UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO - FEEC
DEPARTAMENTO DE ENGENHARIA DE SISTEMAS - DENSIS

ALGORITMOS MEMÉTICOS APLICADOS AOS PROBLEMAS DE SEQUENCIAMENTO EM MÁQUINAS

Alexandre de Sousa Mendes
Orientador: Prof. Dr. Paulo Morelato França

Banca Examinadora:

Prof. Dr. Paulo Morelato França - FEEC/UNICAMP
Dr. Luiz Antonio Nogueira Lorena - INPE/São José dos Campos
Prof. Dr. Fernando José Von Zuben - FEEC/UNICAMP

Tese de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica
Área de Concentração: Automação

Tese de Mestrado
Campinas - SP - Brasil
Agosto de 1999

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

M522a

Mendes, Alexandre de Sousa
Algoritmos meméticos aplicados aos problemas de sequenciamento em máquinas / Alexandre de Sousa Mendes.--Campinas, SP: [s.n.], 1999.

Orientador: Paulo Morelato França.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Algoritmos genéticos. 2. Engenharia de produção. 3. Heurística. I. França, Paulo Morelato. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

RESUMO

O problema de Sequenciamento em Máquina Simples (SMS) é um dos mais tradicionais na área de sequenciamento. Neste trabalho é explorado inicialmente o problema de SMS com restrições de tempo (datas de entrega de produtos e tempos de preparação). O objetivo é a minimização do atraso total, que se caracteriza pela soma dos atrasos na entrega de todos os produtos.

O método escolhido é baseado em Algoritmos Meméticos (AM). AM constituem uma classe de metaheurística do tipo populacional que engloba outras já conhecidas, como Algoritmos Genéticos híbridos, Busca por Espalhamento, entre outras. Nesta tese, o AM utilizado é um Algoritmo Genético (AG) acrescido de uma rotina de busca local aplicada a cada elemento novo da população. Na parte evolutiva são estudadas e testadas várias possibilidades para os operadores de recombinação, mutação, estruturas populacionais, etc. São também pesquisadas estruturas de busca local que melhor se adequam para a formação de um AM.

As comparações de desempenho são feitas com três diferentes abordagens encontradas na literatura.

Como complementação ao trabalho são ainda analisadas a robustez do AM e o *fitness landscape*. Ambos são de extrema importância para validar o método e para caracterizar a dificuldade em resolver diferentes instâncias do problema.

Como extensão são mostrados alguns resultados para o problema de Sequenciamento em Máquinas Paralelas (SMP) com tempos de preparação dependentes da sequência e minimização do *makespan*. O algoritmo utilizado é o mesmo do problema de SMS, porém com modificações em algumas de suas características, necessárias para tratar o novo problema. A comparação de desempenho é feita com um algoritmo do tipo Busca Tabu (BT).

ABSTRACT

The Single Machine Scheduling Problem (SMS) is one of the most representative in the scheduling area. In this work we initially explore the SMS problem with time constraints (due-dates and setup times). The goal is to minimize the total tardiness, which is characterized by the sum of the delays in the production of all products.

The method chosen is based on Memetic Algorithms (MA). MA constitute a class of population metaheuristics that comprise many others, such as Hybrid Genetic Algorithms, Scatter Search, etc. In this thesis the MA implemented is a Genetic Algorithm (GA) with a local search routine that is applied to each new element of the population. Concerning the evolutionary part we study and test several possibilities of recombination operators, mutation, population structures, etc. We also test local search structures that are better fitted to the design of an MA.

Performance comparisons are carried out with three different approaches found in the literature.

As an addition to this work we also analyze the robustness of the MA and the Fitness Landscape. Both are extremely important to validate the method and to characterize the difficulty to solve different instances of the problem.

As an extension we show some results for the Parallel Machine Scheduling (PMS) problem with sequence-dependent setup times. The goal is the minimization of the makespan. The MA applied is the same of the SMS problem with some changes in its characteristics, necessary to deal with the new problem. Performance comparisons are carried out with a Tabu Search algorithm.

DEDICATÓRIA

Dedico este trabalho à minha família, em especial à minha mãe **Lourdes** e meu pai **Walcler**, que sempre estiveram presentes em todos os momentos importantes da minha vida. Principais responsáveis por minha formação de vida, devo a eles tudo que realizei até hoje. Seus esforços para me fornecer muito mais do que o destino a eles concedeu, torna infinito meu débito e meu agradecimento. Dedico este trabalho também ao meu irmão, **Walcler Jr.**, de cuja amizade, verdadeira e desinteressada, tive o privilégio de desfrutar ao longo desses anos. Seu apoio e sua importância na minha vida são também imensuráveis.

AGRADECIMENTOS

Inicialmente ao professor **Paulo França** que, desde o início, depositou toda a confiança em mim e que sempre me apoiou durante a execução deste trabalho.

A todos os membros do projeto MEMEPOOL, cujas idéias permeiam cada capítulo dessa tese. Em especial agradeço aos comentários e sugestões de **Pablo Moscato**, **Luciana Buriol**, **Felipe Müller** e **Andrea Toniolo**.

À **Natasa Dobrasinovic**, que implementou grande parte do Capítulo V desta tese.

A todos os professores e colegas de trabalho que direta ou indiretamente contribuíram para a realização deste trabalho.

Ao **CNPq** – Conselho Nacional de Desenvolvimento Científico e Tecnológico – cujo apoio financeiro tornou possível a conclusão desta tese.

ÍNDICE

CAPÍTULO I

Introdução ao problema de Sequenciamento em Máquina Simples

Sob a denominação genérica de Sequenciamento em Máquina Simples, reconhecemos uma das classes de problemas na área de sequenciamento, como mostram os trabalhos de Graham *et al.* [13] e Graves [14]. Uma das revisões mais recentes nessa área, especificamente tratando do problema de minimização do atraso total, foi realizada por Koulamas [18].

Os principais elementos que caracterizam um problema de sequenciamento determinístico (não-estocástico) são a configuração da(s) máquina(s), das tarefas e a função objetivo. A configuração das máquinas serve para classificar o problema como sendo de máquina simples ou múltiplas máquinas. No caso de múltiplas máquinas, o problema pode envolver máquinas paralelas idênticas ou não, assim como arranjos em série como nos problemas de *flowshop*.

Outras revisões menos recentes são: Gupta e Kyparisis [15], Cheng e Sin [6], Kawaguchi e Kyan [17], Sen e Gupta [29] e Cheng e Gupta [5].

Como exemplo de um dos problemas de SMS mais simples, citamos o que envolve apenas tempos de processamento das operações e datas de entrega dos produtos, com o objetivo de minimizar o atraso total. Este problema pode ser classificado como 1//T, de acordo com o esquema de classificação proposto por Graham *et al.* [13] e foi provado pertencer à classe NP-Completo, em 1990 por Du e Leung [7]. Utilizando-se relações de

dominância de soluções e técnicas de decomposição, pode-se chegar à solução ótima para problemas de médio e grande portes, pois o espeço de busca se reduz consideravelmente.

Este problema pode ser estendido com a inclusão de tempos de preparação de tarefas, relações de precedência entre as operações, *ready-times*, penalidades dependentes da tarefa, penalidades por adiantamento, etc. Do ponto de vista da função objetivo, podemos citar a minimização do *makespan*, atraso máximo, número de tarefas atrasadas, atraso médio, *flowtime*, ou ainda uma combinação deles, o que daria origem a um problema multi-critério.

O problema abordado neste trabalho considera para cada tarefa o seu tempo de processamento e sua data de entrega. Ele ainda envolve tempos de preparação da máquina quando se muda a tarefa a ser processada, sendo esses tempos dependentes da sequência das tarefas. Quando incluímos esse último, perdemos as relações de dominância de soluções do problema original e assim torna-se mais difícil atingir a solução ótima.

A medida de desempenho escolhida, o atraso total, é considerada uma medida *regular*. Isto significa que um sequenciamento ótimo não pode ter tempos ociosos entre operações, reduzindo a complexidade do problema. Assim, qualquer solução válida pode ser vista como uma permutação das tarefas. Usando a representação do espaço de permutações, o espaço de soluções possui $n!$ configurações possíveis.

O problema de SMS com tempos de preparação de operações está presente em muitos sistemas de manufatura industriais. Indústrias químicas, por exemplo, são um exemplo clássico onde tempos de preparação são fundamentais para a determinação do melhor sequenciamento. Tome o caso de uma linha de produção onde a unidade envasadora receba o líquido a ser envasado e o insira em recipientes. Supondo que o próximo produto a ser envasado seja transparente e que o líquido anteriormente envasado fosse escuro e relativamente viscoso, pode-se supor que o tempo de preparação neste caso deva ser considerável. De fato, em situações reais pode-se chegar a casos em que os tempos de preparação são até 10 vezes maiores que os tempos de processamento em si. Outros casos clássicos podem ser encontrados em Ragatz [25].

A partir do exemplo anterior fica claro que o tempo de preparação de uma tarefa depende essencialmente da tarefa imediatamente anterior. Em situações reais esse tempo

pode considerar mudança do ferramental da máquina, limpeza, transporte de peças (se as tarefas forem realizadas em peças diferentes), etc.

O problema de SMS abordado neste trabalho é classificado como $1/sds/T_{total}$, sendo descrito como:

Entrada: Seja n o número de tarefas a serem processadas em uma máquina.
Seja p um vetor $\{p_1, ..., p_n\}$ de tempos de processamento das tarefas.
Seja d um vetor $\{d_1, ..., d_n\}$ de datas de entrega das tarefas.
Seja $\{s_{ij}\}$ uma matriz assimétrica de tempos de preparação onde o elemento s_{ij} é o tempo necessário para preparar a produção da tarefa j depois da máquina terminar o processamento da tarefa i.

Objetivo: Encontrar a permutação que minimize o atraso total do sequenciamento, que é calculado como:

$$T = \sum_{k=1}^{n} \max[0, c_k - d_k] \qquad (1)$$

onde c_k é o instante de término de processamento da tarefa k. Apresentamos a seguir uma possível solução para um problema de cinco tarefas (1-4-3-2-5), bem como a representação gráfica do atraso total.

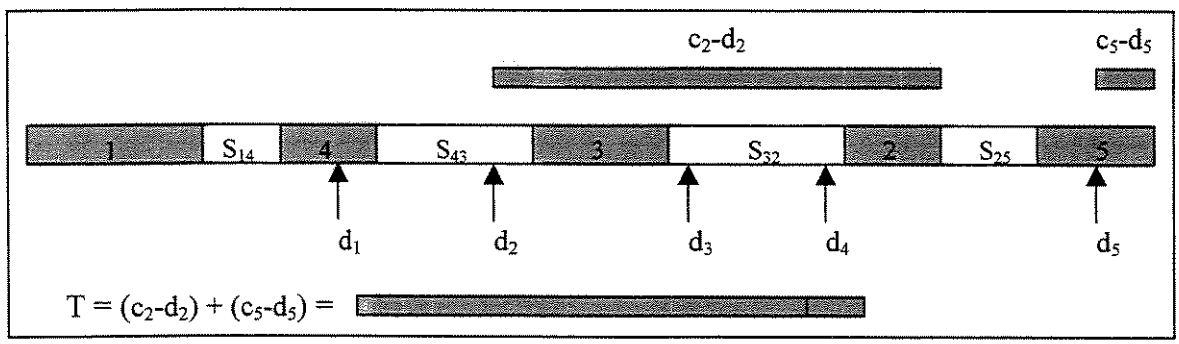


Figura 1: Gráfico de Gantt de uma solução para o problema de SMS.

Apesar de sua clara aplicação em situações reais, o problema abordado nesta tese só começou a receber mais atenção da comunidade científica recentemente. Entre trabalhos importantes que tratam especificamente deste problema podemos citar o de Raman *et al.* [26], onde é proposta a regra de despacho *Apparent Tardiness Cost* (ATC) modificada para incluir tempos de preparação. Em seguida, Ragatz [25] propôs um algoritmo de *Branch and Bound* (B&B). No entanto apenas em instâncias pequenas (até 45 tarefas) chegou-se à solução ótima. Em 1995 tivemos o trabalho de Rubin e Ragatz [28], onde é proposta uma abordagem populacional baseada em AG para este problema. É também apresentado um novo operador de recombinação. Mais recentemente, em 1997, Lee *et al.* [19] propuseram uma extensão da heurística ATC-modificada de Raman, denominada *Apparent Tardiness Cost with Setups* (ATCS). Resultados indicam uma melhoria de desempenho da ordem de 30% sobre a regra anterior. Finalmente, também em 1997, Tan and Narasimhan [30] aplicaram um algoritmo de *simulated annealing* (SA) ao problema e compararam os resultados com uma estratégia tipo *multiple start* (MS). A conclusão a que chegaram é de que o SA superou o MS em todas menos três instâncias com porcentagens de melhoria não superiores a 6%.

CAPÍTULO II

Algoritmos comparativos

Neste trabalho, utilizamos quatro algoritmos comparativos para validação da abordagem memética proposta na tese. São eles: MS, EDD com inserção (EDD_{ins}), ATCS e AG. Neste capítulo discutiremos os aspectos principais dos três primeiros. O AG será discutido mais à frente, juntamente com o AM, devido às semelhanças entre ambos.

2.1 Multiple Start (MS)

O procedimento de MS consiste na geração aleatória de soluções iniciais seguida da aplicação de uma rotina de busca local de modo a fazê-las convergirem a mínimos locais. Rotinas de busca local se baseiam em uma definição de vizinhança que estabelece uma relação entre soluções no grafo de espaço de estados. Pode-se definir a vizinhança de uma solução como o conjunto de soluções que podem ser geradas executando-se qualquer movimento ou conjunto de movimentos possíveis, permitidos pela política da própria vizinhança. Por exemplo, para o MS foi utilizada uma vizinhança baseada na troca de tarefas, chamada *all-pairs*. Ela consiste em trocar todos os pares de tarefas de uma dada solução entre si. Um algoritmo de busca local baseado nessa vizinhança é descrito a seguir.

Passo 1: Obtenha uma solução inicial factível.

Passo 2: Faça uma varredura sequencial avaliando a troca de posição de todos os pares de tarefas da solução inicial. Confirme cada troca que melhore o atraso total. Trocas que pioram o atraso total devem ser ignoradas.

Passo 3: Se no Passo 2 todas as possíveis trocas foram avaliadas e nenhuma melhorou a solução inicial, PARE. A solução é um mínimo local da vizinhança. Caso contrário, volte ao Passo 2, agora com a solução inicial atualizada.

A figura a seguir ilustra a idéia do MS.

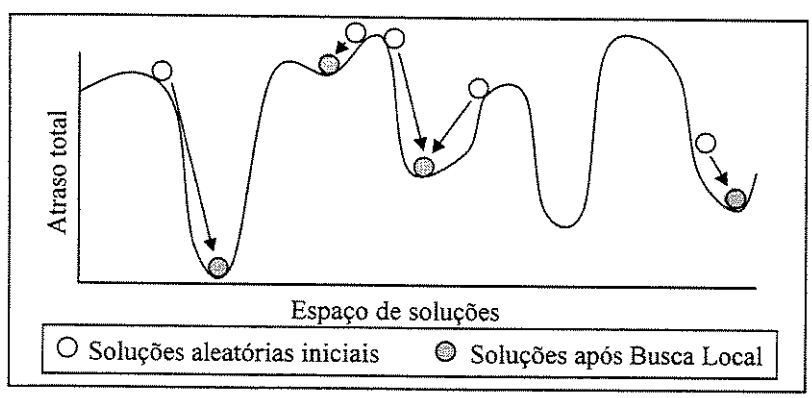


Figura 2: Ilustração do conceito de busca local no espaço de soluções.

No caso do MS, a solução inicial do passo 1 é gerada de forma aleatória e está representada pelos círculos brancos da Figura 2. Após a busca local a solução resultante é um mínimo local, representado pelos círculos em cinza. Há várias soluções caracterizadas na figura pois o procedimento é repetido diversas vezes, com soluções aleatórias iniciais diferentes de forma a recobrir mais eficientemente o espaço de soluções.

Na seção 3.11 tornaremos a abordar o tema de buscas locais, uma vez que o AM implementado se utiliza intensamente delas.

2.2 EDD com inserção

A regra *Earliest Due Date* (EDD) é uma das mais utilizadas quando o problema de sequenciamento envolve datas de entrega de produtos. O problema é que esta regra leva em conta apenas as datas de entrega, ordenando as tarefas segundo uma sequência não-decrescente das mesmas. Para o problema $1/sds/T_{total}$ essa regra tem um desempenho muito ruim, pois os tempos de preparação têm um papel importante no sequenciamento e não são levados em conta pela EDD.

Desta forma optamos por uma regra modificada que chamamos de EDD com inserção (EDD_{ins}), muito semelhante à regra NEH utilizada primeiramente em problemas de *flowshop* [23]. A seguir, mostramos uma descrição passo a passo desta regra.

Passo 1 (Inicializ.): Criar 2 vetores: **A** e **B**. Inicializar o vetor **A** com as tarefas na sequência EDD. Inicializar o vetor **B** como nulo.

Passo 2 (Iterações): Inserir as tarefas segundo a sequência do vetor **A** no vetor **B**, sempre na posição que produza o menor atraso total parcial – caracterizado pelo menor atraso total considerando apenas as tarefas presentes no vetor **B**.

Passo 3 (Pós-otim.): Aplicar um procedimento de melhoria (uma busca local) sobre a sequência armazenada no vetor **B** do Passo 2.

O primeiro passo se resume a ordenar as tarefas, de modo a determinar a ordem em que elas serão inseridas no vetor **B**. O segundo passo efetua a inserção em si. A primeira tarefa inserida só tem uma posição possível de entrada, pois o vetor **B** é nulo na 1ª iteração. A segunda tarefa por sua vez pode entrar em duas posições: antes ou depois da primeira tarefa. Assim escolhe-se a posição que resulte no menor atraso total, considerando-se apenas estas duas tarefas. Prosseguindo-se dessa forma, a última tarefa poderá se encaixar em n posições distintas e uma vez determinada essa posição, a sequência final estará montada e o atraso total pode ser finalmente calculado. Como se vê, é um procedimento que pode ser classificado como "guloso" e construtivo.

A sequência gerada no Passo 2 pode ainda vir a ser melhorada mediante a aplicação de um procedimento de busca local. Para tanto utilizamos duas vizinhanças de busca local: troca e inserção. A vizinhança de troca já foi rapidamente explicada na seção anterior. A vizinhança de inserção, por sua vez, consiste em retirar todas as tarefas de suas posições originais e inseri-las em todas as outras posições. Assim, no Passo 3 aplicamos sobre o vetor **B** uma busca local de troca *all-pairs* e ao fim desta uma de inserção, obtendo o vetor final.

2.3 Regra de prioridade ATCS

Conforme dito no Capítulo I, em 1997 Lee *et al.* [19] propuseram uma heurística especificamente para o problema $1/sds/T_{total}$, mas pelo fato de ainda ser muito recente, ainda não há outros trabalhos que a utilizem como critério de avaliação. A regra ATCS se baseia na ATC-modificada de Raman *et al.* [26] mas leva em conta outros aspectos que a tornam mais sofisticada.

A regra ATC-modificada gera um índice de prioridades baseado no qual as tarefas são sequenciadas. Este índice é obtido pela equação:

$$I_j(t,l) = \frac{w_j}{p_j + s_{lj}} \cdot \exp\left[\frac{-\max(d_j - p_j - s_{lj} - t, 0)}{k.p_{médio}}\right] \qquad (2)$$

onde p, s e d são os mesmos definidos na descrição do problema; w é o peso por unidade de tempo do atraso da tarefa – no nosso caso são todos iguais a 1; t é o instante de tempo atual e k é um parâmetro empírico – denominado *look-ahead* – que deve ser fixado de acordo com a instância – segundo Vepsalainen e Morton [31] seu valor normalmente está restrito ao intervalo [1, 3].

A determinação da sequência é feita inicialmente fixando t e l em 0 e calculando todos os $I_j(t,l)$. O l de maior valor é selecionado – suponha i – e a operação i é fixada como sendo a primeira da sequência. Em seguida, atualiza-se t como sendo $s_{0i} + p_i$. O valor de l é fixado em i e de novo calculam-se os valores de I. O processo se repete até que todas as tarefas

tenham sido sequenciadas. O ponto crítico deste método reside no parâmetro *k* que não está muito bem definido.

A modificação inserida na regra ATCS elimina este problema do parâmetro *k* e torna a equação mais robusta. A equação principal utilizada no ATCS é mostrada a seguir:

$$I_j(t,l) = \frac{w_j}{p_j} \cdot \exp\left[\frac{-\max(d_j - p_j - t, 0)}{k_1 \cdot p_{médio}}\right] \cdot \exp\left[\frac{-s_{lj}}{k_2 \cdot s_{médio}}\right] \quad (3)$$

As variáveis são definidas da mesma forma que na Equação 2. A diferença mais clara é o uso de duas funções exponenciais, o que faz com que tempos de preparação e de processamento fiquem desacoplados. A primeira exponencial trata dos tempos de processamento, ao passo que a segunda trata dos tempos de preparação. Além disso, os parâmetros k_1 e k_2 não são empíricos, sendo determinados por equações bem definidas que serão vistas mais à frente.

A heurística ATCS compreende três fases distintas:

Fase 1: Cálculo dos coeficientes
Fase 2: Construção do sequenciamento
Fase 3: Pós-otimização

2.3.1 Fase 1: Cálculo dos coeficientes

Nesta fase, três coeficientes são calculados de forma a caracterizar a instância. O primeiro é o Coeficiente de Espalhamento das Datas de Entrega, representado por τ e que serve como um estimador da porcentagem de tarefas que serão entregues com atraso. Ele é definido como:

$$\tau = 1 - \frac{d_{médio}}{C_{\max}} \quad (4)$$

onde $d_{médio}$ é a média das datas de entrega e C_{max} o *makespan* (o instante de término do processamento da última tarefa da sequência). Como C_{max} depende fundamentalmente dos s_{ij} e a sequência ainda não foi determinada, lança-se mão de um estimador:

$$C_{\max} = n.(p_{médio} + \beta.s_{médio}), \text{ com } \beta \leq 1, \quad (5)$$

onde n é o número de tarefas e β é um parâmetro a se determinar. Aqui recaímos de novo no problema de determinar um parâmetro de forma empírica. Pela Equação 5, temos que n influencia o valor de β, pois apenas n valores são selecionados da matriz dos tempos de preparação para compor a sequência final e provavelmente estes valores são pequenos. Desta forma, quando n cresce, pode-se esperar uma diminuição de β. Outro fator que influencia β é a variabilidade dos tempos de preparação. Por exemplo, se todos os valores de s_{ij} são iguais, β vale 1.

Como uma medida da variabilidade dos tempos de preparação, é definido um coeficiente de variação c_v:

$$c_v = \frac{Var(s)}{s^2_{médio}} \quad (6)$$

onde *Var(s)* é a variância dos tempos de preparação.

Nesta parte, Lee *et al.* [19] realizam uma série de experimentos que visa obter uma relação empírica entre β, c_v e n. Com base nesses resultados, obtém-se a Equação 7:

$$\beta = -1.88 \cdot c_v + 0.92 \quad (7)$$

Não houve a necessidade de inclusão de n na Equação 7 pois sua influência é dominada pela de c_v. Com o valor de β definido, pode-se calcular τ na Equação 4 e C_{max} na Equação 5. Assim, passamos a um outro coeficiente importante, o Fator de Intervalo das Datas de Entrega, representado por R:

$$R = \frac{(d_{\max} - d_{\min})}{C_{\max}} \tag{8}$$

O fator R fornece uma relação percentual entre o comprimento do intervalo das datas de entrega e o *makespan*. Finalmente, há o terceiro coeficiente, η, chamado Fator de Proporcionalidade de Tempos de Preparação, que é calculado como:

$$\eta = \frac{s_{médio}}{p_{médio}} \tag{9}$$

Este fator nos fornece uma proporção entre os valores dos tempos de preparação e dos tempos de processamento. Calculados esses três fatores, τ, β e η, pode-se dar início à segunda fase do ATCS.

2.3.2 Fase 2: Construção do sequenciamento

Nesta fase, utilizamos a Equação 3 para determinar a sequência. Porém, esta equação utiliza dois parâmetros, k_1 e k_2, ainda não definidos. Baseados em testes experimentais, Lee *et al.* [19] sugerem as seguintes relações para k_1 e k_2.

$$\begin{cases} k_1 = 4.5 + R, \text{se } R \leq 0.5 \\ k_1 = 6.0 - 2.R, \text{se } R \geq 0.5 \end{cases} \tag{10}$$

$$k_2 = \frac{\tau}{2.\sqrt{\eta}} \tag{11}$$

Com k_1 e k_2 devidamente calculados, montamos a sequência da mesma forma construtiva como no caso da equação de Raman (Equação 2). De posse da sequência, passamos agora à fase de pós-otimização.

2.3.3 Fase 3: Pós-otimização

Nesta fase, aplica-se uma busca local na sequência obtida na Fase 2. Lee *et al.* [19] sugerem a vizinhança de troca *all-pairs*, com reduções de vizinhança a fim de reduzir o esforço computacional. Neste trabalho implementamos uma busca local completa, a fim de maximizar o desempenho do ATCS.

Aplicamos inicialmente sobre a solução da Fase 2 uma busca local baseada na troca *all-pairs*. Em seguida é feita uma segunda busca local, desta vez baseada na inserção de tarefas, sobre a sequência resultante da primeira busca. Optamos por não adotar nenhuma redução de vizinhança pois o tempo computacional das buscas é muito pequeno, sempre abaixo de 2 segundos, mesmo para as maiores instâncias.

CAPÍTULO III

Algoritmos Meméticos

Algoritmos Genéticos (AG) tiveram projeção a partir de meados dos anos 70 depois da publicação do livro de John Holland intitulado "*Adaptation in Natural and Artificial Systems*". Desde então os AG se popularizaram e, nos anos 80, uma nova classe denominada "*knowledge-augmented GAs*", algumas vezes referida como Algoritmos Genéticos Híbridos (AGH) começou a aparecer na literatura da área. Reconhecendo importantes diferenças e similaridades com outras abordagens populacionais, alguns desses AGH foram categorizados como Algoritmos Meméticos em 1989 ([21], [22]). Os AM têm sido encarados apenas como um AG que utiliza indivíduos que são mínimos locais. É uma visão simplista, uma vez que um AM pode ser muito mais amplo. Por exemplo, há o exemplo da Busca por Espalhamento – *Scatter Search* – que pode ser classificado como um AM e está presente na literatura desde 1977 ([10], [11]).

Inicialmente, vamos discutir a característica básica dos AG e AM. Algoritmos não-populacionais baseiam sua busca em um único indivíduo, que percorre o espaço de soluções à procura do ótimo global. Essa busca normalmente segue regras bem definidas, podendo ter características aleatórias (*Simulated Annealing*) ou não (Busca Tabu). Um algoritmo populacional, por sua vez, utiliza um conjunto de indivíduos para vasculhar o espaço de soluções. É um processo estruturalmente paralelo, e no caso dos AG e AM existe ainda a troca de informação entre as soluções à medida que efetuam a busca. A seguir, mostramos uma ilustração do conceito de uma busca populacional e de uma não-populacional.

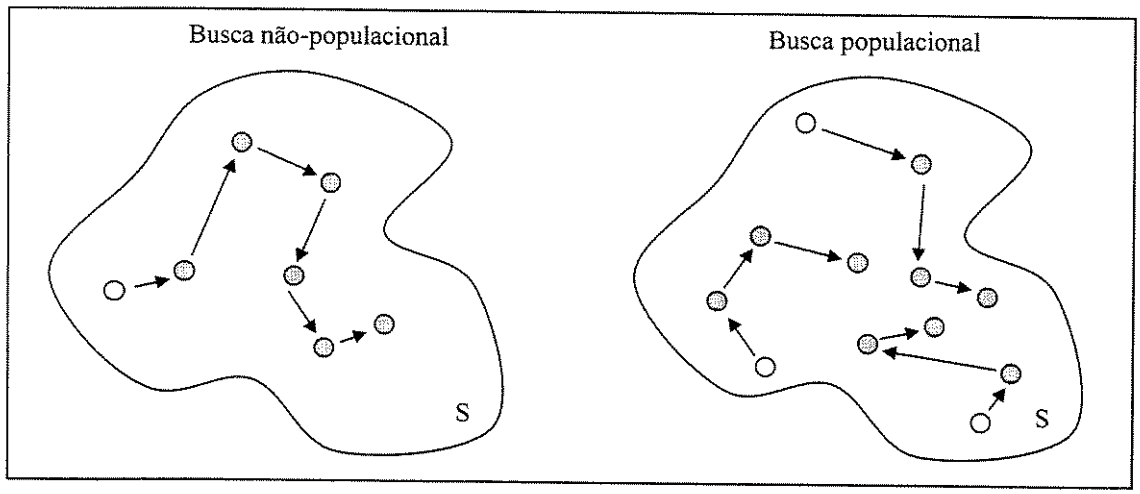


Figura 3: Ilustração comparativa entre uma busca não-populacional e uma populacional.

Como se pode ver na Figura 3, o método não-populacional normalmente consegue realizar mais iterações de busca (no exemplo foram 6) que o método populacional (apenas 3 por indivíduo) em um mesmo intervalo de tempo. Isto é natural, visto que a cada passo todos os indivíduos da população precisam "evoluir". No entanto, o fato da busca ser mais distribuída pelo espaço de soluções geralmente acaba compensando esse menor número de iterações.

Algoritmos Genéticos e Meméticos se baseiam em processos naturais, tais como recombinação, seleção, mutação, entre outros. O princípio básico consiste em selecionar bons indivíduos, identificados aqui como as soluções do problema de otimização, para reprodução e recombiná-los, com o propósito de obter soluções melhores que os pais. Esses filhos, por sua vez, tendem a ocupar o lugar dos indivíduos menos adaptados da população, melhorando a adaptabilidade da população como um todo. A mutação entra como um elemento adicionador de variedade genética. Todos esses elementos agindo sobre uma dada população levarão a um processo evolutivo. Desta forma, uma população inicialmente pouco adaptada, ao fim de um certo número de gerações, constituir-se-á na sua maioria de indivíduos bem adaptados.

Algoritmos Meméticos utilizam ainda o conceito de "evolução cultural", onde a adaptabilidade de um indivíduo pode ser modificada no decorrer de sua existência dentro da população. Um indivíduo pode ser geneticamente pouco favorecido ao nascer, mas devido às condições em que vive, por trocas de informação com outros indivíduos, experiências

pessoais, entre outros aspectos, pode se tornar mais adaptado, e mais do que isso, transmitir essa experiência aos seus descendentes (evolução cultural).

Há muitas palavras novas sendo introduzidas: recombinação, mutação, seleção, adaptabilidade, evolução. A seguir vamos explorar cada um desses novos conceitos cuidadosamente e suas respectivas implementações neste trabalho, começando com o pseudocódigo de um AM.

3.1 Pseudocódigo

Para uma melhor compreensão da estrutura de um algoritmo populacional, em especial de um AM, apresentamos o pseudocódigo de um AM com busca local e em seguida uma cuidadosa explicação das partes integrantes do mesmo.

```
procedure Algoritmo Memético;
begin
    initializePopulation Pop_principal usando Gera_Pop_Inicial();
    forEach indivíduo i ∈ Pop_principal do i := Busca_Local(i);
    forEach indivíduo i ∈ Pop_principal do Avalia_Fitness(i);
    repeat  /*laço das gerações */
        for i := 1 to #recombinações do
            selectToMerge um conjunto S_par ⊆ Pop_principal;
            novo_indivíduo := Recombinar(S_par);
            Avalia_Fitness(novo_indivíduo);
            addInPopulation indivíduo novo_indivíduo to Pop_intermediária;
        endFor;
        for i := 1 to #mutações do
            selectToMutate indivíduo i ∈ Pop_intermediária;
            i_m := Efetuar_Mutação(i);
            Avalia_Fitness(i_m);
            addInPopulation indivíduo i_m to Pop_intermediária;
        endFor;
        forEach indivíduo i ∈ Pop_intermediária do i := Busca_Local(i);
        forEach indivíduo i ∈ Pop_intermediária do Avalia_Fitness(i);
        Pop_principal := Seleciona_Pop(Pop_intermediária, Pop_principal);
        Apagar(Pop_intermediária);
    until (condição_de_parada=True);
end;
```

Como se pode ver, o procedimento possui 4 partes principais. Uma de inicialização, dois laços – um de recombinação e um de mutação – e uma parte final de otimização e atualização. Vamos examinar cada uma delas:

Inicialização

initializePopulation *Pop_principal* usando Gera_Pop_Inicial();
forEach indivíduo $i \in$ *Pop_principal* **do** i := Busca_Local(i);
forEach indivíduo $i \in$ *Pop_principal* **do** Avalia_Fitness(i);

O comando **initializePopulation** inicializa uma população principal de indivíduos através da função Gera_Pop_Inicial(). Essa população pode ser gerada de diversas formas: aleatoriamente, parte aleatória e outra parte mediante a aplicação de heurísticas construtivas, etc. Neste trabalho escolhemos uma inicialização **totalmente aleatória**. A segunda linha é um laço que aplica uma busca local em todas as soluções iniciais através da função Busca_Local(·). Isso garante que a população inicial seja composta por mínimos locais. Nos AG clássicos esta parte é suprimida. A terceira linha faz a avaliação da adaptabilidade – função Avalia_Fitness(·) – de todos os indivíduos. Essa avaliação é necessária para o início da próxima etapa, onde serão feitas as recombinações.

Laço de recombinação

for i := 1 to *#recombinações* **do**
    selectToMerge um conjunto $S_{par} \subseteq$ *Pop_principal*;
    novo_indivíduo := Recombinar(S_{par});
    Avalia_Fitness(*novo_indivíduo*);
    addInPopulation indivíduo *novo_indivíduo* to *Pop_intermediária*;
endFor;

Entramos agora no laço principal, onde se localizam os operadores evolutivos. O laço de recombinação começa após a definição do número desejado de recombinações. Na segunda linha encontramos o comando **selectToMerge** que é responsável pela escolha do conjunto S_{par} de indivíduos, composto pelos dois pais que tomarão parte no processo de

recombinação. Em seguida geramos o(s) filho(s) através da função Recombinar(S_{par}). Ela recebe o conjunto de "pais" e retorna um conjunto de filhos. Em seguida é feita a avaliação do(s) filho(s) e finalmente, com o comando **addInPopulation**, eles são inseridos em uma população intermediária separada da principal de forma que eles não participem de recombinações na mesma geração em que foram gerados.

Laço de mutação

for *i* := 1 to *#mutações* **do**
    selectToMutate an indivíduo $i \in$ *Pop_intermediária*;
    i_m := Efetuar_Mutação(*i*);
    Avalia_Fitness(i_m);
    addInPopulation indivíduo i_m to *Pop_intermediária*;
endFor;

A exemplo do laço de recombinação, o laço de mutação inicia-se após definido o número de mutações. Feito isso, o comando **selectToMutate** seleciona um indivíduo da população intermediária e aplica sobre ele uma mutação definida na função Efetuar_Mutação(·). Aplicada a mutação, a adaptabilidade do novo indivíduo é reavaliada e o indivíduo é reinserido na população intermediária.

Atualização da população

forEach indivíduo $i \in$ *Pop_intermediária* **do** *i* := Busca_Local(*i*);
forEach indivíduo $i \in$ *Pop_intermediária* **do** Avalia_Fitness(*i*);
Pop_principal := Seleciona_Pop(*Pop_intermediária, Pop_principal*);
Apagar(*Pop_intermediária*);

Esta etapa é responsável pela atualização da população principal, cujo número de indivíduos permanece constante ao longo do processo. Na primeira linha é efetuada a otimização dos novos indivíduos – os que constituem a população intermediária – e na linha seguinte realiza-se a avaliação dos mesmos. A função Seleciona_Pop(*Pop_intermediária, Pop_principal*) faz a seleção dos indivíduos da população intermediária que irão substituir indivíduos da população principal e além disso, quais indivíduos da população principal serão

substituídos. Terminada esta etapa, a função Apagar(·) elimina todos os indivíduos da população intermediária. Nesta função também incluiu-se uma rotina que verifica se foi gerada uma solução melhor que a incumbente (a melhor solução até o momento).

A seguir comentaremos cada aspecto do Algoritmo Memético implementado.

3.2 População hierarquicamente estruturada

Em geral, é muito raro encontrar trabalhos em Algoritmos Evolutivos com populações hierarquicamente estruturadas. Mais comuns são trabalhos com populações estruturadas sem hierarquia, normalmente implementados em computadores paralelos, com cada máquina ficando responsável por um ou mais indivíduos. Neste trabalho implementamos uma população estruturada e hierarquizada, comparando-a com uma população não-estruturada.

A estrutura populacional implementada possui uma única população estruturada segundo uma árvore ternária. Desta forma, podemos dividi-la em subgrupos de 4 indivíduos, cada um composto por um *líder* e três *seguidores* (veja Figura 4). Em cada subgrupo, o líder é sempre o indivíduo mais adaptado dos quatro. Esta hierarquia faz com que indivíduos dos níveis superiores sejam em geral melhor adaptados que os dos níveis inferiores. Pode-se concluir facilmente que a melhor solução estará sempre presente como o líder do subgrupo "raiz" da árvore populacional.

Como utilizamos árvores ternárias completas, o tamanho da população fica restrito a certos valores: 13, 40, 121, etc. São necessários treze indivíduos para construir uma árvore com 3 níveis, 40 para uma com 4 níveis, e assim por diante.

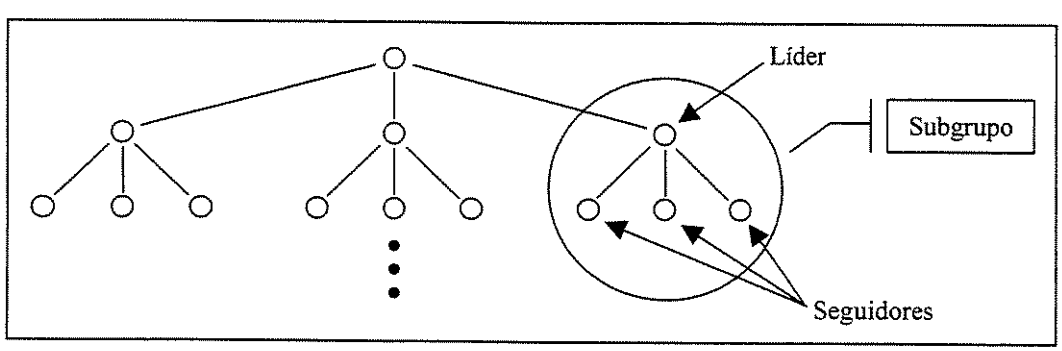


Figura 4: Estrutura populacional em árvore ternária.

3.3 Representação genética

O sucesso de um AG ou AM depende muito da representação genética escolhida para descrever as soluções como cromossomos. Essa escolha influenciará o desempenho de todas as etapas do algoritmo: avaliação da adaptabilidade, operadores de recombinação e de mutação, busca local – no caso de AM – entre outras. O ideal é sempre utilizar representações compactas, completas e estáveis.

Uma representação compacta deve se valer do menor número possível de variáveis para representar de forma unívoca uma solução.

Representações completas devem ser capazes de representar todas as possíveis soluções do problema, inclusive a ótima, é claro.

Uma representação estável tem como característica que pequenas mudanças no cromossomo levam a alterações também pequenas da adaptabilidade. Caso essas mudanças resultem em grandes alterações de adaptabilidade, poderá haver problemas na evolução da população. Pode-se chegar a um quadro onde mesmo após muitas gerações, a população como um todo seja muito pouco adaptada.

A representação escolhida para o problema de SMS é bem intuitiva e clássica, com a solução sendo representada por um cromossomo cujos alelos assumem valores inteiros e distintos no intervalo [1, *n*]. Desta forma, um exemplo de solução válida para um problema com 10 tarefas seria < 5 3 4 8 9 1 2 6 10 7 >.

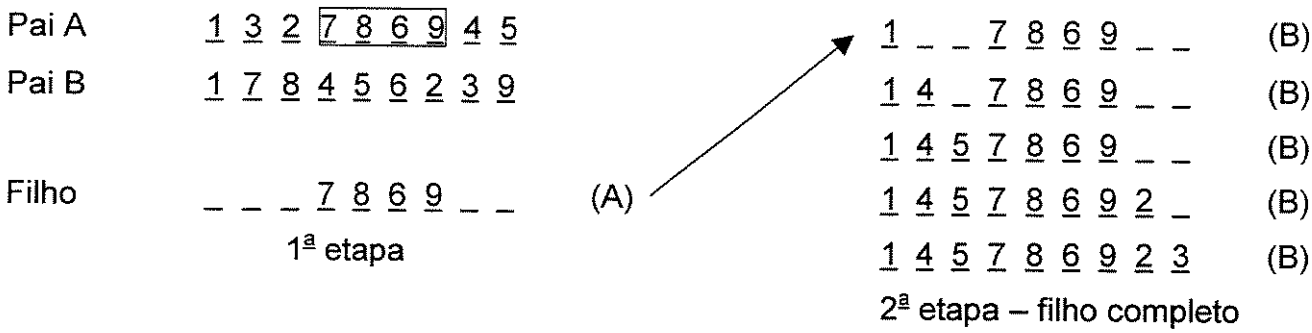
3.4 Recombinação – *Crossover*

Três tipos de operadores de recombinação, que chamaremos também por *crossover*, foram implementados neste trabalho. O primeiro é denominado *Order Crossover* (OX) [12]. O segundo é um *crossover* proposto por Rubin e Ragatz [28] para o problema 1/sds/T_{total}, e o último é um *crossover* híbrido mesclando os dois anteriores. Todos eles levam em consideração, mas de maneiras diferentes, dois critérios importantes para este problema: as posições relativa e absoluta das tarefas dentro das sequências. Um *crossover* que transmita a posição relativa das tarefas é interessante pois tende a perpetuar tempos de preparação baixos, reduzindo o *makespan* e, indiretamente, o atraso das tarefas. Porém, essa relação

makespan baixo ⇒ *atraso total baixo* é válida somente se as tarefas são posicionadas na sequência levando-se em conta suas datas de entrega. Isso é feito montando-se um esquema que preserve boas posições absolutas de tarefas.

3.4.1 Crossover OX

O procedimento se inicia com a escolha de dois pais. Uma vez escolhidos, um fragmento de um deles é selecionado aleatoriamente e copiado para o filho. Esta etapa tende a preservar boas posições absolutas, e pelo fato do fragmento ser contínuo, preserva também as posições relativas. As posições não-preenchidas do filho são então completadas com a sequência das tarefas do outro pai, varrendo-se o cromossomo deste no sentido esquerda-direita. Esta segunda etapa procura preservar a posição relativa das tarefas. O exemplo a seguir ilustra a idéia do procedimento:



No exemplo acima, o fragmento *7-8-6-9* foi selecionado do pai **A** e copiado na mesma posição do filho. As posições não completadas foram então preenchidas segundo a sequência das tarefas do pai **B**. Como pode-se notar, a característica mais marcante desse procedimento é a preservação, sempre que possível, da posição relativa das tarefas. Assim, se uma dada operação é realizada antes de outra em ambos os pais, muito provavelmente ela manterá essa relação no filho.

3.4.2 Crossover RRX

Em seu trabalho de 1995, Rubin e Ragatz [28] abordaram o problema 1/sds/T_{total} para o qual foi proposto um esquema de *crossover* específico. Como o trabalho não menciona nenhum nome em especial, decidimos referenciá-lo como RRX. O RRX tenta, a exemplo do OX, preservar tanto posições relativas quanto absolutas das tarefas. Como o *crossover* é relativamente complexo, adotaremos um exemplo com 6 tarefas para explicar o processo.

Primeiramente as tarefas são separadas em dois grupos:

Grupo 1: Tarefas com datas de entrega baixas.
Grupo 2: Tarefas com datas de entrega altas.

Desta forma, presume-se que as tarefas do grupo 1 deverão aparecer na sequência antes das tarefas do grupo 2. No nosso exemplo de 6 tarefas, suponha: Grupo 1 = {1, 2, 3} e Grupo 2 = {4, 5, 6}. Em seguida escolhem-se dois pais para recombinação. Sejam eles: **A** = < 2 6 5 3 1 4 > e **B** = < 1 4 3 6 5 2 >.

A etapa seguinte é um sequenciamento dentro dos grupos envolvendo ambos os pais.

Tabela 1: Sequenciamento dentro dos grupos.

Sequenciamento dentro dos grupos	
Grupo 1	Grupo 2
A	A
< 2 3 1 >	< 6 5 4 >
B	B
< 1 3 2 >	< 4 6 5 >

A Tabela 1 mostra as subsequências de cada pai referentes a cada grupo. Por exemplo, as tarefas do grupo 1 são encontradas na sequência < 2 3 1 > no pai **A** e na sequência < 1 3 2 > no pai **B**. A leitura deve ser a mesma para o grupo 2. Em seguida é feita

uma etapa de designação no cromossomo das posições aos grupos. As posições são nomeadas de *a* até *f*, ficando o cromossomo representado pela sequência < a b c d e f >.

Tabela 2: Designação no cromossomo das posições aos grupos.

Designação das posições aos grupos	
Grupo 1	Grupo 2
A	A
< a d e >	< b c f >
B	B
< a c f >	< b d e >

Nesta tabela estão as posições ocupadas pelas tarefas dos grupos em cada indivíduo. Por exemplo, as tarefas do grupo 1 estão nas posições < a d e > do pai **A** e nas posições < a c f > no pai **B**. O mesmo se aplica às tarefas do grupo 2. Mesclando as Tabelas 1 e 2, montamos a Tabela 3, que apresenta os possíveis filhos a serem gerados.

Tabela 3: Conjunto dos oito filhos gerados pelo crossover RRX a partir de dois pais.

		Designação das posições aos grupos	
		A	B
Sequenciamento nos grupos		Grupo 1 → < a d e >	Grupo 1 → < a c f >
Grupo 1	Grupo 2	Grupo 2 → < b c f >	Grupo 2 → < b d e >
A	A		
< 2 3 1 >	< 6 5 4 >	**< 2 6 5 3 1 4 >***	**< 2 6 3 5 4 1 >**
	B		
	< 4 6 5 >	**< 2 4 6 3 1 5 >**	**< 2 4 3 6 5 1 >**
B	A		
< 1 3 2 >	< 6 5 4 >	**< 1 6 5 3 2 4 >**	**< 1 6 3 5 4 2 >**
	B		
	< 4 6 5 >	**< 1 4 6 3 2 5 >**	**< 1 4 3 6 5 2 >***

A Tabela 3 mostra os oito possíveis filhos gerados ao se combinar grupos, tarefas e posições. Os filhos marcados com um asterisco são clones dos pais. Neste caso, como cada recombinação dá origem a 8 indivíduos, é necessária uma política de seleção nesta etapa, pois na nossa abordagem o número de indivíduos na população principal permanece constante. Rubin e Ragatz [28] sugerem uma simples seleção aleatória de um dos indivíduos, salvo se algum deles melhorar a solução incumbente. Neste caso o indivíduo em questão é selecionado. Esta foi a estratégia adotada neste trabalho.

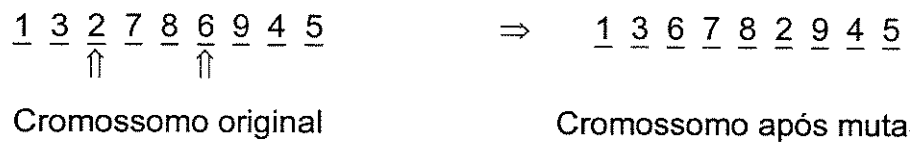
O *crossover* RRX não teve um desempenho muito satisfatório nos testes, como será mostrado depois. Cada dois pais só pode dar origem a 6 filhos diferentes, visto que dois são clones dos mesmos. Esse fato, aliado à ausência de qualquer característica aleatória no procedimento, leva a uma perda de diversidade muito acelerada, comprometendo a evolução da população.

3.4.3 Crossover híbrido OX/RRX

O terceiro tipo de *crossover* é um híbrido, composto da união dos dois *crossovers* anteriores. Desta forma, uma vez selecionados os dois indivíduos para recombinação, faz-se uma escolha aleatória de qual esquema será utilizado com igual probabilidade para ambos. Isso acarreta que em uma mesma geração aproximadamente metade das recombinações será de um tipo e metade do outro.

3.5 Mutação

Muitos pesquisadores acreditam que a mutação não possui um papel importante no processo evolutivo associado a problemas com representação de natureza similar a adotada neste estudo, e que na presença de um bom esquema de recombinação, a mutação se torna dispensável. Neste trabalho viemos a comprovar que a mutação é sem dúvida muito importante para o nosso problema, como um fator de adição de diversidade, chegando até a cumprir o papel de uma busca local limitada – no caso do AG. A mutação implementada – Efetuar_Mutação(·) – consiste da troca de posição entre tarefas. Desta forma, duas posições são selecionadas aleatoriamente e os alelos dessas posições trocam os valores entre si.



3.6 Função de avaliação – *fitness*

A função de avaliação tem por finalidade quantificar a qualidade dos indivíduos gerados. Para tanto ela deve guardar uma relação estreita com a função objetivo do problema em questão. No nosso caso, utilizamos uma função já bem conhecida e que pode ser encontrada em outros trabalhos ([4], [28]).

A função escolhida é descrita pela Equação 12:

$$f_i = (T_i + 1)^{-1} \tag{12}$$

onde T_i é o atraso total da solução i, calculado pela Equação 1. O expoente negativo é devido ao fato de que um atraso total pequeno indica uma boa solução, cujo *fitness* consequentemente deve ser alto. Há, dessa forma, uma relação inversa entre a adaptabilidade e o atraso total. A soma da unidade ao atraso total é um mero artifício matemático para se evitar uma divisão por zero se o atraso total da solução também for zero. Lembramos que, da forma como foi definido na entrada do problema, o atraso total está restrito ao conjunto dos inteiros não-negativos, estando incluído assim o elemento zero.

3.7 Seleção para recombinação

A forma de selecionar indivíduos para recombinação varia se estamos trabalhando com a população não estruturada ou com a população estruturada. Desta forma, subdividimos esta seção para abordar os dois casos.

População não-estruturada

A seleção para recombinação implementada segue um procedimento de caráter aleatório conhecido como *Roulette Wheel* ou "Roleta Ponderada". Apesar de aleatório, esse

procedimento utiliza uma boa quantidade de conhecimento na escolha dos indivíduos. O intervalo [0,1] é inicialmente repartido entre todos os indivíduos, com os indivíduos mais adaptados recebendo fatias proporcionalmente maiores. Um número aleatório entre zero e um é gerado e o elemento cujo intervalo contiver este número é selecionado. Esse procedimento faz com que indivíduos melhor adaptados sejam selecionados mais frequentemente que os menos adaptados, sem eliminar a possibilidade de indivíduos pouco adaptados também serem selecionados. Com isso, pode-se esperar uma diversidade maior na população do que quando seleções determinísticas são utilizadas.

A Figura 5 exemplifica o procedimento de *Roulette Wheel*. Com base nos valores do atraso total T_i, calculamos o *fitness* F_i de cada indivíduo. Em seguida calculamos a porcentagem com que cada indivíduo contribui para a adaptabilidade total da população. Ao se gerar um número aleatório no intervalo [0, 1], verificamos a qual intervalo ele pertence e o indivíduo em questão é selecionado. Por exemplo, na Figura 5, se o número gerado for 0,773, o indivíduo escolhido será o 4. Pela observação do gráfico de setores, fica clara a maior probabilidade de escolha de indivíduos mais bem adaptados.

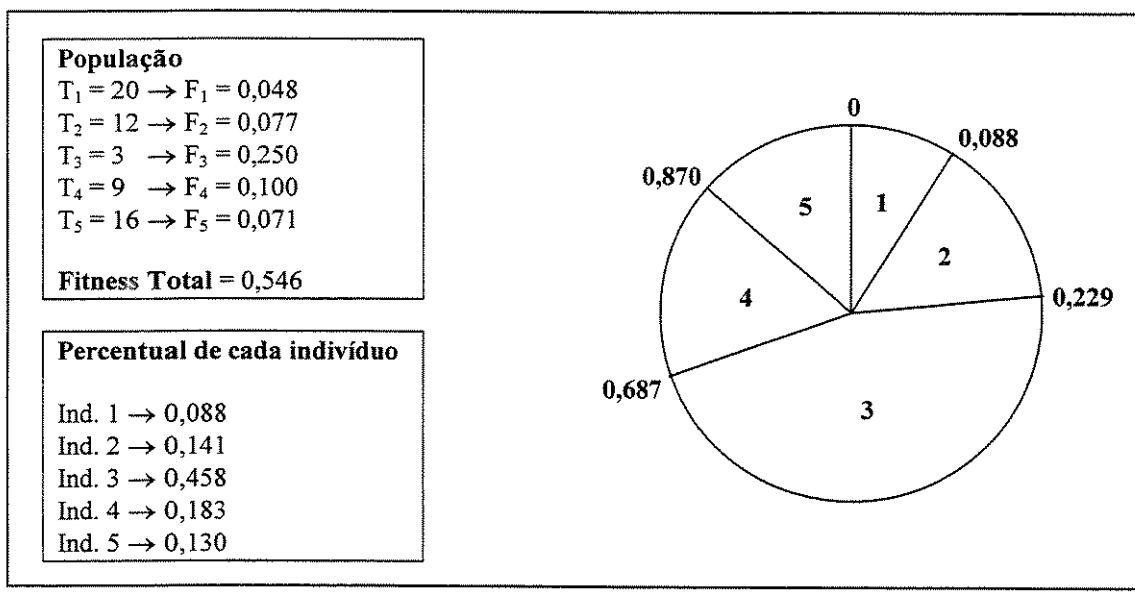


Figura 5: Exemplo de um procedimento tipo *Roulette Wheel* com uma população de 5 indivíduos.

Apesar do *Roulette Wheel* tradicional se comportar razoavelmente bem, notamos alguns problemas na condução da escolha dos indivíduos. Testes iniciais indicaram que o melhor indivíduo da população participava de quase todas as recombinações.

Desta forma foi necessária a criação de mecanismos que mantivessem a diversidade nas escolhas, diminuindo a pressão sobre a solução incumbente. O primeiro mecanismo faz com que no início do laço de recombinação todos os indivíduos possam ser escolhidos, mas à medida em que vão sendo selecionados, ficam proibidos de serem selecionados de novo naquela geração. Essa proibição se aplica a todos os indivíduos com exceção do melhor, cuja participação fica restrita a no máximo 50% dos *crossovers*. Se em um dado instante mais de 70% da população estiver proibida de recombinar, as proibições caem e todos voltam a poder participar de recombinações. Esse limite de 70% só pode ser alcançado se o número de recombinações por geração for muito alto em relação ao número de indivíduos presentes na população principal. Com isso, na prática, cada indivíduo não-incumbente só participa de no máximo dois *crossovers* por geração.

Esses mecanismos, agindo em conjunto, levaram a resultados melhores que os do *Roulette Wheel* original, reduzindo a queda da diversidade sem prejudicar a obtenção de boas soluções.

O *Roulette Wheel* foi utilizado apenas nos testes com população não-estruturada. Para a população estruturada utilizamos um método próprio de seleção, como será visto a seguir.

População estruturada

Na seleção dos indivíduos para recombinação, primeiramente é escolhido um líder, aleatoriamente e com igual probabilidade para todos os líderes. O próximo passo é escolher qual dos 3 seguidores participará da recombinação. Essa seleção é de novo aleatória e também com igual probabilidade para todos os três. Não há restrição ao número de vezes que um dado indivíduo pode participar de *crossovers* em uma mesma geração.

Para a população estruturada foi elaborado um mecanismo de intensificação sobre o melhor indivíduo: ele é escolhido aproximadamente 10% a mais que os outros líderes.

3.8 Inserção de novos indivíduos

A inserção de novos indivíduos também segue normas distintas quando a população é não-estruturada ou estruturada. A seguir apresentamos as políticas de inserção para ambos os casos.

População não-estruturada

Como dito antes, nossa implementação utiliza duas populações distintas: uma principal e uma intermediária. À medida que novos indivíduos vão sendo criados na etapa de recombinação, estes vão sendo copiados para a população intermediária. Ao fim de cada geração, é necessário atualizar a população principal, inserindo os novos indivíduos. A adoção da política de manter constante o número de indivíduos, torna necessária a eliminação de indivíduos antigos para que os novos tomem seus lugares. A escolha das soluções a serem eliminadas é inteiramente determinística.

Primeiramente ordenamos a população principal pela ordem não-decrescente de *fitness* e em seguida efetuamos a substituição dos indivíduos segundo essa sequência. Mecanismos não-aleatórios tendem a causar uma perda de diversidade acelerada, e dessa forma tentamos reduzir esse efeito pela imposição de uma condição de unicidade de soluções. Essa condição exige que um indivíduo, para ser inserido na população principal, seja único, o que na prática proíbe a presença de soluções duplicadas na população.

Outra característica adotada foi o elitismo, que obriga a presença da solução incumbente na população principal no início de cada geração. Na realidade, não é necessária nenhuma rotina de inserção à parte, pois a única forma da solução incumbente ser apagada da população principal é se o número de recombinações for igual ao número de indivíduos, o que geraria um número de filhos igual ao número de indivíduos da população principal. Em nossos testes, no entanto, o número de recombinações sempre esteve bem abaixo desse número. Por fim, terminada a etapa de inserção, a população intermediária é eliminada pela função Apagar(·).

População estruturada

Cada novo indivíduo gerado tem duas opções de inserção na população principal. Se seu *fitness* for melhor que o do líder do subgrupo, ele toma o seu lugar. Caso contrário, ele toma o lugar do seguidor que participou do *crossover* que lhe deu origem, independente de seu *fitness*. Para a população estruturada também há o problema de perda de diversidade acelerada e decidimos, então, manter a característica de unicidade dos indivíduos.

Uma vez inseridos todos os novos indivíduos não-duplicados, inicia-se a etapa de atualização da população, caracterizada pela reestruturação da mesma. Como a árvore deve manter a hierarquia entre líderes e seguidores, verifica-se se algum indivíduo se tornou melhor adaptado que o líder do seu subgrupo. Caso isso ocorra, eles trocam de posição. Ao efetuar essa comparação para todos os subgrupos, mantém-se a consistência hierárquica da população.

3.9 Ajuste dos parâmetros do AG e do AM

Apesar de serem abordagens poderosas, metaheurísticas em geral apresentam um problema: ajustes de parâmetros. Na Busca Tabu (BT) temos o tempo de proibição de um movimento, critério de aspiração, entre outros elementos que precisam ser ajustados. Em *Simulated Annealing* (SA) temos a temperatura inicial e a taxa de esfriamento. AG e AM não fogem a essa necessidade de ajustes: tamanho da população, taxa de mutação e de *crossover*, são apenas alguns dos elementos que precisam ser ajustados para um melhor desempenho do método. Muito poucos trabalhos dão ao ajuste dos parâmetros a importância que merece. A fim de tornar a implementação tanto do AG quanto do AM a melhor possível, implementamos um procedimento de ajuste automático. Os parâmetros considerados críticos para o bom desempenho foram: tamanho da população, taxa de *crossover* e taxa de mutação. O procedimento de ajuste e seu pseudocódigo é apresentado a seguir:

```
procedure Ajuste_de_Parâmetros;
  begin
     finaliza_procedimento := false;
     initializeInstances conjunto_instâncias using Inicializa_Conjunto_Instâncias();
     initializeParameters [melhor_tamanho_pop, melhor_taxa_recomb, melhor_taxa_mut]
                                                    usando Inicializa_Parâmetros();
     do
        finaliza_procedimento := true;
        for tamanho_pop := min_tamanho_pop to max_tamanho_pop step passo_pop
           valor := Resolve_Conjunto_Instâncias();
           if valor < melhor_valor do
              melhor_valor := valor;
              Atualizar (melhor_tamanho_pop);
              finaliza_procedimento := false;
           endIf
        endFor
        for taxa_recomb := min_taxa_recomb to max_taxa_recomb step passo_recomb
           valor := Resolve_Conjunto_Instâncias();
           if valor < melhor_valor do
              melhor_valor := valor;
              Atualizar (melhor_taxa_recomb);
              finaliza_procedimento := false;
           endIf
        endFor
        for taxa_mut := min_taxa_mut to max_taxa_mut step passo_mut
           valor := Resolve_Conjunto_Instâncias();
           if valor < melhor_valor do
              melhor_valor := valor;
              Atualizar (melhor_taxa_mut);
              finaliza_procedimento := false;
           endIf
        endFor
     while (finaliza_procedimento = false);
  end;
```

O procedimento descrito possui 4 partes principais. A primeira é uma sequência de inicializações das instâncias que serão usadas como base para o desempenho do método, seguida pela inicialização dos parâmetros que serão ajustados.

Terminada essa fase inicia-se o laço de otimização em si. Ele consiste de três laços: um para o número de indivíduos da população, um para a taxa de *crossover* e um para a taxa de mutação. A cada alteração desses valores, verifica-se a qualidade das soluções obtidas para o conjunto de instâncias. Se os resultados forem os melhores obtidos até então – como o problema é de minimização verificamos se *valor* é menor que *melhor_valor* – atualiza-se o parâmetro que está sendo ajustado e o valor do melhor resultado obtido até o momento. O critério de parada é simples: se após uma rodada completa, passando pelos três parâmetros, não foi possível melhorar *melhor_valor*, o algoritmo pára e são exibidos os parâmetros que geraram o melhor *melhor_valor.*

População estruturada

Cada parâmetro possui um intervalo de avaliação e esse intervalo está dividido em possíveis valores que o parâmetro pode assumir. Para a população estruturada este valores são:

Número de indivíduos : 13, 40 ou 121 indivíduos (árvores ternárias com 3, 4 ou 5 níveis)
Taxa de *crossover* : de 0 a 100% (em incrementos de 10%)
Taxa de mutação : de 0 a 100% (em incrementos de 10%)

No AM foi utilizada a busca local de troca *all-pairs*, sem redução de vizinhança. Os parâmetros são fixados inicialmente em valores aleatórios quaisquer dentro dos intervalos mostrados anteriormente. Para os testes foram utilizadas duas instâncias geradas aleatoriamente e o atraso total é a variável que decide pela atualização dos parâmetros ou não.

Pode-se notar que não são avaliadas todas as possibilidades de combinação de valores, mas apenas um subconjunto delas. Uma busca enumerativa completa exigiria 363 avaliações, ao passo que uma passagem no algoritmo descrito exige apenas 25 avaliações. Como em duas ou no máximo três passagens já não há mais melhoria, o número de avaliações não passa de 75 em todos os casos. A avaliação de desempenho foi feita com base no *crossover* OX, tanto no AG quanto no AM. Veja a tabela a seguir com os resultados:

Tabela 4: Ajuste dos parâmetros para o AG e para o AM com uso de população estruturada.

	Algoritmo Genético			Algoritmo Memético		
n	**Número de indiv.**	**Taxa de *crossover***	**Taxa de mutação**	**Número de indiv.**	**Taxa de *crossover***	**Taxa de mutação**
20	121	0,3	0,8	40	0,2	0,1
40	121	0,4	0,7	40	0,3	0
60	40	0,3	0,6	13	0,2	0,1
80	40	0,5	0,8	13	0,3	0
100	13	0,5	0,8	13	0,2	0

População não-estruturada

O ajuste para a população não estruturada seguiu o mesmo padrão da estruturada, porém os valores possíveis para o número de indivíduos está compreendido no intervalo entre 10 e 150, em incrementos de 10 indivíduos.

Tabela 5: Ajuste dos parâmetros para o AG e para o AM com uso de população não-estruturada.

	Algoritmo Genético			Algoritmo Memético		
n	**Número de indiv.**	**Taxa de *crossover***	**Taxa de mutação**	**Número de indiv.**	**Taxa de *crossover***	**Taxa de mutação**
20	100	0,5	0,7	80	0,3	0,1
40	70	0,4	0,8	60	0,5	0,1
60	70	0,4	0,9	50	0,4	0,0
80	50	0,5	0,8	40	0,3	0,0
100	30	0,6	0,7	20	0,3	0,1

Baseado nos números das Tabela 4 e 5, nota-se uma oscilação dos valores que impossibilita a determinação de uma "regra de formação" para os valores ideais. Existem diversas configurações de parâmetros com valores muito próximos da melhor – a até 5% do menor atraso encontrado. Assim, os números apresentados não devem ser encarados como valores absolutos, mas sim como uma base aproximada de escolha.

A taxa de *crossover* oscilou no intervalo [0,3; 0,5] e [0,2; 0,3] para o AG e AM estruturado, respectivamente. Para a versão não estruturada os valores foram levemente maiores. Isto leva à conclusão de que taxas de recombinação devem ser mantidas em patamares baixos. Já a taxa de mutação apresentou um comportamento interessante. No AG ela deve ser mantida bem alta – ao redor de 0,8 – ao passo que no AM ela deve ser muito baixa, ou mesmo zero. Uma mutação alta acaba por agir como uma busca local bem restritiva no AG. No AM essa mutação age como um gerador de ruído que aparentemente é prejudicial ao método.

O comportamento do número de indivíduos na população também apresentou resultados bem interessantes. Tanto no AG quanto no AM, o tamanho da população diminui com o aumento do número de tarefas. Isto se deve ao fato de que, dado um tempo computacional fixo, é mais proveitoso ter menos indivíduos e deixá-los evoluir por mais gerações do que ter muitos indivíduos e efetuar poucas gerações. A evolução tem um papel muito importante no resultado, sendo necessárias centenas, às vezes milhares de gerações para se chegar à soluções de boa qualidade. Note que a população inicial utilizada é inteiramente aleatória.

No AM, por sua vez, foi verificado que a população deve ser composta por um número de indivíduos menor que no AG. A demora associada à busca local torna o algoritmo mais sensível à falta de tempo para evoluir.

3.10 Resultados Computacionais: Estrutura Populacional

Nesta seção apresentamos alguns resultados computacionais para as populações estruturada e não-estruturada do AM e do AG. A seguir, apresentamos uma tabela comparando ambas as abordagens a uma estratégia de MS. Tanto o MS quanto o AM e o AG utilizam a mesma busca local baseada na troca *all-pairs* e o mesmo tempo

computacional: 2 minutos. Os números representam a melhoria percentual média sobre a estratégia de MS (ver Equação 16). Foram testadas 5 instâncias aleatórias para cada valor de *n*. Valores negativos indicam desempenho inferior ao desempenho do MS.

Tabela 6: Comparação entre populações estruturadas e não-estruturadas.

	Algoritmo Genético		**Algoritmo Memético**	
n	**População não-estruturada**	**População estruturada**	**População não-estruturada**	**População estruturada**
20	-2,3	-2,7	1,9	1,7
40	-2,8	-3,0	13,2	13,4
60	-9,2	-7,0	12,4	14,1
80	-11,2	-6,3	10,1	12,6
100	-17,0	-8,6	6,1	11,9
Média	-8,5	-5,5	8,7	10,7

Os resultados da Tabela 6 indicam uma melhora de desempenho quando a estrutura populacional é utilizada. Para instâncias pequenas os valores são muito similares, mas para 80 e 100 tarefas a diferença se torna mais evidente. Os frustrantes valores negativos para o AG nessa etapa mostram que para este problema a busca local cumpre um papel muito importante no processo de busca. Ressalte-se que ambos os algoritmos – AG e AM – foram implementados com as mesmas características, a única diferença é a função Busca_Local(·) que é chamada no caso do AM.

3.11 Busca local

Algoritmos de busca local se baseiam na definição de uma vizinhança que estabelece uma relação entre soluções no espaço de configurações. Neste trabalho, duas vizinhanças foram implementadas, além de diversas políticas de redução.

A vizinhança de uma solução pode ser descrita como o conjunto de soluções que são alcançáveis através da execução de qualquer movimento definido pelo mecanismo de geração da vizinhança.

Por exemplo, uma das vizinhanças implementadas e já citada anteriormente foi a troca *all-pairs*. Ela consiste em trocar de forma ordenada todos os pares de tarefas em uma dada solução. Seja a sequência < *a b c d e f g h* >. Inicialmente trocamos as posições (*a*,*b*), gerando a solução < *b a c d e f g h* >. Se a troca for vantajosa, confirmamos a mudança e continuamos o processo. Caso contrário a troca é desfeita. Seguindo a sequência trocamos (*a*,*c*), (*a*,*d*), até o par (*a*,*h*). O processo é reiniciado agora a partir da segunda posição: *b*. Trocamos (*b*,*c*), (*b*,*d*) até (*b*,*h*). O processo termina na troca das posições (*g*,*h*). Ao terminar uma passagem completa, verificamos se alguma mudança foi vantajosa. Em caso positivo realizamos uma nova passagem, pois a solução não é necessariamente um mínimo local. Se nenhuma troca foi confirmada em uma passagem completa então a solução é um mínimo local e podemos parar.

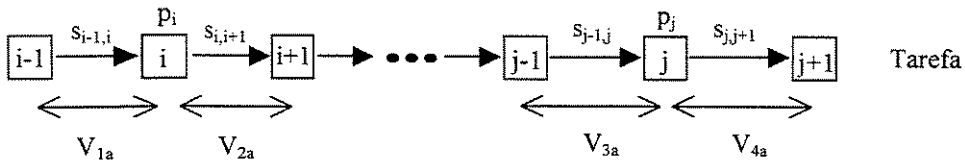
A segunda vizinhança implementada foi a de Inserção. Ela consiste em retirar uma tarefa de sua posição original e inseri-la em outra. A varredura e o critério de parada são análogos aos da vizinhança *all-pairs*.

Ambas a vizinhanças são $O(n^2)$, o que significa que o número de trocas/inserções cresce de forma quadrática com o número de tarefas. Assim, devido ao tempo necessário para se varrer toda a vizinhança, decidimos implementar algumas políticas de redução para as duas vizinhanças. Políticas de redução são importantes em casos onde a avaliação do *fitness* é custosa. No problema $1/sds/T_{total}$ isso é verdade pois cada avaliação envolve um laço que se inicia na posição da primeira tarefa trocada/inserida e termina no final da sequência.

3.11.1 Reduções de vizinhança para a troca all-pairs

Todas as reduções de vizinhança implementadas utilizam a informação de como os tempos de preparação serão alterados caso a troca seja confirmada.

Configuração antes da troca:

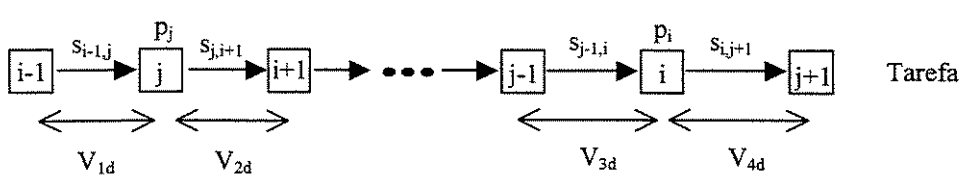


$V_{1a} = s_{i-1,i} + p_i$
$V_{2a} = s_{i,i+1}$
$V_{3a} = s_{j-1,j} + p_j$
$V_{4a} = s_{j,j+1}$

Figura 6: Configuração dos tempos de preparação e de processamento antes da troca das tarefas.

A Figura 6 mostra os tempos de preparação e os tempos de processamento que influenciarão o atraso total caso as tarefas *i* e *j* sejam trocadas. Os valores V's estão descritos em função das variáveis *s* e *p* e o critério que decide se a troca será ou não avaliada baseia-se neles.

Configuração após a troca:



$V_{1d} = s_{i-1,j} + p_j$
$V_{2d} = s_{j,i+1}$
$V_{3d} = s_{j-1,i} + p_i$
$V_{4d} = s_{i,j+1}$

Figura 7: Configuração dos tempos de preparação e de processamento após a troca das tarefas.

A decisão se uma troca deverá ou não ser avaliada leva em conta a variação dos V's. Se eles melhoram com a troca, a solução deve ser avaliada. Como existem quatro valores de

V envolvidos, há várias combinações possíveis. A Tabela 7 ilustra as quatro políticas de decisão.

Tabela 7: Descrição das quatro políticas para a redução de vizinhança de troca all-pairs.

Política	Teste para efetuar a avaliação
T-1	$(V_{1d} < V_{1a}) \vee (V_{2d} < V_{2a}) \vee (V_{3d} < V_{3a}) \vee (V_{4d} < V_{4a})$
T-2	$[(V_{1d} < V_{1a}) \vee (V_{2d} < V_{2a})] \wedge [(V_{3d} < V_{3a}) \vee (V_{4d} < V_{4a})]$
T-3	$[(V_{1d} < V_{1a}) \wedge (V_{2d} < V_{2a})] \vee [(V_{3d} < V_{3a}) \wedge (V_{4d} < V_{4a})]$
T-4	$(V_{1d} + V_{2d} < V_{1a} + V_{2a}) \vee (V_{3d} + V_{4d} < V_{3a} + V_{4a})$

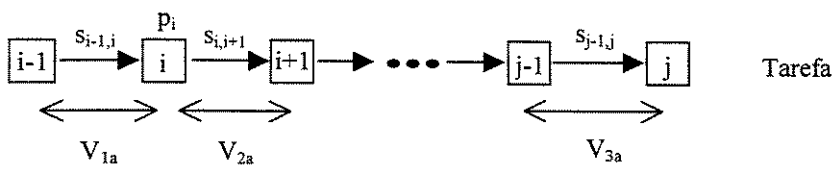
A política **T-1** exige que qualquer um dos valores de V diminua para a troca ser avaliada. Na política **T-2** a troca será avaliada se pelo menos um dos valores de V relacionados à mesma tarefa for reduzido em ambas as tarefas. A política **T-3** exige que ambos os V's referentes à mesma tarefa diminuam em pelo menos uma delas. Por fim, na política **T-4**, se pelo menos uma das somas dos valores dos V's relacionados às tarefas diminuir, a troca é avaliada.

A política **T-1** é a menos restritiva de todas, reduzindo a vizinhança *all-pairs* em pouco menos de 50%. As políticas **T-2** e **T-3** são altamente restritivas com redução da ordem de 95%. A política **T-4** por sua vez elimina aproximadamente 80% da vizinhança *all-pairs*.

3.11.2 Reduções de vizinhança para a Inserção

As duas reduções de vizinhança de Inserção são descritas nas Figuras 8 e 9, a seguir.

Configuração antes da inserção:



$V_{1a} = s_{i-1,i} + p_i$

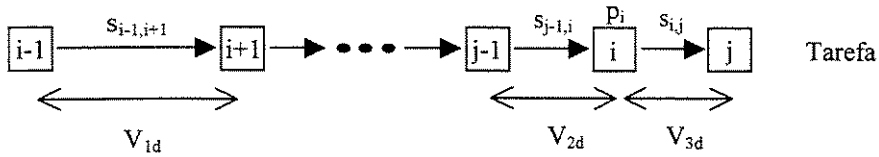
$V_{2a} = s_{i,i+1}$

$V_{3a} = s_{j-1,j}$

Figura 8: Configuração dos tempos de preparação e de processamento antes da inserção.

A Figura 8 mostra a configuração antes da tarefa *i* ser retirada de sua posição inicial e inserida imediatamente antes da tarefa *j*. Neste caso temos apenas três valores para V, uma vez que a tarefa *j+1* não é relevante para a avaliação do movimento e, portanto, não está representada na figura.

Configuração depois da inserção:



$V_{1d} = s_{i-1,i+1}$
$V_{2d} = s_{j-1,i} + p_i$
$V_{3d} = s_{i,j}$

Figura 9: Configuração dos tempos de preparação e de processamento após a inserção da tarefa em outra posição.

A configuração após a inserção muda consideravelmente. Onde estava posicionada a tarefa retirada *i* havia dois valores relevantes de V. Com a execução do movimento, passamos a ter apenas um valor. Já na região da inserção passamos a ter dois valores de V, ao invés de apenas um. As políticas de avaliação levam em conta essa mudança e estão listadas na tabela a seguir.

Tabela 8: Descrição das duas políticas de decisão para a redução de vizinhança de Inserção.

Política	Teste para efetuar a avaliação
I-1	$(V_{1d} < V_{1a} + V_{2a}) \wedge (V_{2d} + V_{3d} < V_{3a})$
I-2	$(V_{1d} < V_{1a} + V_{2a}) \vee (V_{2d} + V_{3d} < V_{3a})$

As políticas neste caso também dividem os V's em grupos. O grupo da região da tarefa *i* e o grupo da região da tarefa *j*. A política **I-1** exige que em ambas as regiões o valor dos V's melhore. No caso de serem dois os valores de V, comparamos a soma deles com o

valor anterior. A política **I-2**, por sua vez, exige que a melhora ocorra em pelo menos uma das duas regiões.

O esquema **I-1** é extremamente restritivo, reduzindo a vizinhança de Inserção em mais de 99%. Já no **I-2**, essa redução é de aproximadamente 50%.

3.11.3 Resultado das reduções de vizinhanças

A seguir, apresentamos uma série de tabelas de desempenho do AM para cada uma das reduções e para as duas vizinhanças completas, usando como base de comparação o método MS com a vizinhança *all-pairs*. Os valores são a melhoria percentual, isto é, em quanto o atraso total foi diminuído (ou aumentado), se comparado ao MS. Os testes foram executados para o AM com população estruturada, *crossover* OX e para várias configurações de $(\tau; R)$. Esses dois parâmetros classificam as instâncias em função da porcentagem aproximada de tarefas atrasadas na sequência ótima e da dispersão das datas de entrega ao longo do tempo, respectivamente. Em linhas gerais, um valor de τ pequeno indica que a sequência ótima terá atraso total próximo de zero. Similarmente, um valor de R pequeno indica que as datas de entrega estão concentradas em um intervalo de tempo também pequeno. O raciocínio é análogo para τ e R grandes. Mais à frente discutiremos τ e R com maior profundidade. Não utilizamos $\tau = 0{,}2$ pois este valor gera desvios percentuais elevados, o que poderia facilmente distorcer o resultado médio final.

Algoritmo Memético com população estruturada e crossover OX

Tabela 9: $(\tau; R) = (0{,}6; 0{,}2)$

n	Troca *all-pairs*	T-1	T-2	T-3	T-4	Inserção	I-1	I-2	T-2/I-2
20	2,1	2,5	2,0	1,0	2,4	1,0	-2,0	0,8	1,7
40	12,8	12,8	13,1	10,7	13,0	10,3	-2,4	9,0	16,2
60	9,8	10,0	11,7	9,7	10,5	7,9	-13,9	10,0	14,7
80	9,5	10,8	11,9	12,2	11,6	5,0	-14,6	10,2	12,8
100	8,7	8,5	9,5	10,6	10,1	5,2	-22,9	5,9	13,1
Média	8,6	8,9	**9,6**	8,8	9,5	5,9	-11,2	**7,2**	**11,7**

Tabela 10: (τ; R) = (0,6; 0,6)

n	Troca *all-pairs*	T-1	T-2	T-3	T-4	Inserção	I-1	I-2	T-2/I-2
20	1,2	1,5	0,8	0,6	2,5	0,6	0,4	2,1	2,1
40	17,8	15,0	15,6	12,1	12,8	11,0	1,4	13,0	18,5
60	13,2	17,9	18,0	13,2	16,3	5,2	2,0	8,2	15,6
80	8,6	11,4	13,3	14,6	12,6	2,6	-18,3	6,7	16,2
100	7,4	5,7	9,5	8,4	12,8	-4,2	-28,1	0,9	15,5
Média	9,6	10,3	**11,4**	9,8	**11,4**	3,0	-8,5	**6,2**	**13,6**

Tabela 11: (τ; R) = (0,6; 1,0)

n	Troca *all-pairs*	T-1	T-2	T-3	T-4	Inserção	I-1	I-2	T-2/I-2
20	0,7	0,7	0,3	0,1	0,2	0,3	0,2	0,5	0,5
40	16,0	17,7	15,9	12,9	14,5	4,2	-9,6	6,8	15,5
60	16,2	18,0	17,6	16,5	19,5	4,8	-24,2	7,5	16,7
80	13,6	12,9	17,2	13,7	15,7	1,2	-27,2	3,9	18,4
100	9,1	11,7	14,4	11,9	13,2	-2,1	-37,9	1,8	17,5
Média	11,1	12,2	**13,1**	11,0	12,6	1,7	-19,7	**4,1**	**13,7**

Tabela 12: (τ, R) = (1,0; 0,2)

n	Troca *all-pairs*	T-1	T-2	T-3	T-4	Inserção	I-1	I-2	T-2/I-2
20	0,2	0,0	0,0	-0,1	0,0	0,1	-2,5	0,1	0,2
40	5,8	6,0	5,1	5,3	5,4	3,7	-6,7	4,0	6,2
60	6,6	6,0	6,1	6,1	6,0	3,6	-9,5	5,3	7,3
80	6,0	6,4	6,3	6,8	6,5	1,5	-16,6	2,9	6,8
100	3,4	3,4	4,9	4,6	5,0	-1,0	-17,8	0,8	6,2
Média	4,4	4,4	4,5	4,5	**4,6**	1,6	-10,6	**2,6**	**5,3**

Tabela 13: (τ; R) = (1,0; 0,6)

n	Troca *all-pairs*	T-1	T-2	T-3	T-4	Inserção	I-1	I-2	T-2/I-2
20	0,4	0,4	0,4	0,4	0,4	0,1	-1,9	0,3	0,7
40	4,4	4,9	4,9	3,5	5,0	2,9	-8,3	2,5	4,7
60	4,7	4,9	5,5	4,8	5,2	3,4	-10,1	4,5	7,9
80	4,6	4,3	5,3	3,6	5,3	0,9	-15,1	1,6	6,5
100	3,3	3,9	5,2	4,0	4,5	0,6	-17,3	1,4	6,2
Média	3,5	3,7	**4,3**	3,3	4,1	1,6	-10,5	**2,1**	**5,2**

Tabela 14: (τ; R) = (1,0; 1,0)

n	Troca *all-pairs*	T-1	T-2	T-3	T-4	Inserção	I-1	I-2	T-2/I-2
20	0,2	0,2	0,2	0,2	0,2	0,3	-0,9	0,2	0,3
40	4,7	5,3	5,3	4,4	4,9	3,8	-5,0	2,5	6,7
60	5,7	5,5	5,6	5,4	5,5	2,5	-6,9	3,2	9,1
80	4,1	6,0	5,8	5,2	5,5	1,1	-14,6	2,4	6,0
100	3,3	2,9	4,7	3,6	3,3	-1,0	-14,3	1,6	5,9
Média	3,6	4,0	**4,3**	3,8	3,9	1,3	-8,3	**2,0**	**5,6**

Tabela 15: média de (τ; R)

n	Troca *all-pairs*	T-1	T-2	T-3	T-4	Inserção	I-1	I-2	T-2/I-2
20	0,8	0,9	0,6	0,4	1,0	0,4	-1,1	0,7	0,9
40	10,2	10,3	10,0	8,2	9,3	6,0	-5,1	6,3	11,3
60	9,4	10,4	10,8	9,3	10,5	4,6	-10,4	6,5	11,9
80	7,7	8,6	10,0	9,4	9,5	2,0	-17,7	4,6	11,1
100	5,9	6,0	8,0	7,2	8,2	-0,4	-23,0	2,1	10,7
Média	6,8	7,2	**7,9**	6,9	7,7	2,5	-11,5	**4,0**	***9,2***
Fração	*	0,51	0,05	0,05	0,21	*	< 0,01	0,50	0,24

Pelas Tabelas 9-14 e pela tabela 15 que mostra os resultados médios alcançados, nota-se um crescimento da melhora em relação ao MS até um máximo entre 40 e 80 tarefas, seguido por um decréscimo em 100 tarefas. Para 20 tarefas o MS consegue resultados muito bons, pois a vizinhança de troca *all-pairs* consegue varrer com grande eficiência o espaço de soluções, sobrando muito pouco para o AM melhorar. Desta forma os resultados não ultrapassam os 2,5% de melhoria.

A região de melhor desempenho do AM está localizada entre 40 e 80 tarefas. O espaço de soluções já é bem maior e o MS só consegue explorá-lo de forma bem restrita. O AM então aproveita seu método de busca mais eficiente e obtém resultados expressivos. O grande número de gerações que o AM consegue executar contribui também para essa melhor exploração.

Com 100 tarefas, o desempenho do AM começa a piorar levemente. Dependendo da vizinhança temos um número de gerações muito pequeno e o AM acaba por evoluir pouco nos dois minutos de processamento alocado para a resolução de cada instância.

Na Tabela 15, a linha intitulada "Fração" mostra o tamanho percentual da vizinhança reduzida em relação às vizinhanças completas. Curiosamente, nas vizinhanças de troca, as políticas de redução com melhor e pior desempenho são justamente as mais restritivas, o que elimina qualquer relação entre tamanho de vizinhança e desempenho neste caso. Assim concluímos que mais importante que o tamanho é a estrutura da vizinhança ser compatível com o problema.

A melhora do AM com vizinhança de troca *all-pairs* sem redução teve um desempenho médio de 6,8%. Esse valor foi ultrapassado por todas as estratégias de redução, sendo que a melhor redução é a T-2, seguida de perto pela T-4.

Nas vizinhanças de inserção as diferenças são mais acentuadas. A redução I-1 se mostrou muito fraca, provavelmente devido ao seu reduzido tamanho – menos que 1% da vizinhança completa – ao passo que a redução I-2 mostrou melhor desempenho.

O último teste, com as duas melhores vizinhanças acabou obtendo de novo o melhor desempenho: 9,2% de melhoria média.

3.12 Resultados computacionais: desempenho das heurísticas

Nesta seção, fazemos uma avaliação do desempenho do AM implementado comparando-o com todos os outros métodos. O método utilizado como base da avaliação é o mesmo do utilizado na redução de vizinhança: MS com vizinhança de troca *all-pairs*.

Os testes incluem os três esquemas de recombinação apresentados na seção 3.4, a estrutura populacional em árvore, os parâmetros obtidos na seção 3.9 e a redução de vizinhança T-2/I-2 para o AM.

Tempos de processamento e tempos de preparação foram gerados segundo uma distribuição discreta uniforme no intervalo [1, 100]. Datas de entrega foram geradas segundo os parâmetros τ e R. O fator τ indica o percentual de tarefas que se acredita serão entregues com atraso. Com esse valor define-se a média das datas de entrega.

$$d_m = (1 - \tau).n.p_m \tag{13}$$

onde p_m é o tempo de processamento médio das tarefas.

Em seguida, definimos o tamanho do intervalo – Δd – em que serão geradas as datas de entrega. Esse tamanho é dependente do fator R, que indica se as datas de entrega estão concentradas ou espalhadas ao longo do tempo de produção e é definido através da equação:

$$\Delta d = R.n.p_m \tag{14}$$

Com base nos valores de d_m e Δd geramos as datas de entrega segundo a distribuição discreta uniforme:

$$\mathrm{DU}\left(d_m - \frac{\Delta d}{2}, d_m + \frac{\Delta d}{2}\right) \tag{15}$$

Note que as equações anteriores, que definem os parâmetros τ e R, são diferentes das utilizadas na regra ATCS. Assim, com base nos valores das datas de entrega geradas a

partir das equações 13, 14 e 15, recalculamos os valores de τ e de R que serão utilizados pelo ATCS. Estes valores são diferentes, pois a regra ATCS utiliza o estimador de *makespan* C_{max}, o que não é feito nas Equações 13, 14 e 15.

Foram escolhidos três valores para τ e R: 0,2, 0,6 e 1,0. Combinando estes valores temos nove possibilidades de configuração das datas de entrega, com datas de entrega e intervalos "apertados", médios ou "frouxos". Assim, acreditamos poder fornecer uma melhor descrição do desempenho dos algoritmos quaisquer que sejam as características da instância a ser resolvida.

Nesta parte do trabalho foi utilizado um tempo de CPU de 2 minutos para cada método, com exceção das regras ATCS e EDD_{ins}, cujos tempos não ultrapassam dois segundos de processamento.

O algoritmo de comparação é o MS com troca *all-pairs*. Cada valor nas tabelas a seguir apresenta a melhoria percentual média para um conjunto de 5 instâncias. O desempenho do algoritmo avaliado é calculado pela equação:

$$perf_{MA/GA/ATCS/EDDins} = \left(1 - \frac{T_{MA/GA/ATCS/EDDins}}{T_{MS}}\right) \cdot 100 \qquad (16)$$

onde $perf_{MA/GA/ATCS/EDDins}$ é a desempenho relativo do método MA/GA/ATCS/EDDins. $T_{MA/GA/ATCS/EDDins}$ é o atraso total obtido pelo MA/GA/ATCS/EDDins e T_{MS} é o atraso total do MS.

A seguir apresentamos um conjunto de 10 tabelas. Nove são tabelas que apresentam os resultados para cada par $(\tau; R)$ e a última é um resumo dos resultados médios.

Notação:

- X : *Crossover* OX
- RRX : *Crossover* proposto em Rubin and Ragatz [28]
- RRX/OX : *Crossover* híbrido.
- ATCS : Regra proposta em Lee *et al.* [19]

Algoritmo Genético/Memético x ATCS x EDD_{ins}

Tabela 16: (τ; R) = (0,2; 0,2)

	Algoritmo Genético			Algoritmo Memético				
n	**RRX**	**OX**	**RRX/OX**	**RRX**	**OX**	**RRX/OX**	**ATCS**	**EDD_{ins}**
20	-28,3	4,8	5,0	3,9	8,6	6,7	-80,0	-110,0
40	-20,5	12,1	-3,6	5,4	32,4	23,9	-23,8	-33,4
60	-62,3	3,1	-7,2	21,9	42,9	36,4	-10,5	-14,5
80	-70,1	-0,5	-15,4	18,9	38,2	30,3	-3,0	-1,5
100	-112,1	-10,2	-32,7	19,9	42,0	40,0	3,3	5,7
Média	-58,6	1,9	-10,8	14,0	32,8	27,5	-22,8	-31,7

Tabela 17: (τ; R) = (0,2; 0,6)

	Algoritmo Genético			Algoritmo Memético				
n	**RRX**	**OX**	**RRX/OX**	**RRX**	**OX**	**RRX/OX**	**ATCS**	**EDD_{ins}**
20	-45,2	2,4	-22,0	13,0	13,2	13,2	-112,1	-307,9
40	-37,8	3,3	6,8	26,6	54,2	50,5	-70,2	-76,2
60	-60,4	-14,9	-20,7	35,7	86,0	78,6	-40,6	-67,7
80	-92,6	-22,2	-74,4	36,4	82,5	74,7	-15,9	-15,1
100	-123,7	-36,5	-98,8	23,1	60,2	54,6	-9,8	0,3
Média	-71,9	-13,6	-41,8	26,9	59,2	54,3	-49,7	-93,3

Tabela 18: (τ; R) = (0,2; 1,0)

	Algoritmo Genético			Algoritmo Memético				
n	**RRX**	**OX**	**RRX/OX**	**RRX**	**OX**	**RRX/OX**	**ATCS**	**EDD_{ins}**
20	-42,1	-23,0	-30,2	9,9	12,6	10,0	-95,1	-251,9
40	-88,9	-10,1	-44,2	16,0	67,5	62,6	-111,2	-181,1
60	-143,7	-47,7	-66,1	65,8	100,0*	96,1	-137,6	-208,9
80	-168,1	-119,2	-140,8	45,6	88,7	86,6	-25,2	-100,1
100	-203,4	-170,5	-196,3	52,4	86,7	80,7	10,2	-65,0
Média	-129,2	-74,1	-95,5	37,9	71,1	67,2	-71,8	-161,4

* A melhoria de 100% indica que o AM encontrou um atraso total igual a zero em todas as 5 instâncias, ao passo que o MS não obteve nenhum valor igual a zero.

Tabela 19: (τ; R) = (0,6; 0,2)

	Algoritmo Genético			Algoritmo Memético				
n	**RRX**	**OX**	**RRX/OX**	**RRX**	**OX**	**RRX/OX**	**ATCS**	**EDD_{ins}**
20	-8,8	-3,9	-3,6	1,7	1,7	1,7	-20,5	-40,4
40	-11,2	-5,4	-2,0	3,2	16,2	14,1	-13,1	-15,5
60	-16,7	-3,2	-9,6	5,7	14,7	13,1	-1,1	-3,8
80	-23,6	-1,9	-10,7	4,0	12,8	9,4	4,2	2,4
100	-27,2	-3,3	-16,1	5,3	13,1	11,0	5,3	3,2
Média	-17,5	-3,5	-8,4	4,0	11,7	10,4	-5,0	-10,8

Tabela 20: (τ; R) = (0,6; 0,6)

	Algoritmo Genético			Algoritmo Memético				
n	**RRX**	**OX**	**RRX/OX**	**RRX**	**OX**	**RRX/OX**	**ATCS**	**EDD_{ins}**
20	-10,1	-2,7	-3,3	1,2	2,1	1,8	-29,2	-20,3
40	-15,4	-3,0	-5,4	3,4	18,5	15,6	-9,8	-9,9
60	-16,6	-7,0	-7,2	4,3	15,6	15,0	-4,2	-4,5
80	-18,7	-6,3	-12,5	5,8	16,2	15,8	2,3	2,7
100	-22,6	-8,6	-17,5	5,3	15,5	12,2	5,3	4,9
Média	-16,7	-5,5	-9,2	4,0	13,6	12,1	-7,1	-5,4

Tabela 21: $(\tau; R) = (0,6; 1,0)$

	Algoritmo Genético			Algoritmo Memético				
n	**RRX**	**OX**	**RRX/OX**	**RRX**	**OX**	**RRX/OX**	**ATCS**	**EDD_{ins}**
20	-8,9	1,0	-2,2	0,5	0,5	0,5	-23,9	-32,1
40	-19,7	-3,6	-4,7	3,5	15,5	15,5	-14,7	-11,9
60	-18,7	-3,7	-8,8	4,1	16,7	15,6	-2,7	-6,7
80	-30,3	-8,9	-10,3	3,8	18,4	14,0	-3,4	-4,1
100	-39,3	-12,9	-27,3	2,7	17,5	11,0	1,0	1,2
Média	-23,3	-5,6	-10,6	2,9	13,7	11,3	-8,7	-10,7

Tabela 22: $(\tau; R) = (1,0; 0,2)$

	Algoritmo Genético			Algoritmo Memético				
n	**RRX**	**OX**	**RRX/OX**	**RRX**	**OX**	**RRX/OX**	**ATCS**	**EDD_{ins}**
20	-2,5	-1,3	-3,2	0,2	0,2	0,2	-8,3	-15,9
40	-10,7	-2,9	-4,8	1,7	6,2	5,9	-3,5	-8,1
60	-12,4	-2,0	-3,1	2,0	7,3	6,4	3,3	-1,4
80	-15,2	-4,2	-5,4	1,4	6,8	5,0	4,1	-0,8
100	-16,4	-3,9	-9,8	1,7	6,2	5,2	4,7	1,0
Média	-11,4	-2,9	-5,2	1,4	5,3	4,5	-0,5	-5,0

Tabela 23: $(\tau; R) = (1,0; 0,6)$

	Algoritmo Genético			Algoritmo Memético				
n	**RRX**	**OX**	**RRX/OX**	**RRX**	**OX**	**RRX/OX**	**ATCS**	**EDD_{ins}**
20	-6,3	-1,3	-2,1	0,7	0,7	0,7	-9,2	-12,8
40	-7,5	-3,1	-0,9	2,1	4,7	4,8	-4,4	-6,7
60	-7,6	-3,5	-3,3	2,5	7,9	6,8	1,2	-2,0
80	-10,6	-3,2	-5,6	1,3	6,5	5,4	2,6	-2,8
100	-12,7	-5,7	-9,4	1,5	6,2	3,9	3,8	1,2
Média	-8,9	-3,3	-4,3	1,6	5,2	4,3	-1,2	-4,6

Tabela 24: (τ; R) = (1,0; 1,0)

	Algoritmo Genético			Algoritmo Memético				
n	**RRX**	**OX**	**RRX/OX**	**RRX**	**OX**	**RRX/OX**	**ATCS**	**EDD_{ins}**
20	-4,2	-2,5	-2,9	0,1	0,3	0,3	-4,7	-16,2
40	-5,2	-2,9	-2,2	2,6	6,7	6,1	-3,5	-2,5
60	-7,8	-2,1	-3,3	2,5	9,1	7,1	-0,1	-1,3
80	-9,6	-3,1	-6,1	1,8	6,0	4,3	1,5	0,3
100	-12,5	-3,7	-8,3	3,1	5,9	4,8	1,8	-0,1
Média	-7,8	-2,9	-4,5	2,2	5,6	4,7	-0,2	-3,2

Tabela 25: Melhoria média

	Algoritmo Genético			Algoritmo Memético				
(τ; R)	**RRX**	**OX**	**RRX/OX**	**RRX**	**OX**	**RRX/OX**	**ATCS**	**EDD_{ins}**
(0,2; 0,2)	-58,6	1,9	-10,8	14,0	32,8	27,5	-22,8	-31,7
(0,2; 0,6)	-71,9	-13,6	-41,8	26,9	59,2	54,3	-49,7	-93,3
(0,2; 1,0)	-129,2	-74,1	-95,5	37,9	71,1	67,2	-71,8	-161,4
(0,6; 0,2)	-17,5	-3,5	-8,4	4,0	11,7	10,4	-5,0	-10,8
(0,6; 0,6)	-16,7	-5,5	-9,2	4,0	13,6	12,1	-7,1	-5,4
(0,6; 1,0)	-23,3	-5,6	-10,6	2,9	13,7	11,3	-8,7	-10,7
(1,0; 0,2)	-11,4	-2,9	-5,2	1,4	5,3	4,5	-0,5	-5,0
(1,0; 0,6)	-8,9	-3,3	-4,3	1,6	5,2	4,3	-1,2	-4,6
(1,0; 1,0)	-7,8	-2,9	-4,5	2,2	5,6	4,7	-0,2	-3,2
Média	**-31,8**	**-12,1**	**-21,1**	**10,5**	**24,2**	**21,8**	**-18,6**	**-32,7**

Os resultados tanto para o AG quanto para o AM apontam o *crossover* OX como o melhor. O *crossover* RRX teve um desempenho muito fraco, provavelmente devido a uma rápida perda de diversidade ocasionada pela ausência de aleatoriedade no esquema. O AG com OX perdeu em quase todos os casos para o MS, apesar da perda raramente ultrapassar os 10%. Os outros AG se mostraram bem inferiores chegando na média a perder para a regra ATCS.

O AM superou o MS em todos os casos. Para $\tau = 0,2$ os resultados percentuais foram extremamente elevados. Com este valor de τ os atrasos totais são muito próximos de zero, com todas ou quase todas as tarefas sendo entregues no prazo. Valores próximos de zero fazem com que qualquer desvio retorne valores de melhoria ou de piora muito altos, valores esses observados nas Tabelas 16-18.

A regra ATCS teve um desempenho muito bom, ganhando da regra EDD_{ins} em 8 dos 9 casos. Para 80 e 100 tarefas, o ATCS pode rivalizar com o MS e obter resultados superiores ao AG em quase todas as configurações de parâmetros.

Na Tabela 25 é apresentado o desempenho médio de cada método. Em primeiro lugar temos o AM, seguido pelo MS, que foi utilizado como base de comparação. Atrás do MS vêm os algoritmos que obtiveram melhorias médias negativas: o AG (com *crossover* OX), o ATCS e finalmente o EDD_{ins} em último lugar.

CAPÍTULO IV

Robustez dos métodos: AM x ATCS

Neste capítulo, apresentamos uma análise de robustez de desempenho entre dois métodos: AM e ATCS. Normalmente, para se caracterizar um método como robusto deve-se aplicá-lo a um conjunto grande de instâncias aleatórias. Em seguida analisa-se em quantas vezes, ou com qual percentual de melhoria o método foi superior ao concorrente.

Esta análise foi realizada na 3.12 e ficou claro o melhor desempenho do AM frente a todos os outros métodos. Nesta parte do trabalho realizamos um estudo de desempenho mais crítico da robustez do AM e da regra ATCS. Neste caso, o objetivo é encontrar instâncias que sejam boas para um dos métodos e ruins para o outro e vice-versa. Assim, estamos preocupados com comportamentos extremos. Utilizando dados extremos poderemos acrescentar informações complementares às conclusões sobre os comportamentos médios.

Uma busca manual de instâncias boas ou ruins para um determinado problema é uma tarefa impraticável, especialmente para o $1/sds/T_{total}$, e desta forma utilizamos uma abordagem evolutiva para efetuar essa busca. Também descartamos do escopo desta tese análises de pior caso para os algoritmos propostos. O método que propomos consiste de um AG aplicado sobre uma população inicial não-estruturada de instâncias aleatórias. Espera-se assim, ao fim de um certo número de gerações, ter um conjunto de instâncias que sejam fáceis para um método e difíceis para o outro. Para tanto, foi necessária a elaboração de

novas representações genéticas, operadores de recombinação e mutação e de uma função de *fitness*. Esses aspectos são discutidos a seguir.

4.1 Representação das instâncias

O modelo de cromossomo para as instâncias leva em conta todas as características das mesmas. A instância é composta pelos elementos citados no Capítulo I: uma matriz de tempos de preparação e dois vetores – um de datas de entrega e um de tempos de processamento – todos compostos de valores inteiros. Assim, um cromossomo que represente de forma única uma instância deve possuir n^2+2n posições. A forma geral do cromossomo é apresentada na Figura 10.

Instância:

$$[s_{ij}] = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1n} \\ s_{21} & s_{22} & \cdots & s_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n1} & s_{n2} & \cdots & s_{nn} \end{bmatrix} \qquad [d_i] = [d_1, d_2, \ldots, d_n] \qquad [p_i] = [p_1, p_2, \ldots, p_n]$$

Representação genética:

$I = [s_{11}, s_{12}, \ldots, s_{1n}, s_{21}, s_{22}, \ldots, s_{2n}, \ldots, s_{n1}, s_{n2}, \ldots, s_{nn}, d_1, d_2, \ldots, d_n, p_1, p_2, \ldots, p_n]$

Figura 10: Forma geral da representação genética de uma instância do problema de SMS.

No cromossomo são mantidos os valores originais dos tempos de preparação e de processamento, bem como das datas de entrega. Assim, o vetor I é composto por valores inteiros e nos intervalos permitidos. Decidimos manter os elementos s_{ii} na representação genética. No entanto, esses elementos permanecem "inertes" tanto na etapa de recombinação quanto na de mutação.

4.2 Recombinação

Dadas as características especiais da representação genética para as instâncias, foi necessária a elaboração de um esquema de recombinação particular. A principal diferença é no valor do alelo em cada posição do cromossomo. Enquanto que no caso de uma solução do problema de SMS temos valores inteiros que representam as tarefas, no caso de uma instância temos valores inteiros que representam tempos de preparação, tempos de processamento e datas de entrega de tarefas.

Optamos por efetuar a recombinação e a mutação apenas entre os alelos que representam os tempos de preparação das tarefas. Como o estudo é realizado para cada par (τ, R), não faz sentido alterar os valores das datas de entrega e dos tempos de processamento ao longo da evolução das instâncias. Os valores das datas de entrega são gerados a partir dos valores de τ, R e dos tempos de processamento. Assim, os únicos elementos que podem ser modificados sem descaracterizar muito a instância são os tempos de preparação entre tarefas.

A recombinação pode ser dividida em duas etapas. A primeira escolhe uma das instâncias-pai e copia os vetores de datas de entrega e de tempos de processamento dela para o filho. Em seguida o cromossomo do filho é completado varrendo-se as posições dos tempos de preparação em ambos os pais e gerando novos tempos com base nos seus valores. Os alelos na mesma posição em ambos os pais são utilizados para gerar o alelo correspondente no filho (ver Figura 11).

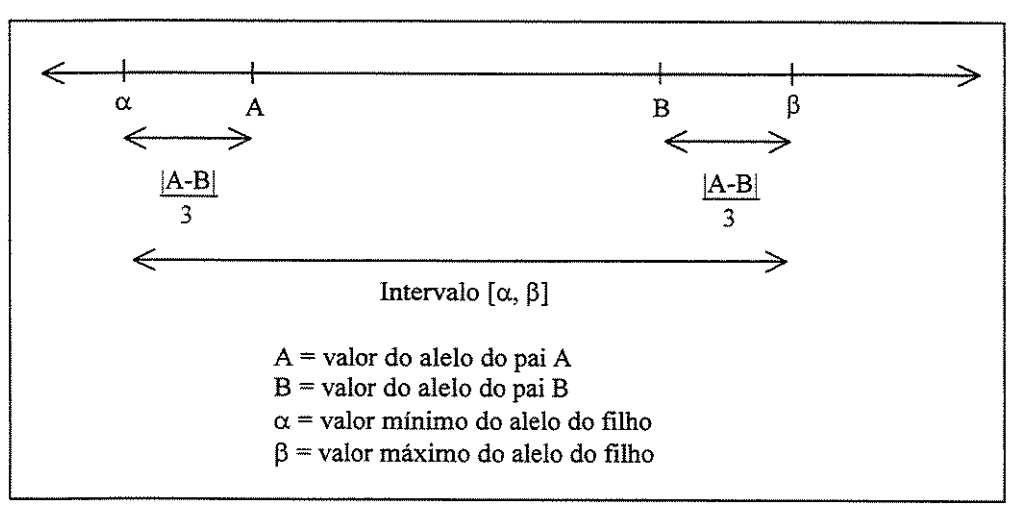


Figura 11: Esquema de geração do novo valor para o alelo do filho.

Sejam A e B os valores dos alelos na mesma posição em ambos os pais. O alelo do filho será gerado segundo uma distribuição DU(α, β). O intervalo (α, β) é centrado no ponto médio entre A e B e seu comprimento é equivalente à diferença A-B em módulo, somada de 2/3 da mesma. Se o valor gerado ultrapassar os limites permitidos, isto é, se for menor que zero ou maior que 100, ele é corrigido para o limite mais próximo.

O esquema tende a preservar bons valores, uma vez que a chance de se gerar um valor dentro do intervalo [A, B] é maior que a de se gerar fora desse intervalo. No entanto, a possibilidade de se gerar um valor fora do intervalo dos pais adiciona uma certa dose de flexibilidade e diversidade ao método.

O método é extremamente aleatório. Analisando alelo a alelo, os novos indivíduos diferem em muito de seus pais pois raramente será gerado um valor de tempo de preparação igual ao de um deles, salvo se os valores de A e B forem muito próximos. Mesmo assim, a estrutura geral dos pais é herdada e houve a convergência da população para indivíduos de alta qualidade.

4.3 Mutação

A mutação é aplicada após a etapa de recombinação somente e cada nova instância tem uma probabilidade *taxa_mut* de sofrer mutação. Uma vez que uma dada instância foi selecionada para mutação, escolhe-se aleatoriamente *n* posições da porção do cromossomo referente aos tempos de preparação das tarefas. Esses *n* valores são substituídos por valores gerados segundo uma distribuição DU[0, 100]. Determinamos que *n* valores deveriam ser modificados a cada mutação pois em uma solução há n^2-*n* valores representativos. A modificação de apenas 1 ou 2 valores por vez adicionaria pouca diversidade na população.

4.4 Função de *fitness*

Como o objetivo desta parte é encontrar instâncias fáceis para o AM e difíceis para a regra ATCS, e vice-versa, uma função de *fitness* razoável deveria levar em conta as soluções retornadas por estes métodos para a mesma instância. A diferença de resultado

entre elas fornece uma estimativa da dificuldade da instância para ambas as estratégias. No nosso caso, a função de *fitness* leva em conta não a diferença absoluta, mas a diferença percentual entre os valores de atraso total fornecidos pelo AM e pelo ATCS. A função de *fitness* escolhida para encontrar instâncias que fossem fáceis para os AM e difíceis para o ATCS é:

$$f_{i_{AM}} = \left(\frac{T_{i_{ATCS}} - T_{i_{AM}}}{T_{i_{ATCS}}} \right) \cdot 100 \qquad (17)$$

Onde f_{iAM} é o *fitness* do indivíduo *i* (da instância *i*) e T_i é o atraso total da instância *i*, calculado via AM ou ATCS. Vamos analisar algumas características da Equação 17. Se $T_{ATCS} = T_{AM}$ então $fitness_i = 0$. Se $T_{AM} < T_{ATCS}$ então $f_i > 0$ e quanto maior a diferença percentual entre ambos, maior o *fitness*, até o valor máximo de 100%, ou seja teríamos gerado uma instância que é muito mais fácil para o AM que para o ATCS. Por fim, se $T_{ATCS} < T_{AM}$ então $f_i < 0$, o que indica uma instância de característica oposta. Casos onde $T_{ATCS} = 0$ são prontamente eliminados devido à inconsistência matemática. Isto não enfraquece a função, pois nos raros casos onde isso ocorreu, verificou-se que T_{AM} também valia zero. Analogamente, a função de *fitness* escolhida para encontrar instâncias que fossem fáceis para o ATCS e difíceis para os AM é:

$$f_{i_{ATCS}} = \left(\frac{T_{i_{AM}} - T_{i_{ATCS}}}{T_{i_{AM}}} \right) \cdot 100 \qquad (18)$$

A Equação 18 é análoga à Equação 17 e as considerações são as mesmas para ambas. No entanto, ressaltamos que nos testes, quando T_{iAM} vale zero, raramente T_{iATCS} também vale zero. Instâncias com tais características são prontamente eliminadas da população pois além da inconsistência matemática, representam problemas muito difíceis para o ATCS e fáceis para o AM.

De acordo com as Equações 17 e 18, a adaptabilidade de uma determinada solução pode ser negativa, o que gera problemas na etapa de seleção para recombinação. Assim, antes da etapa de recombinação, se houver algum *fitness* negativo, todos os valores são

atualizados determinando-se o mais negativo e somando-se esse valor ao *fitness* de todos os indivíduos da população, sem exceção. Desta forma, o pior indivíduo terá adaptabilidade igual a zero e todos os outros serão positivos.

4.5 Seleção para recombinação

A seleção para recombinação segue o mesmo procedimento de *Roulette Wheel* descrito na seção 3.7. Não houve a necessidade da aplicação de uma estrutura na população de instâncias, uma vez que a população não-estruturada com seleção tipo "roleta" obteve resultados satisfatórios. Também não foi imposta nenhuma restrição ao número de recombinações em que um indivíduo pode tomar parte em uma mesma geração, bem como a seleção arbitrária do melhor indivíduo para uma certa porcentagem de recombinações.

Mesmo para instâncias bem pequenas, de 20 tarefas, houve poucos indivíduos com a mesma adaptabilidade ao fim dos testes. Vale ressaltar que indivíduos com valores de adaptabilidade iguais podem ser diferentes, uma vez que apenas n dos n^2 tempos de preparação são realmente utilizados no cálculo do atraso total das soluções do AM e do ATCS.

4.6 Inserção de novos indivíduos

A inserção de novos indivíduos seguiu os mesmos procedimentos descritos na seção 3.8. Todos os filhos obtidos ao longo da geração são copiados para uma população intermediária. A população principal é então ordenada e os piores indivíduos são substituídos pelos novos. Como o esquema de *crossover* apresenta características bastante aleatórias, não foi necessária a elaboração de procedimentos para evitar perda de diversidade, como proibição de indivíduos duplicados. As populações finais, mesmo após muitas gerações, possuem alto grau de diversidade.

4.7 Ajuste de parâmetros

As taxas de recombinação e de mutação para o AG das instâncias foram fixadas em 0,4 e 0,8, respectivamente, baseado na tabela 5. Para o AM que é executado no cálculo da adaptabilidade de cada indivíduo, também seguiu-se a tabela 5. O tamanho da população de instâncias foi fixado em 20 indivíduos. Populações maiores precisariam de muito tempo

computacional para efetuar um número razoável de gerações, visto que a avaliação de cada indivíduo envolve uma rodada de AM e uma de ATCS. Além disso, mesmo com apenas 20 indivíduos foi possível obter um quadro geral do comportamento de cada algoritmo, que é o objetivo desta etapa do trabalho.

4.8 Resultados computacionais: Robustez dos métodos AM x ATCS

Nesta seção, apresentamos os resultados dos testes de robustez do AM e do ATCS. O AM utilizado nesta etapa é o mesmo utilizado nos resultados do Capítulo III. Ele inclui a melhor redução de vizinhança – T-2/I-2 – e utiliza o *crossover* OX. Devido ao esforço computacional requerido para evoluir as instâncias, decidimos reduzir o tempo do AM utilizado na avaliação dos indivíduos nos seguintes termos:

Tabela 26: Tempo de CPU para o AM utilizado no cálculo do *fitness* das instâncias.

n	Tempo de CPU (seg)
20	20
40	30
60	40
80	60
100	90

O tempo de CPU utilizado como limite para a evolução da população de instâncias foi fixado em 4 horas. Esse valor foi tomado como base para se obter um mínimo 20 gerações no AG das instâncias com 100 tarefas e um máximo 180 gerações no AG das instâncias com 20 tarefas.

Cabe ressaltar que se multiplicarmos o tempo de CPU pelo número de indivíduos da população e pelo número de gerações obteremos um tempo total bem superior a duas horas. Por exemplo, para o caso de 100 tarefas teremos: 90.20.20 = 36000 seg = 10 horas. No entanto a avaliação do indivíduo é feita apenas uma única vez – quando este é gerado – e o número de indivíduos gerados é função da taxa de recombinação, que foi fixada em 40% da

população. A seguir apresentamos 5 tabelas – Tabelas 27 a 31 – com os resultados dos testes. Os valores apresentados foram obtidos com base nas Equações 17 e 18.

Notação:

f_{AM}: Melhoria percentual do AM sobre o ATCS – instâncias difíceis para o ATCS

f_{ATCS}: Melhoria percentual do ATCS sobre o AM – instâncias difíceis para o AM

Max: Melhoria percentual máxima – *fitness* do melhor indivíduo da população final

Média: Melhoria percentual média – *fitness* médio da população final

Tabela 27: 20 tarefas

τ / R	Equação	0,2		0,6		1,0	
	f_{AM} / f_{ATCS}	Max	Média	Max	Média	Max	Média
0,2	f_{AM}	100,0*	92,1	100,0*	94,4	100,0*	91,9
	f_{ATCS}	-2,2	-6,3	-1,1	-5,4	0,0	0,0
0,6	f_{AM}	43,9	35,0	53,4	42,1	52,0	44,5
	f_{ATCS}	0,0	-4,6	-0,1	-4,4	-0,1	-4,7
1,0	f_{AM}	24,3	20,3	21,3	17,8	20,7	17,3
	f_{ATCS}	0,2	0,1	0,2	-2,7	1,5	-1,9

Tabela 28: 40 tarefas

τ / R	Equação	0,2		0,6		1,0	
	f_{AM} / f_{ATCS}	Max	Média	Max	Média	Max	Média
0,2	f_{AM}	71,0	60,1	100,0*	93,2	100,0*	96,4
	f_{ATCS}	-7,6	-21,1	-12,0	-34,7	-3,2	-8,5
0,6	f_{AM}	40,6	32,4	40,0	30,3	43,5	32,2
	f_{ATCS}	0,2	-7,8	-0,3	-8,8	0,6	-5,1
1,0	f_{AM}	18,2	11,7	15,7	11,2	16,1	10,4
	f_{ATCS}	1,8	-2,2	2,3	0,0	-0,7	-3,2

Tabela 29: 60 tarefas

τ / R	Equação	0,2		0,6		1,0	
	f_{AM} / f_{ATCS}	Max	Média	Max	Média	Max	Média
0,2	f_{AM}	65,4	54,3	100,0*	97,7	100,0*	97,2
	f_{ATCS}	-15,0	-30,4	0,0	0,0	0,0	0,0
0,6	f_{AM}	35,0	27,6	37,9	30,7	31,6	23,9
	f_{ATCS}	-2,8	-9,7	-0,4	-7,0	-0,8	-8,1
1,0	f_{AM}	13,8	9,7	12,8	9,6	13,1	10,3
	f_{ATCS}	0,9	-2,5	1,0	-2,1	1,2	-2,8

Tabela 30: 80 tarefas

τ / R	Equação	0,2		0,6		1,0	
	f_{AM} / f_{ATCS}	Max	Média	Max	Média	Max	Média
0,2	f_{AM}	64,5	52,6	100,0*	95,7	100,0*	96,7
	f_{ATCS}	-17,3	-24,7	0,0	0,0	0,0	0,0
0,6	f_{AM}	26,6	22,7	30,2	20,9	28,1	21,0
	f_{ATCS}	-3,6	-8,1	-2,2	-8,7	-0,8	-5,1
1,0	f_{AM}	11,4	8,0	13,8	8,1	13,7	10,5
	f_{ATCS}	2,7	-1,0	2,6	-2,3	1,5	-0,8

Tabela 31: 100 tarefas

τ / R	Equação	0,2		0,6		1,0	
	f_{AM} / f_{ATCS}	Max	Média	Max	Média	Max	Média
0,2	f_{AM}	68,6	54,3	100,0*	96,2	100,0*	92,7
	f_{ATCS}	-10,4	-29,8	0,0	-1,9	0,0	0,0
0,6	f_{AM}	27,6	19,0	29,4	21,4	28,4	21,1
	f_{ATCS}	1,0	-6,5	-2,6	-7,8	-1,4	-6,7
1,0	f_{AM}	10,4	6,6	7,7	5,6	8,6	6,8
	f_{ATCS}	3,2	-0,7	3,3	-1,4	1,5	-1,6

Tabela 32: Média dos resultados em *n*

τ / R	Equação	0,2		0,6		1,0	
	f_{AM} / f_{ATCS}	Max	Média	Max	Média	Max	Média
0,2	f_{AM}	73,9	62,7	100,0*	95,4	100,0*	95,0
	f_{ATCS}	-10,5	-22,5	-2,6	-8,4	-0,6	-1,7
0,6	f_{AM}	34,7	27,3	38,2	29,1	36,7	28,5
	f_{ATCS}	-1,0	-7,3	-1,1	-7,3	-0,5	-5,9
1,0	f_{AM}	15,6	11,3	14,3	10,5	14,4	11,1
	f_{ATCS}	1,8	-1,3	1,9	-1,7	1,0	-2,1

* A melhoria de 100% indica que o AG encontrou uma instância onde o AM retornou um atraso total zero e o ATCS não.

Com base nas Tabelas 27-32 notamos uma clara vantagem do AM sobre o ATCS. A diferença de desempenho depende fundamentalmente do número de tarefas e do tipo de instância – mais do parâmetro τ que do *R*.

Para o AM foram obtidos valores sempre melhores que o ATCS, inclusive nas médias da população final. No caso do ATCS temos uma instabilidade bem maior, com valores máximos e médios oscilando bastante entre positivos e negativos. A grande quantidade de valores médios negativos é um outro indicativo da dificuldade do ATCS para superar o AM.

Além disso, quando o ATCS supera o AM, as porcentagens de melhoria são bem baixas, não superiores a 3,5%, mesmo levando-se em conta que na Fase 3 do ATCS é realizada uma busca local com vizinhança de troca *all-pairs* e Inserção completas, ao passo que no AM a busca local é feita sobre vizinhanças reduzidas. Na Tabela 32, que traz as médias dos resultados, somente quando τ = 1,0 o ATCS tem um desempenho melhor que o AM, mesmo assim somente na coluna dos valores máximos.

No entanto, convém lembrar que os tempos computacionais do ATCS são ínfimos quando comparados aos do AM.

CAPÍTULO V

Análise de *fitness landscape*

Estudos de *fitness landscape* tiveram início em 1932 com o trabalho de Sewall Wright [32] para visualização da adaptabilidade de espécies biológicas ao meio ambiente. A principal finalidade desses estudos no nosso caso consiste em avaliar se os operadores de *crossover* e de mutação estão bem relacionados com a estrutura do problema. Traduzimos por estrutura do problema, neste caso, a representação genética e o espaço de estado gerado por ela. Esta técnica foi utilizada em outros trabalhos, como Boese [2] para o problema do Caixeiro Viajante. Podemos citar também os trabalhos de Reeves [27] em *flow-shop* e Merz e Freisleben [20] em Bipartição de Grafos.

O problema de SMS pode ser caracterizado como uma busca em um espaço de estado onde cada permutação das tarefas, ou solução válida, é considerado um estado. Operadores de *crossover* e de mutação bem elaborados devem fazer com que a população inicial convirja, em termos de uma medida de distância, para a solução ótima. Se não houver convergência, ou se a convergência for ocasional, os operadores devem ser reestudados.

Para se avaliar a convergência da população para a solução ótima é necessário definir uma métrica para a distância, e nesta parte do trabalho adotamos duas.

5.1 Distância de Hamming

A distância de Hamming (DH) é normalmente utilizada para medir a distância entre sequências de valores 0 e 1, isto é, sequências de bytes. No SMS, como uma solução é definida por uma sequência de valores inteiros, adaptamos o conceito para o novo caso. Assim, a distância de Hamming adotada mede o número de posições em que os alelos das soluções diferem de valor. Veja a Tabela 33.

Tabela 33: Exemplo de DH aplicada a duas soluções de SMS

Posição	1	2	3	4	5	6	7	8	9	10
Solução A	4	3	7	9	2	1	10	8	6	5
Solução B	7	3	1	9	2	6	10	8	4	5
Alelos Diferentes	Sim	Não	Sim	Não	Não	Sim	Não	Não	Sim	Não
Distância de Hamming	4									

5.2 Distância Acumulada

A distância Acumulada (DA) possui maior afinidade com o problema que a DH. Neste caso é avaliado não apenas se os alelos das soluções em uma determinada posição são diferentes, mas também quão distantes estão um do outro. Na Tabela 34 exibimos um exemplo do cálculo da DA com as mesmas soluções A e B da Tabela 33.

Tabela 34: Exemplo de DA aplicada à duas soluções de SMS

Posição	1	2	3	4	5	6	7	8	9	10
Solução A	4	3	7	9	2	1	10	8	6	5
Solução B	7	3	1	9	2	6	10	8	4	5
Distância	8	0	2	0	0	3	0	0	3	0
Distância Acumulada	16									

Na Tabela 34 a tarefa 4 ocupa a posição 1 na solução A e a posição 9 na solução B, o que resulta em uma distância de 8 para esta tarefa. Ao se varrer toda a sequência, obtém-se a DA de 16. Essa segunda definição de distância é vantajosa para o problema de SMS, pois a informação de quão separadas estão duas tarefas nas soluções reflete melhor o nível de "similaridade" entre elas. É natural supor que duas soluções em que uma das tarefas ocupe, por exemplo, a quinta e a sexta posições, respectivamente, sejam mais semelhantes do que se a tarefa em questão ocupasse a primeira e a última posições. A segunda hipótese indica uma diferença considerável na sequência das operações que convém ser refletida na medida de distância. A DH, ao contrário da DA, consideraria em ambos os casos as soluções como estando à mesma distância uma da outra, pelo menos no que se refere à tarefa analisada.

5.3 Configurações do *landscape*

O *fitness landscape* foi gerado com base nos mínimos locais obtidos ao longo do processo evolutivo de busca. No nosso trabalho, a população inicial e todos os filhos gerados são considerados para esta caracterização. Para tornar a análise mais fácil, normalmente utiliza-se uma visualização gráfica em duas dimensões. Os eixos x e y referem-se à distância para a solução ótima e à diferença da função objetivo em relação ao ótimo, respectivamente. Assim, cada solução é traduzida em um ponto e o *landscape* fica representado por uma nuvem de pontos. A Figura 12 mostra duas configurações relativamente comuns de *landscapes*.

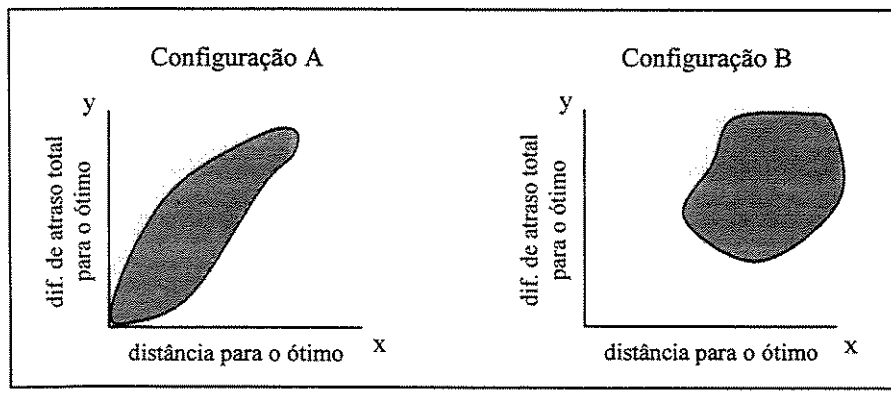


Figura 12: Duas configurações possíveis de *fitness landscape*.

A configuração **A** sugere um "bom" *landscape*, pois temos uma boa correlação entre função objetivo e distância para o ótimo, sendo ambas minimizadas em conjunto ([16], [20]). Ao visualizar as configurações devemos notar que estamos representando um fenômeno dinâmico de uma forma estática. No entanto, podemos extrair algumas informações dinâmicas pela forma que o gráfico assume. Por exemplo, na configuração **A**, é natural deduzir que os mínimos locais obtidos no início do processo evolutivo são de baixa qualidade e estão localizados na parte superior da área hachurada. À medida que o algoritmo evolui, vai-se constituindo a parte intermediária e, por fim, já ao final do processo, a parte mais próxima da origem. Essa parte concentra elementos de alta qualidade, característica da população final do AM.

A configuração B, por sua vez, sugere um comportamente "ruim" pois as soluções estão concentradas em uma região afastada do ponto ótimo. A correlação entre função objetivo e distância para o ótimo é inexistente e a obtenção do ótimo é muito difícil. A evolução pode ser classificada como muito restrita, pois do início ao fim do processo evolutivo os mínimos locais foram gerados sempre em uma mesma região, distante da solução ótima. Isto acontecendo, convém reavaliar os operadores de recombinação e de mutação. Um problema para este tipo de análise é que se torna necessário saber de antemão a solução ótima. Como tal fato não é possível no nosso caso, adotamos a melhor solução encontrada pelo algoritmo como a ótima.

5.4 Resultados computacionais: *fitness landscape*

Os testes computacionais para análise do *fitness landscape* foram feitos com os mesmos parâmetros e vizinhanças de busca local da seção 3.12. Foram analisados em separado dois esquemas de *crossover*: OX e RRX. As instâncias variaram de 20 a 60 tarefas e os parâmetros τ e R foram fixados em (0,2; 0,6), (0,6; 0,6) e (1,0; 0,6).

Os gráficos são apresentados sempre aos pares, considerando as duas métricas para a distância entre soluções: Distância de Hamming e Distância Acumulada. A melhor solução encontrada está localizada na origem e os eixos x e y representam a distância para a melhor solução e a diferença de atraso total, respectivamente, para cada ótimo local encontrado durante a execução do AM. A diferença do atraso total está em termos percentuais, onde o

valor 1 representa a melhor solução encontrada. Desta forma, um valor igual a 2 representaria um atraso duas vezes maior (ou 100% pior) que o menor atraso obtido e assim por diante.

5.4.1 Gráficos para Crossover OX

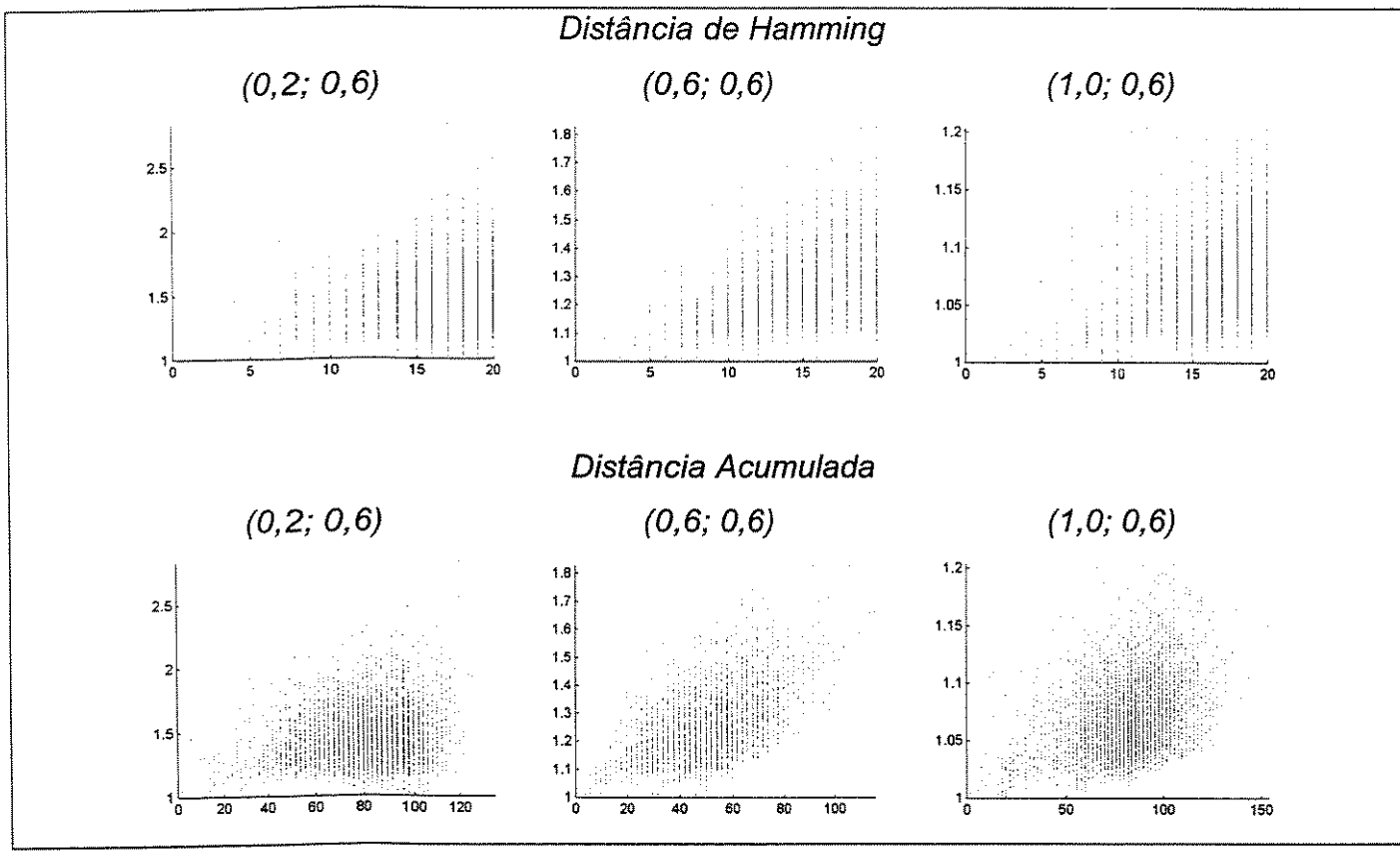


Figura 13: *Fitness landscapes* para 20 tarefas

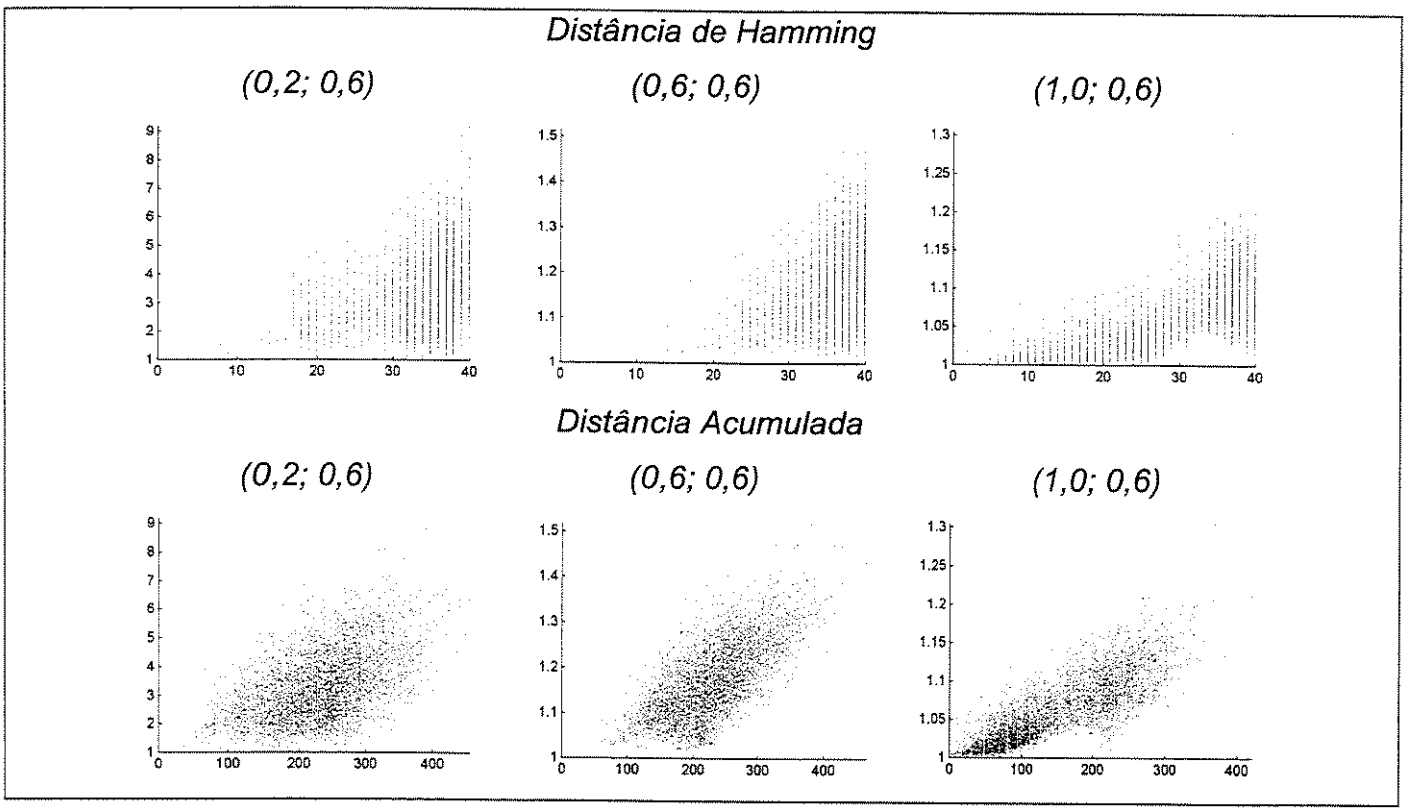


Figura 14: *Fitness landscapes* para 40 tarefas

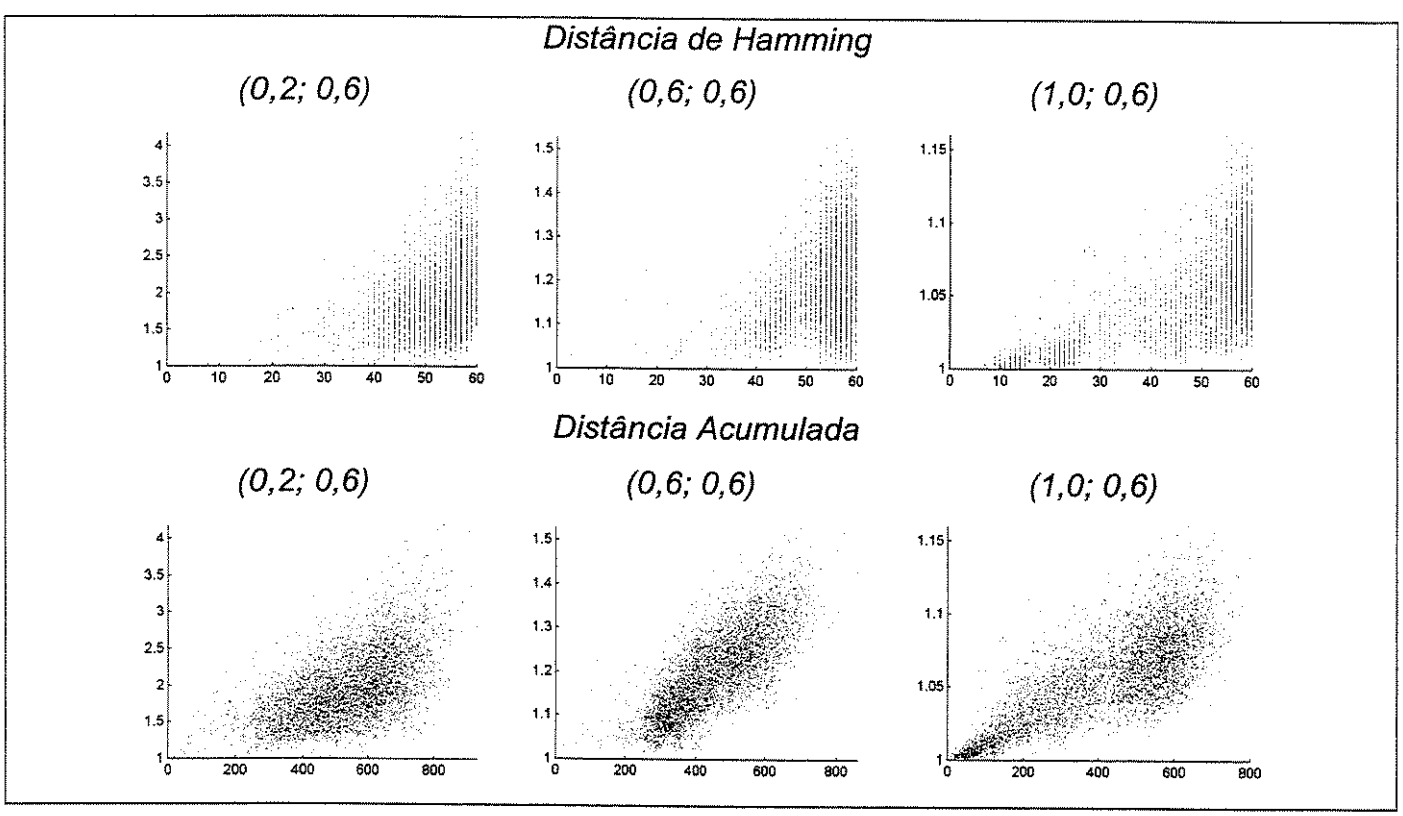


Figura 15: *Fitness landscapes* para 60 tarefas

As Figuras 13-15 refletem o bom desempenho do *crossover* OX. Para 20 tarefas, apesar da correlação não ser evidente, notamos facilmente o caminho percorrido desde as soluções iniciais até o ótimo, em um desenho semelhante à configuração **A** da figura 12. Esta característica fica mais evidenciada com o uso da DH.

Para 40 tarefas a configuração dos ótimos locais se torna melhor, indicando uma convergência suave dos pontos em direção à origem. Para o caso τ = 1,0, nas primeiras gerações os mínimos locais estão localizados a até 25% da melhor solução encontrada. Nas últimas gerações temos os mínimos locais a 5% desta solução; um sinal claro de evolução. Para τ = 0,6 temos uma situação não muito boa, representada pela concentração de mínimos locais em torno da DA de 200. Apesar de existirem vários mínimos a até 100 unidades de distância, essa não é a situação ideal pois gera um desvio na direção de convergência. Outros testes foram rodados para verificar se tal comportamento era uma exceção, mas ele se repetiu.

Finalmente, para 60 tarefas, visualmente nota-se que a correlação aumenta mais, e para τ = 1,0 de novo temos o comportamento ideal, com uma convergência muito acentuada para a região da melhor solução encontrada. A densidade da nuvem de mínimos locais em torno da origem é muito alta.

Analisaremos agora a diferença do atraso total nos diferentes gráficos. Para 20 tarefas temos os mínimos locais com valores no máximo 2,5 vezes piores que o melhor encontrado – no caso *(τ; R) = (0,2; 0,6)*. Para 40 tarefas essa relação piora muito e obtemos mínimos locais até 9 vezes piores que o melhor mínimo local. Essa é uma situação delicada e caracteriza um problema difícil. Já para o caso *(τ; R) = (1,0; 0,6)* temos uma concentração extremamente elevada de mínimos locais abaixo de 5% do menor atraso encontrado. Para 60 tarefas a situação se repete, sendo que para *(τ; R) = (1,0; 0,6)* a concentração maior de mínimos locais está a menos de 1% do menor atraso, o que caracteriza um problema fácil.

5.4.2 Gráficos para Crossover RRX

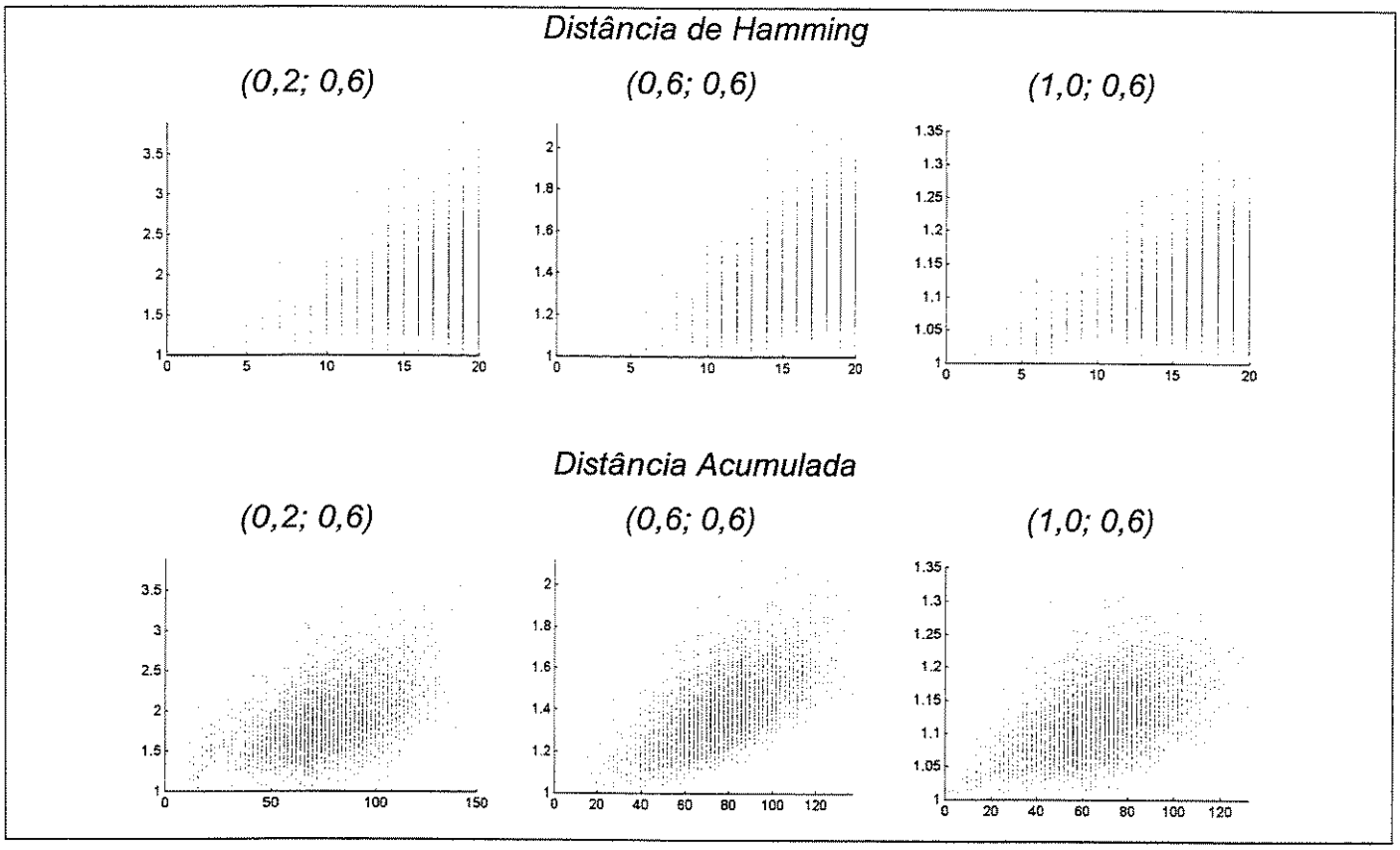


Figura 16: *Fitness landscapes* para 20 tarefas

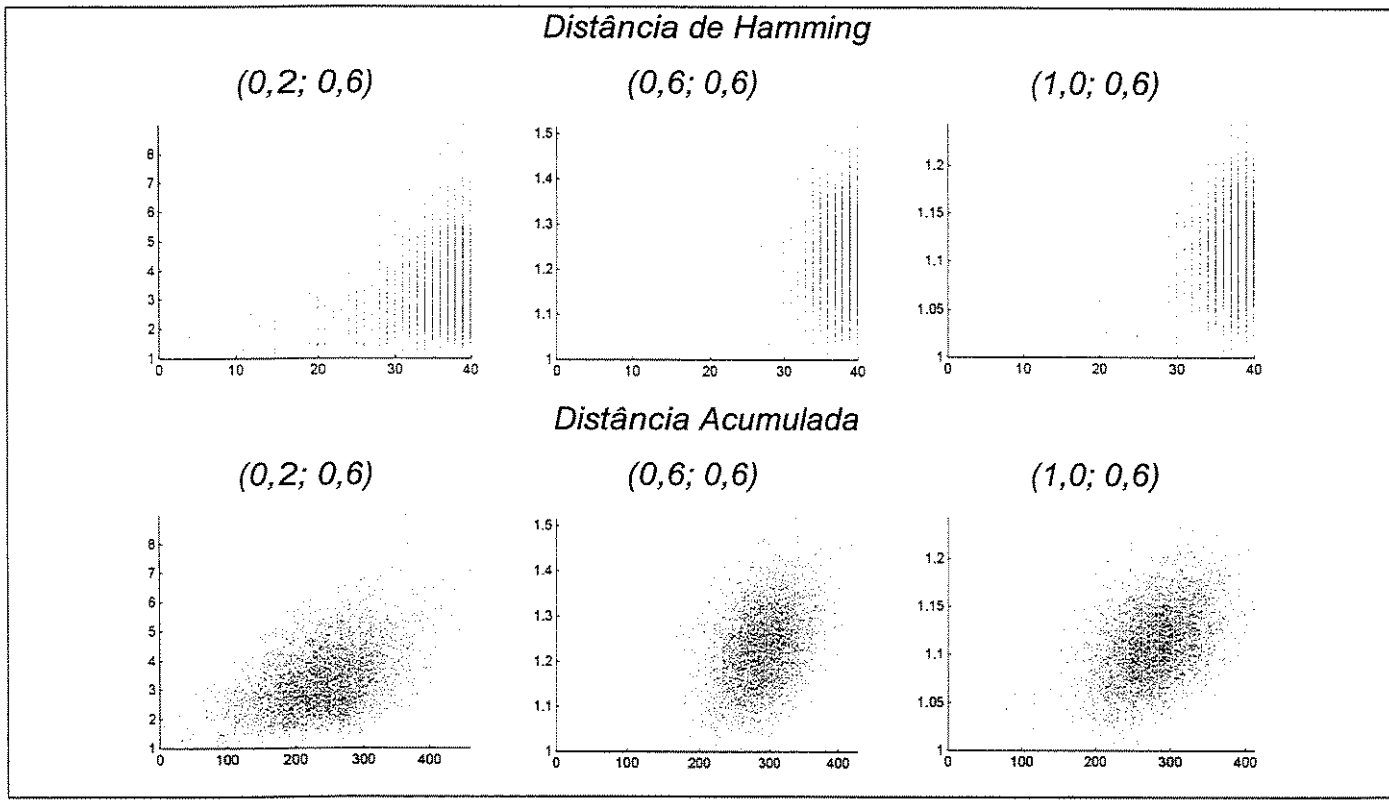


Figura 17: *Fitness landscapes* para 40 tarefas

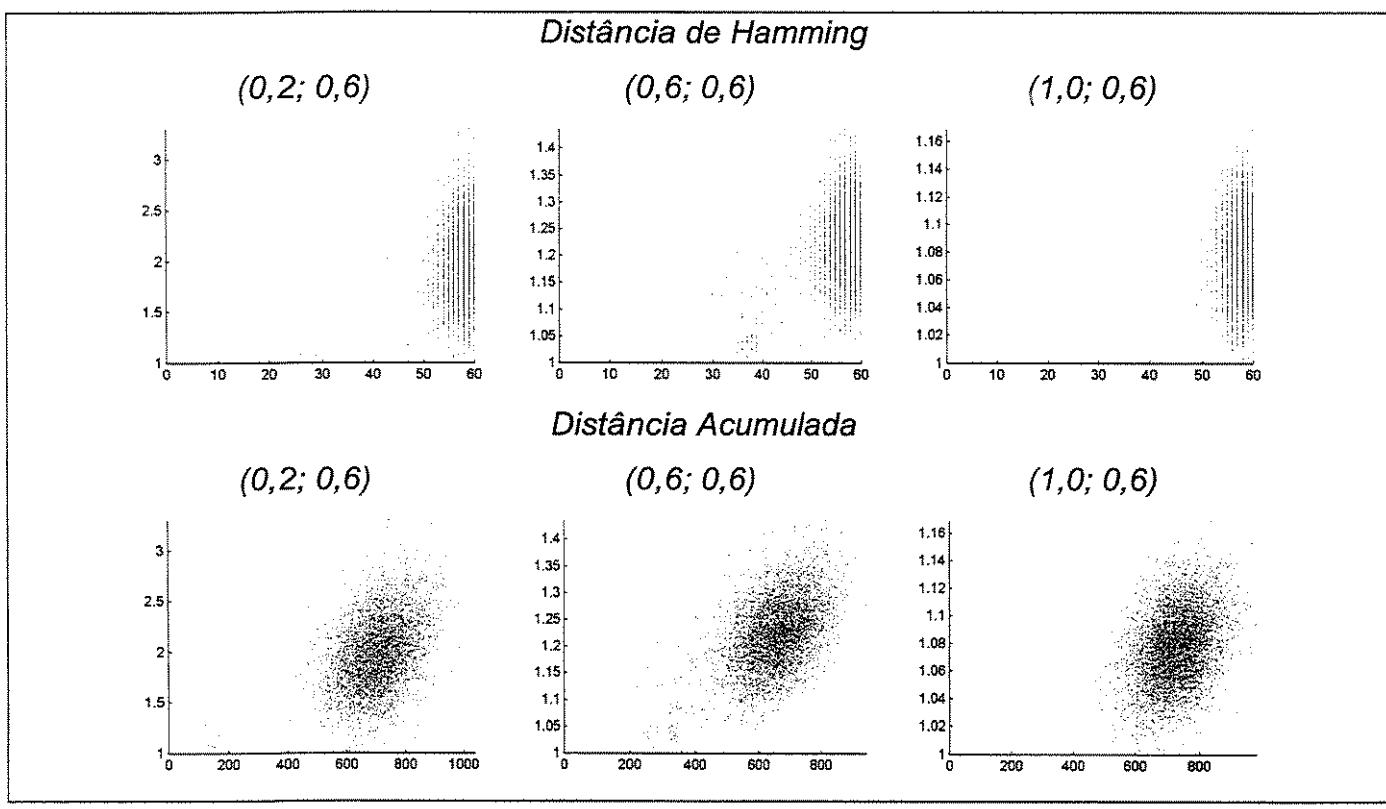


Figura 18: *Fitness landscapes* para 60 tarefas

O *crossover* RRX apresentou configurações de *fitness landscapes* em geral piores que o OX. Para o caso de 20 tarefas o *crossover* mostrou uma boa correlação entre distância e diferença de atraso total. Assim, para este tamanho de instância não podemos concluir diferenças de desempenho com base na simples análise do *landscape*. De fato, para 20 tarefas temos ambos os *crossovers* com comportamento muito semelhante (ver tabela 8).

Já para 40 tarefas a única correlação razoável foi para o caso $(\tau; R) = (0,2; 0,6)$. Para as outras duas combinações de τ e R o *landscape* é muito parecido com a configuração **B** da figura 12. Note que utilizando a DH, a concentração dos mínimos locais se dá a 30 posições ou mais de distância para a melhor solução encontrada. Esse resultado é um indicativo de falta de uniformidade na convergência e conclui-se que a melhor solução foi encontrada por acaso, não havendo outras soluções próximas dela. A concentração de mínimos locais para $(\tau; R) = (1,0; 0,6)$ fica entre 5% e 15% do melhor valor encontrado, ao passo que no OX, para esta configuração, a concentração estava abaixo de 5%.

Para 60 tarefas a situação se torna mais crítica, com ausência de correlação até para o caso $(\tau; R) = (0,2; 0,6)$.

As conclusões obtidas com a análise de *fitness landscape* vão ao encontro dos resultados apresentados na seção 3.12 onde as Tabelas 16 a 25 mostram uma superioridade do *crossover* OX em relação ao *crossover* RRX. Este fato é explicado pela diferença de eficiência na exploração do espaço de soluções.

CAPÍTULO VI

Introdução ao problema de Sequenciamento em Máquinas Paralelas

O problema de Sequenciamento em Máquinas Paralelas (SMP) guarda muita semelhança com o problema de SMS. A diferença se dá, como o próprio nome diz, no número de máquinas envolvidas no processo produtivo. As considerações feitas para o problema de SMS foram estendidas para o problema de SMP: tempos de preparação dependentes da sequência e tempos de processamento. As máquinas são idênticas e a função objetivo a ser minimizada é o *makespan*.

Assim, definimos o problema tratado neste capítulo como:

Entrada: Seja n o número de tarefas a serem processadas em m máquinas.

Seja p um vetor $\{p_1, ..., p_n\}$ de tempos de processamento das tarefas.

Seja $\{s_{ij}\}$ uma matriz de tempos de preparação, onde o elemento s_{ij} é o tempo necessário para preparar a produção da tarefa j depois da máquina terminar o processamento da tarefa i.

Seja $\{so_j\}$ um vetor de tempos de preparação, onde o elemento so_j é o tempo necessário para preparar a produção da tarefa j se esta for a primeira a ser processada na máquina.

Objetivo: Encontrar a permutação das tarefas que minimize o *makespan* do sequenciamento, que é calculado como:

$$makespan = \max_{i}[\text{tempo\_de\_término}_i] \tag{19}$$

$$\text{tempo\_de\_término}_i = so_{j(1)} + p_{j(1)} + \sum_{k=1}^{num\_tarefas_i} s_{j(k),j(k+1)} + p_{j(k+1)} \tag{20}$$

Onde *makespan* é o valor máximo do tempo_de_término envolvendo todas as máquinas. A notação *j(k)* indica a *k*-ésima operação processada na máquina *i* e $num\_tarefas_i$ é o número de tarefas processadas na máquina *i*.

CAPÍTULO VII

AM aplicado ao problema de Sequenciamento em Máquinas Paralelas

Esta parte do trabalho visa avaliar a robustez do AM implementado, no que diz respeito à capacidade de atender a outros problemas sem perda significativa de desempenho e com um mínimo de modificações. Para tanto, alteramos o código do AM original, que resolve o problema de SMS, para fazê-lo resolver o problema de SMP. As comparações são feitas com uma Busca Tabu [11] apresentada em França *et al.* [8]. As partes do AM que foram modificadas estão descritas a seguir.

7.1 Representação genética

A representação escolhida para o problema de SMP foi a mesma utilizada em Cheng and Gen [4]. A solução é representada por um cromossomo cujos alelos assumem valores inteiros e distintos no intervalo [1, *n*]. Além disso, há *m-1* pontos de quebra na sequência, representadas por um asterisco, que definem as sub-sequências associadas às várias máquinas.

Desta forma, um exemplo de solução válida para um problema com 10 tarefas e três máquinas seria < 5 3 4 * 8 9 1 * 2 6 10 7 >. A máquina 1 processa as tarefas 5, 3 e 4, nesta ordem; a máquina 2 as tarefas 8, 9 e 1; finalmente a máquina 3 é responsável pelas tarefas 2, 6, 10 e 7, em sequência.

7.2 Função de *fitness*

A função de fitness foi modificada para que atendesse à nova função objetivo, que considera o cálculo do *makespan*, assumindo a forma:

$$f_i = (makespan)^{-1} \tag{21}$$

Onde f_i é o *fitness* do indivíduo *i*. Como o valor do *makespan* é sempre maior que zero, não há a necessidade de se somar uma unidade antes de efetuar a inversão. O cálculo do *makespan* é feito segundo as Equações 19 e 20.

7.3 Busca Local

A busca local implementada não incluiu nenhum tipo de redução de vizinhança. Para o caso de SMP foram implementadas as duas vizinhanças completas: troca *all-pairs* e Inserção. Assim, dada uma sequência inicial, aplica-se um processo de *hill-climbing* baseado na troca *all-pairs* seguido de um outro baseado em Inserção. Uma vez terminada essa primeira passagem, repete-se o *hill-climbing* até que não se consiga melhorar mais a solução em nenhuma das duas vizinhanças. É um processo extremamente intensificador, mas que para o problema em questão obteve o melhor desempenho. A seguir descrevemos o algoritmo de busca local passo-a-passo.

Passo 1: Aplique a busca local *all-pairs* na solução de partida, até que esta solução atinja um mínimo local.

Passo 2: Aplique a busca local de Inserção na solução obtida no Passo 1, até que esta atinja um mínimo local.

Passo 3: Se a solução de partida do Passo 1 é a mesma do final do Passo 2, PARE. A solução é um mínimo local de ambas as vizinhanças. Caso contrário retorne ao Passo 1, com a solução de partida atualizada como sendo a obtida ao final do Passo 2.

Como este procedimento requer muito tempo computacional, outras estratégias de *hill-climbing* foram testadas, como por exemplo, executar os Passos 1 e 2 apenas uma única vez, ou efetuar uma única passagem em cada uma das vizinhanças não garantindo que a solução seja um mínimo local. Estes procedimentos alternativos se mostraram menos eficientes que o procedimento mais intensificador.

7.4 Ajuste dos parâmetros

O ajuste de parâmetros seguiu o mesmo procedimento descrito na seção 3.9, considerando-se a população estruturada e o $m = 6$. O valor de n é mais determinante para os parâmetros que o valor de m. O tempo de CPU para os testes foi de 2 minutos e os resultados obtidos estão na Tabela 35.

Tabela 35: Ajuste de parâmetros para o AM aplicado ao SMP

Algoritmo Memético			
n	**Número de indivíduos**	**Taxa de *crossover***	**Taxa de mutação**
20	121	0,2	0,1
30	40	0,3	0
40	40	0,2	0,1
50	40	0,2	0,1
60	13	0,3	0
80	13	0,2	0
100	13	0,2	0

Nesta parte do trabalho vemos a semelhança de resultados entre os problemas de SMS e SMP. Tanto o número de indivíduos, quanto as taxas de crossover e de mutação apresentaram resultados muito próximos daqueles obtidos na seção 3.9.

7.5 Resultados computacionais: AM x Busca Tabu

Nesta seção, comparamos o desempenho de duas heurísticas de Busca Tabu – *Fast Tabu* e *Long Tabu* – implementadas em França *et al.* [8] com o AM para este problema. Para problemas pequenos, ambos os métodos puderam ser comparados com soluções ótimas obtidas por uma busca dicotômica. Para as instâncias pequenas, em que o método exato não obteve solução ótima, um bom limitante inferior obtido pela busca dicotômica foi utilizado para comparação quando possível. A última coluna na Tabela 36 mostra o número de instâncias comparadas com a solução ótima (OT) e com o limitante inferior (LI). A medida de desempenho adotada é a porcentagem média de desvio em relação ao ótimo/limitante inferior.

Para problemas maiores, o método exato geralmente falha em encontrar até mesmo um limitante inferior. Assim adotou-se a heurística Long Tabu como base de comparação e os valores são os desvios percentuais dos outros métodos em relação a esta heurística. Foram escolhidos dois tempos computacionais para o AM: 2 minutos (AM_{2min}) e o mesmo tempo computacional do *Fast Tabu* (AM_{igual}).

As instâncias utilizadas nos experimentos computacionais foram geradas aleatoriamente e os intervalos utilizados para n e m foram [20, 100] e [2, 10], respectivamente. Os tempos de processamento seguem uma distribuição DU(1, 100) e os tempos de preparação uma distribuição DU(1, 10). Foram consideradas 10 replicações para cada combinação do número de máquinas e de tarefas. Desta forma, 420 instâncias foram geradas. No entanto, como utilizamos apenas limitantes inferiores e soluções ótimas para avaliação dos resultados para as instâncias consideradas pequenas (m = 2, 3 e 4), 71 instâncias foram descartadas, pois para elas não foi possível determinar nem o *makespan* ótimo nem um limitante inferior.

Tabela 36: Comparações AM x Busca Tabu para instâncias pequenas

n	*m*	Long Tabu	CPU (seg.)	Fast Tabu	CPU (seg.)	AM_{2min}	AM_{igual}	OT/LI
20	2	0,43	304	0,62	13,7	0,16	0,27	10 / 0
30		0,66	2016	0,87	40,6	0,50	0,55	10 / 0
40		0,53	6576	0,70	87,0	0,56	0,61	10 / 0
50		0,42	14248	0,65	165,8	0,52	0,50	9 / 0
60		0,36	12957	0,47	225,8	0,55	0,50	6 / 4
80		0,17	50478	0,45	417,2	0,45	0,41	1 / 4
100		0,17	57823	0,23	1751	0,37	0,36	3 / 0
20	3	1,07	54	1,14	3,6	0,55	1,01	4 / 6
30		0,89	277	0,98	10,5	0,65	1,09	3 / 7
40		0,89	968	1,02	22,9	0,92	1,00	2 / 8
50		0,75	2758	0,95	45,7	0,87	0,94	0 / 7
60		0,64	6613	0,88	65,9	0,76	0,82	2 / 4
80		0,43	23167	0,56	348,6	0,68	0,66	3 / 5
100		*	*	*	*	*	*	0 / 0
20	4	1,76	23	1,88	3,1	1,06	1,61	0 / 8
30		1,35	144	1,35	8,7	1,22	1,49	0 / 2
40		1,44	468	1,56	17,2	1,16	1,51	0 / 3
50		0,99	1216	1,04	35,6	0,99	1,22	0 / 3
60		0,94	3215	1,05	59,3	1,05	1,13	0 / 5
80		0,61	14435	0,81	93,4	0,93	0,98	0 / 4
100		0,45	36833	0,77	197,4	0,83	0,81	0 / 6
Média		**0,71**	**11844**	**0,90**	**180,7**	**0,74**	**0,87**	

Tabela 37: Comparações AM x Busca Tabu para instâncias grandes

n	*m*	Fast Tabu	CPU (seg.)	AM_{2min}	AM_{Igual}
20	6	0,78	2,4	-1,11	-0,61
30		0,39	5,0	-0,43	0,31
40		0,23	11,8	-0,06	0,20
50		0,26	20,1	0,34	0,50
60		0,20	30,1	0,18	0,43
80		0,31	62,4	0,39	0,46
100		0,29	124,8	0,54	0,48
20	8	1,40	1,4	-0,73	-0,59
30		0,77	3,9	-0,51	-0,15
40		0,69	7,2	-0,19	0,31
50		0,34	14,9	0,19	0,53
60		0,29	23,3	0,13	0,34
80		0,33	44,6	0,40	0,54
100		0,30	71,3	0,43	0,56
20	10	0,49	1,0	-1,48	-1,40
30		0,53	4,7	-1,65	-0,99
40		0,96	6,6	-0,20	0,30
50		0,93	10,9	-0,15	0,26
60		0,39	16,5	0,23	0,48
80		0,55	27,5	0,38	0,57
100		0,49	50,9	0,47	0,53
Média		**0,52**	**25,8**	**-0,13**	**0,15**

O desempenho do AM na Tabela 36 está localizado entre o do Long Tabu e o do Fast Tabu. O AM_{2min} obteve resultados melhores que o Long Tabu, especialmente para instâncias com poucas tarefas. Para instâncias com 40 e 50 tarefas os resultados foram semelhantes e para 60 tarefas ou mais, o AM perdeu para o Long Tabu. Quando comparamos o AM_{2min} com o Fast Tabu, o resultado final médio é bem melhor: 0,74% para o AM contra 0,90% para a Busca Tabu. Uma diferença de 0,16 pontos percentuais a favor do AM. A comparação com o

Long Tabu mostra que este ganha do AM por apenas 0,03 pontos percentuais, o que consideramos um desempenho médio muito bom, levando-se em conta a diferença dos tempos computacionais entre as duas abordagens.

Na Tabela 37, o desvio é em relação ao Long Tabu. O Fast Tabu perde em 0,52% para o Long Tabu, ao passo que o AM com mesmo tempo computacional perde por 0,15%, um desempenho bem superior. Muito bom também foi o desempenho médio do AM_{2min}, superando o Long Tabu em 0,13%, cujo tempo computacional médio é de 1042 segundos.

CAPÍTULO VIII

Conclusões

A primeira parte desta tese tratou do problema de Sequenciamento em Máquina Simples (SMS) e nela foram feitas comparações entre uma implementação baseada em Algoritmos Meméticos (AM) e três outros métodos propostos na literatura. Como contribuições mais relevantes nesta parte estão a introdução de uma população hierarquicamente estruturada, o estudo de reduções de vizinhança para duas buscas locais tradicionais e o ajuste automático de parâmetros do AG / AM.

Os resultados desta parte indicam que a população estruturada tem desempenho superior à não-estruturada, especialmente quando o tamanho da instância se torna muito grande. O estudo de reduções de vizinhanças gerou resultados interessantes, onde as vizinhanças reduzidas em geral obtiveram resultados melhores que as vizinhanças completas. Em um AM aplicado ao problema de SMS mostrou-se fundamental uma busca local "leve", de modo que o algoritmo tenha tempo para realizar um número suficiente de gerações. Nesta etapa também introduziu-se um método de ajuste de parâmetros para o AG e para o AM. Este método consiste em uma busca enumerativa parcial das combinações de possíveis valores de tamanho de população, taxa de *crossover* e taxa de mutação. Um resultado interessante foi a taxa de mutação ideal encontrada para o AG, bem mais alta que as normalmente utilizadas em outros trabalhos.

Nesta parte também realizou-se uma comparação entre dois operadores de recombinação: RRX e OX. Os resultados indicaram um desempenho superior do OX em relação ao RRX, apesar deste último ter sido especialmente projetado para o problema de SMS.

A segunda parte do trabalho consistiu de uma análise de robustez do AM. Nesta etapa colocou-se o AM para competir contra a regra ATCS. Ficou demonstrado o melhor desempenho do AM para a ampla gama de problemas testados, o que era de se esperar dado que o AM é bem mais complexo e exige um tempo de CPU bem maior. A contribuição desta parte se concentra na metodologia de busca de instâncias difíceis para um certo método pela aplicação de um AG sobre uma população de instâncias. Com isso pode-se testar novos métodos de forma mais intensiva, encontrado possíveis pontos fracos de maneira rápida e automática.

Na parte seguinte fez-se uma análise de *fitness landscape* visando avaliar de forma mais criteriosa o comportamento dos *crossovers* utilizados neste problema. O resultado veio ao encontro do que se concluiu na primeira parte do trabalho. O *landscape* gerado a partir do *crossover* RRX indica que ele contribui muito pouco na evolução da população. O *crossover* OX por sua vez apresentou um comportamento bem melhor, claramente gerando um grande número de indivíduos de boa qualidade ao final do processo evolutivo.

A última parte desta tese se concentrou no problema de Sequenciamento em Máquinas Paralelas. O AM utilizado para SMS foi modificado em algumas de suas características para atender ao novo problema e o desempenho final foi satisfatório quando comparado com uma Busca Tabu especializada. Com isso, atingiu-se o objetivo de mostrar a flexibilidade do método.

Trabalhos futuros podem incluir extensões do que foi implementado nesta tese. Entre elas citamos o uso de múltiplas populações, *crossovers* e parâmetros adaptativos e, finalmente, buscas locais variáveis. A aplicação da metodologia de evolução de instâncias via AG também parece promissora.

CAPÍTULO IX

Referências

[1] K. Baker and G. Scudder, *Sequencing with earliness and tardiness penalties: A review*, Operations Research, (38), pp. 22-36, (1990).

[2] K. Boese, *Cost versus Distance in the Traveling Salesman Problem*, Tech. Report. TR-950018, UCLA, CS Department (1995). (pág. 66)

[3] E. K. Burke, J. P. Newall, R. F. Weare, *A Memetic Algorithm for University Exam Timetabling*, Lecture Notes in Computer Science (1153), pp. 241-250 (1996).

[4] R. Cheng and M. Gen, *Parallel Machine Scheduling Problems Using Memetic Algorithms*, Computers ind. Engng, (33), n. 3-4, pp. 761-764 (1997). (pág. 78)

[5] T. C. E. Cheng and M. C. Gupta, *A Survey of Scheduling Research Involving Due Date Determination Decisions*, EJOR (38), 156-166 (1989). (pág. 8)

[6] T. C. E. Cheng and C. C. S. Sin. *A State-of-the-Art Review of Parallel-Machine Scheduling Research*, EJOR (47), pp 271-292 (1990). (pág. 8)

[7] J. Du and J. Y. -T. Leung, *Minimizing Total Tardiness on One Machine is NP-hard*. Math. Opns. Res. (15), pp. 483-495 (1990). (pág. 8)

[8] P. M. França, M. Gendreau, G. Laport and F. Muller, *A Tabu Search Heuristic for the Multiprocessor Scheduling Problem with Sequence Dependent Setup Times*, Int. J. Prod. Econ. (43), pp. 79-89 (1996). (pág. 78, 81)

[9] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*, John Wiley, (1997).

[10] F. Glover, *Scatter Search and Star-paths - beyond the genetic metaphor*, OR Spektrum, (17), n. 2-3, pp. 125-137 (1995). (pág. 20)

[11] F. Glover and M. Laguna, *Tabu Search*, Kluwer (1997). (pág. 20, 78)

[12] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley (1989). (pág. 12)

[13] R. L. Graham, E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, *Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey*, Ann. Discr. Math. (5), pp. 287-326 (1979). (pág. 8)

[14] S. C. Graves, *A Review of Production Scheduling*, Operations Research (29), pp. 646-675 (1981). (pág. 8)

[15] S. K. Gupta and J. Kyparisis, *Single Machine Scheduling Research*, Omega (15) pp. 207-227 (1987). (pág. 8)

[16] T. Jones and S. Forrest, *Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms*, Proceedings of the 6th International Conference on Genetic Algorithms, pp. 184-192, Morgan Kaufmann (1995). (pág. 69)

[17] T. Kawaguchi and S. Kyan, *Deterministic Scheduling in Computer Systems: A Survey*, Oper. Res. Soc. Japan (31), pp. 190-217 (1986). (pág. 8)

[18] C. Koulamas, *The Total Tardiness Problem: Review and Extensions*, Operations Research (42), n. 6, pp. 1025-1041 (1994). (pág. 8)

[19] Y. H. Lee, K. Bhaskaran and M. Pinedo, *A heuristic to Minimize the Total Weighted Tardiness with Sequence-dependent Setups*, IIE Transactions (29), pp. 45-52 (1997). (pág. 11, 15, 17, 18, 19, 50)

[20] P. Merz and B. Freisleben, *Memetic Algorithms and the Fitness Landscape of the Graph Bi-partitioning Problem*, Proceedings of the 5th International Conference on Parallel Problem Solving from Nature - PPSN V, pp. 765-774, Springer (1998). (pág. 66, 69)

[21] P. Moscato, *An Introduction to Population Approaches for Optimization and Hierarchical Objective Functions: A Discussion on the role of Tabu Search*, Annals of Operations Research, (41), n. 1-4, pp. 85-121 (1993). (pág. 20)

[22] P. Moscato, *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms*, Caltech Concurrent Computation Program, C3P Report 826 (1989). (pág. 20)

[23] N. Nawaz, E. E. Enscore and I. Ham, *A heuristic algorithm for the M-machine, N-job flowshop sequencing problem*, Omega - International Journal of Management Science (11), n. 1, pp. 91-95 (1983). (pág. 14)

[24] P. S. Ow and T. E. Morton, *The Single Machine Early/Tardy Problem*, Management Science, (35), n. 2, pp. 177-191 (1989).

[25] G. L. Ragatz, *A Branch-and-Bound Method for Minimum Tardiness Sequencing on a Single Processor with Sequence Dependent Setup Times*. Proceedings of the 24th Annual Meeting of the Decision Sciences Institute, pp. 1375-1377 (1993). (pág. 9, 11)

[26] N. Raman, R. V. Rachamadugu and F. B. Talbot, *Real Time Scheduling of an Automated Manufacturing Center*, European Journal of Operational Research, (40), pp. 222-242 (1989). (pág. 11, 15)

[27] C. R. Reeves, *Landscapes, Operators and Heuristic Search*, Annals of Operations Research, to appear (1998). (pág. 66)

[28] P. A. Rubin and G. L. Ragatz, *Scheduling in a Sequence Dependent Setup Environment with Genetic Search*, Computers Ops. Res. (22), n. 1, pp. 85-99 (1995). (pág. 11, 26, 28, 30, 50)

[29] T. Sen and S. K. Gupta, *A State-of-the-Art Survey of Static Scheduling Research Involving Due Dates*, Omega (12), pp. 63-76 (1984). (pág. 8)

[30] K. C. Tan and R. Narasimhan, *Minimizing Tardiness on a Single Processor with Sequence-dependent Setup Times: a Simulated Annealing Approach*, Int. Journal of Management Science (25), n. 6, pp. 619-634 (1997). (pág. 11)

[31] A. Vepsalainen and T. E. Morton, *Priority rules for jobshops with weighted tardiness costs*, Management Science (33), pp. 1035-1047 (1987). (pág. 15)

[32] S. Wright, *The Roles of Mutation, Inbreeding, Crossbreeding and Selection in Evolution*, Proceedings of the 6th International Congress on Genetics, Vol. 1, pp. 356-366 (1932). (pág. 66)

Siglas:

AG – Algoritmo Genétido

AM – Algoritmo Memético

ATC – Regra de despacho de Raman *et al.*

ATCS – Regra de despacho de Lee *et al.*

B&B – *Branch and Bound*

BT – Busca Tabu

DA – Distância Acumulada

DH – Distância de Hamming

DU – Distribuição discreta uniforme

EDD – *Earliest Due Date*

EDD_{ins} – *Earliest Due Date* com inserção

LI – Limitante Inferior

MS – *Multiple Start*

OT – Solução ótima

OX – *Order Crossover*

RRX – Rubin & Ragatz *Crossover*

SA – *Simulated Annealing*

SMP – Sequenciamento em Máquinas Paralelas

SMS – Sequenciamento em Máquina Simples