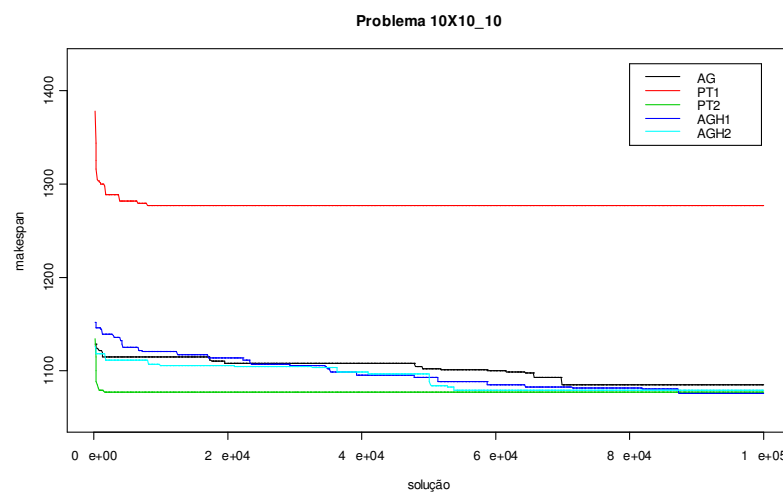
- Resultados médios da evolução da procura para o problema 10x10\_8:



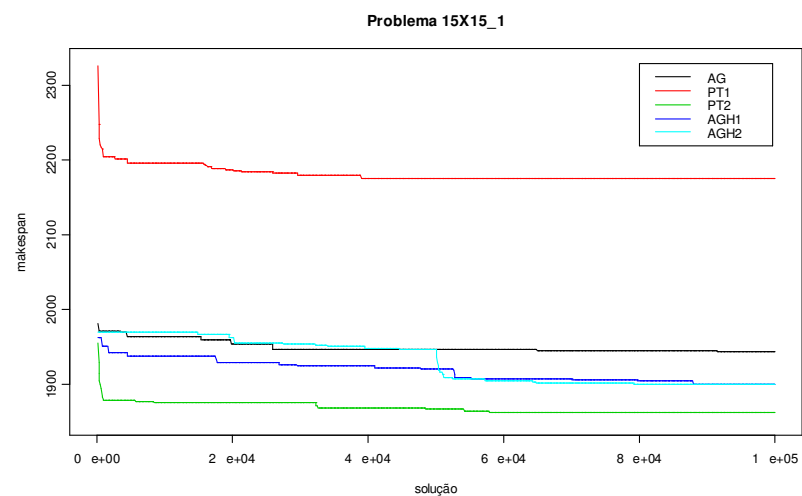
- Resultados médios da evolução da procura para o problema 10x10\_9:



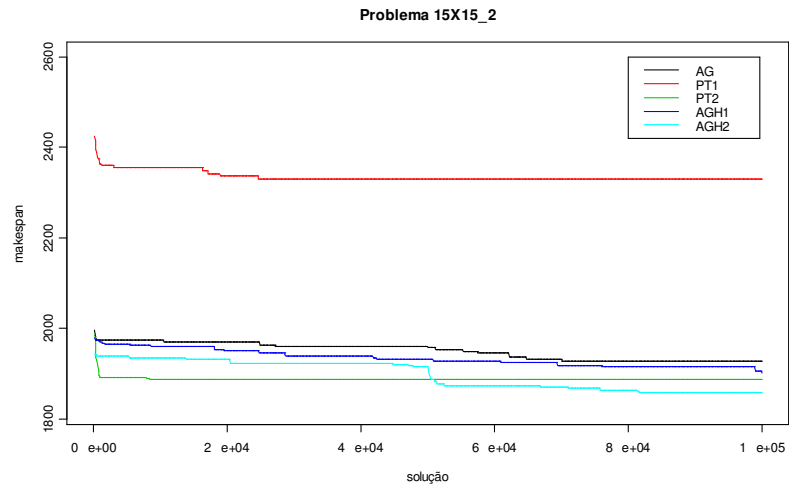
- Resultados médios da evolução da procura para o problema 10x10\_10:



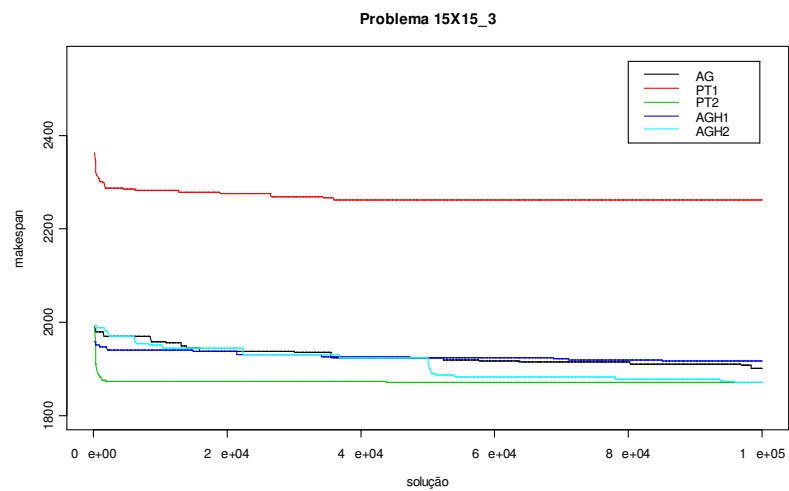
- Resultados médios da evolução da procura para o problema 15x15\_1:



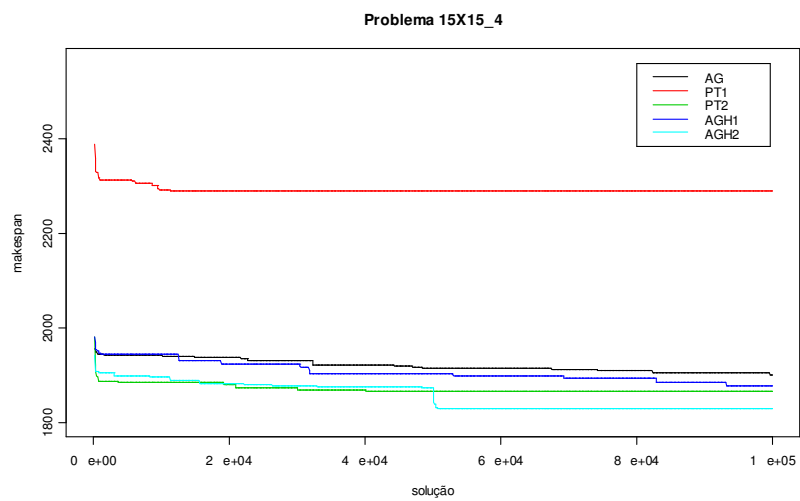
- Resultados médios da evolução da procura para o problema 15x15\_2:



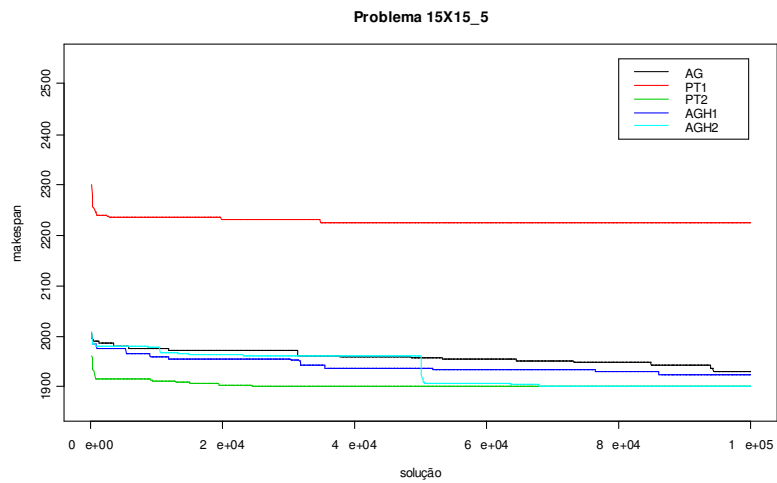
- Resultados médios da evolução da procura para o problema 15x15\_3:



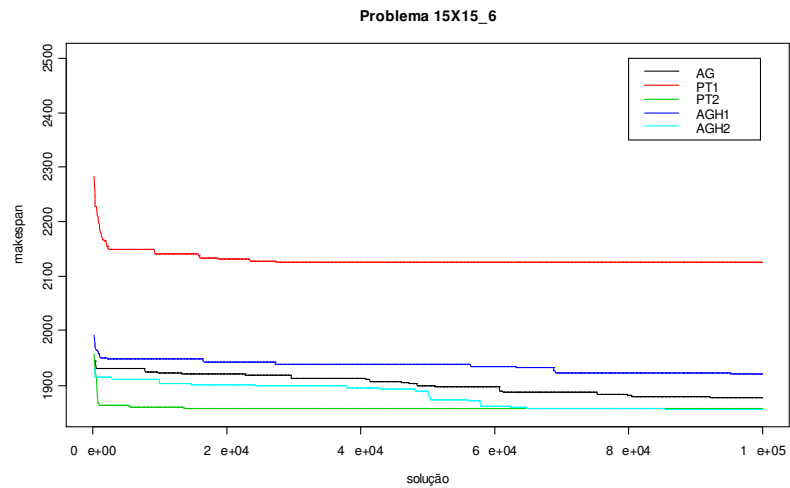
- Resultados médios da evolução da procura para o problema 15x15\_4:



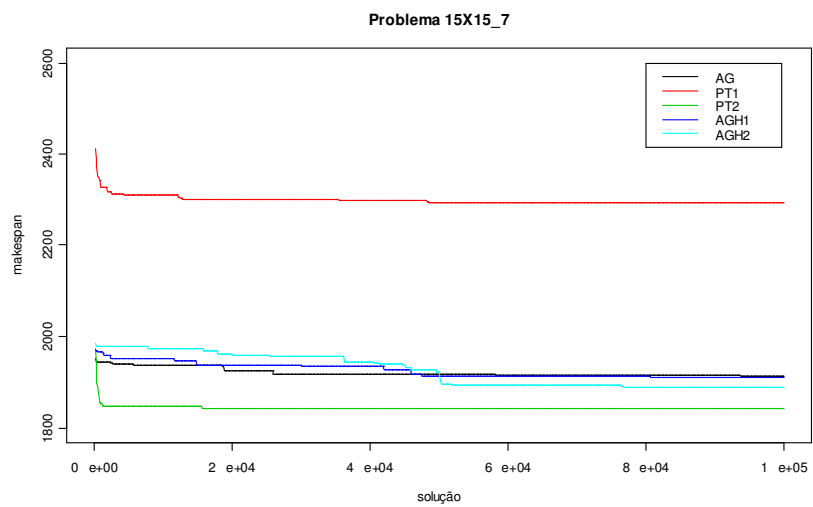
- Resultados médios da evolução da procura para o problema 15x15\_5:



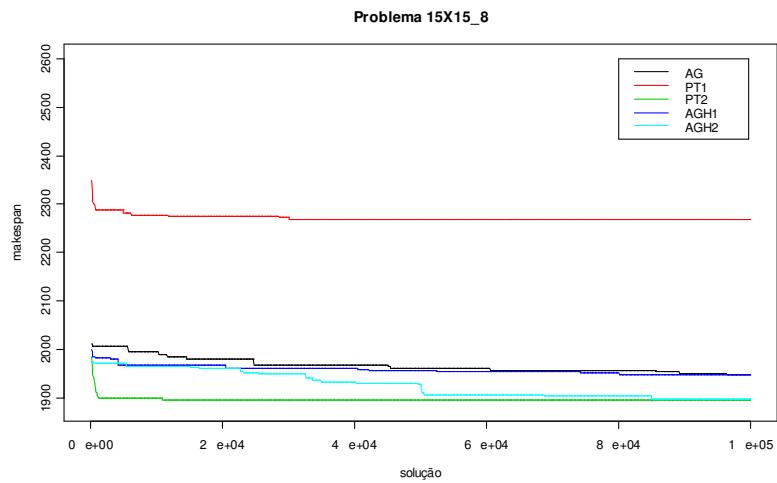
- Resultados médios da evolução da procura para o problema 15x15\_6:



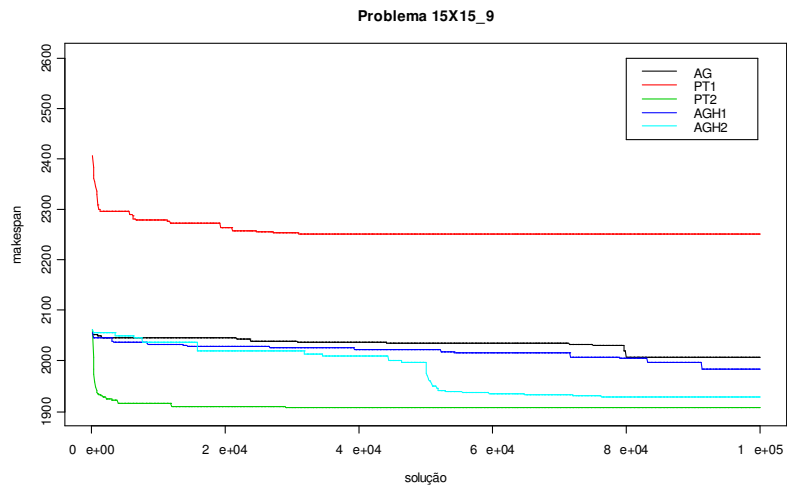
- Resultados médios da evolução da procura para o problema 15x15\_7:



- Resultados médios da evolução da procura para o problema 15x15\_8:



- Resultados médios da evolução da procura para o problema 15x15\_9:



- Resultados médios da evolução da procura para o problema 15x15\_10:

