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

ALGORITMOS MEMÉTICOS APLICADOS AO PROBLEMA DE *NO-WAIT FLOWSHOP*

Gilberto Jorge Tin Junior
Orientador: Dr. Paulo Morelato França

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação, como requisito parcial para a obtenção do título de Mestre em Engenharia Elétrica e Computação.
Área de concentração: Automação.

Banca Examinadora:

- Dr. Paulo Morelato França – FEEC/UNICAMP
- Dr. Takaaki Ohishi – FEEC/UNICAMP
- Dr. Fernando Antônio Campos Gomide - FEEC/UNICAMP
- Dr. Antônio Carlos Moretti - IMECC/UNICAMP

Campinas – SP – Brasil
6 de março de 2001

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

T49a

Tin Junior, Gilberto Jorge
Algoritmos meméticos aplicados ao problema de *no-wait flowshop* / Gilberto Jorge Tin Junior. --Campinas, SP: [s.n.], 2001.

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

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

RESUMO

O problema de *no-wait flowshop* é caracterizado pelo sequenciamento de tarefas em máquinas sem que as tarefas sofram interrupções em seu processamento, na máquina em si ou entre máquinas, desde o seu início até o fim de sua execução. Neste trabalho, considera-se o uso de restrições de tempo (de preparação, dependente da seqüência e data de liberação da tarefa para o chão de fábrica). O objetivo é o de minimizar o *makespan.* O algoritmo escolhido é o algoritmo memético pois tem se mostrado útil para a resolução de problemas complexos como o problema do caixeiro viajante assimétrico. Vários operadores de recombinação e busca local foram testados. Também é introduzida uma análise de *fitness landscape* para demonstrar o comportamento do algoritmo memético. Instâncias ótimas foram geradas a partir de instâncias do caixeiro viajante para comparar o desempenho do algoritmo memético com uma heurística construtiva. Os testes foram realizados com instâncias de 10 a 100 tarefas e com 2, 5 e 10 máquinas.

ABSTRACT

The no-wait flowshop problem is characterized by scheduling jobs on machines such that jobs may not have their execution interrupted on or between machines after they have been started. In this work, we consider the use of time restrictions (setup times and ready times). The objective function is to minimize the makespan. We use memetic algorithms to solve the problem because it has been usefull to solve complex problems such as the Asymmetric Traveling Salesman Problem (ATSP). Several recombination operators and local search operators have been tested. We also introduce a fitness landscape to analyse the memetic algorithm behaviour. Optimal instances were created from the ATSP instances found in the literature and used to compare the memetic algorithms to a constructive heuristic. The tests were made with instances having 10 to 100 jobs and 2, 5, and 10 machines.

DEDICATÓRIA

Dedico este trabalho à minha família que tanto me apoiou em meus estudos e em minha carreira profissional, mostrando o caminho através de conselhos e exemplos para que um dia eu pudesse galgar por mim mesmo minhas metas e realizar minhas obrigações, seja nos estudos ou no trabalho.

Dedico também à minha esposa Eliane e ao meu filho Giovanni que sempre me apoiaram com muito amor e paciência, compreendendo os obstáculos e dificuldades que pudessem advir. Não há riqueza maior na terra do que as coisas que aprendemos nessa vida e o que fazemos ao nosso próximo, pois o conhecimento e a caridade são mais importante do que qualquer tesouro ou bens que adquirimos porque quando morremos de nada eles nos servem.

Não entesourais para vós
tesouros na terra, onde a traça e
a ferrugem consomem e onde os
ladrões minam e roubam.
Mas entesourai para vós
tesouros nos céus, onde nem a
traça nem a ferrugem consomem
e onde os ladrões não minam
nem roubam.
Porque onde estiver o teu
tesouro, ali estará também o teu
coração.
(Mateus 6: 19-21)

AGRADECIMENTOS

Aos meus amigos de trabalho na IBM que me apoiaram e me incentivaram a cursar um mestrado quando então trabalhava na empresa. Especialmente ao Claúdio Nogueira de Meneses que foi meu coordenador na IBM e que muito me ajudou em meus estudos.

Ao professor Dr. Paulo Morelato França por ter depositado confiança em mim e ter me ajudado durante todo o mestrado.

Aos professores da FEEC/UNICAMP que me ajudaram durante suas aulas, nas quais muito pude aprender e aplicá-las em meu mestrado.

Aos amigos de aulas e trabalhos, nos quais participamos juntos durante o período acadêmico.

À Luciana Buriol e ao Pablo Moscato que me ajudaram muito com o *framework* do algoritmo memético e com suas idéias para o problema do *no-wait flowshop*.

Ao Alexandre Mendes que me ajudou nas instâncias do *no-wait flowshop* e com a análise do *fitness landscape*.

Ao Lucio Bianco, Paolo Dell´Olmo e Stefano Giordani por terem cedido as instâncias experimentais com limitantes inferiores para comparação do algoritmo memético.

Ao Paulo, Luciana e Pablo pelas reuniões e conversas que tivemos no CAB'S.

À UNICAMP por proporcionar ensino de alta qualidade, público e gratuito ao alcance de todos.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

INTRODUÇÃO

Algoritmos meméticos têm sido amplamente estudados e aplicados em vários problemas de otimização encontrados na literatura, tais como: designação quadrática (Merz & Freisleben [1997]), sequenciamento em máquinas paralelas (Cheng & Gen [1997]), particionamento de grafo (Bui & Moon [1996]), caixeiro viajante (Moscato & Norman [1992]), sequenciamento em máquinas (Mendes et al. [2001]) e outros.

Vários métodos foram estudados para resolver o problema de *no-wait flowshop* (NWFSP), desde regras de prioridade até algoritmos exatos. Métodos determinísticos como *branch & bound* e o algoritmo de Gilmore e Gomory [1964] encontram resultados ótimos mas para instâncias pequenas e pouco usadas na prática. Tais métodos determinísticos têm provado ser praticamente inviáveis devido ao alto custo computacional para se obter resultados próximos da otimalidade. Quando os obtêm, ou levam muito tempo ou os conseguem para instâncias com apenas 2 máquinas como o de Gilmore e Gomoroy [1974] ou com poucas tarefas (abaixo de 20) como em Wismer [1972].

O problema de *no-wait flowshop* abordado nesta dissertação tem como objetivo a minimização do *makespan,* ou seja, o tempo gasto para processar todas as tarefas desde o início da primeira operação da primeira tarefa até o fim da última operação da última tarefa processada na seqüência. Além disso, possui como restrições a data de liberação das tarefas e o tempo de preparação entre duas tarefas subsequentes, que no caso em estudo é considerado dependente da seqüência em que as tarefas são executadas.

O objetivo desta dissertação é o de desenvolver e testar a eficácia de um algoritmo memético para o problema de *no-wait flowshop* quando reduzido ao problema do caixeiro viajante e, ao mesmo tempo, expandir o *framework* utilizado por Buriol [2000] para o problema do caixeiro viajante. Também será mostrado uma análise de *fitness landscape* para algumas instâncias do *no-wait flowshop* bem como a geração de instâncias com resultados ótimos usadas para as comparações.

A dissertação está organizada em 4 capítulos.

O Capítulo 1 é dedicado ao problema de *no-wait flowshop,* apresentando a definição do problema, a aplicação prática, a redução do problema para o problema do caixeiro viajante, a complexidade do problema e a revisão bibliográfica.

O Capítulo 2 descreve o algoritmo memético utilizado, sua aplicação em problemas combinatórios, introduz o *framework,* apresenta a organização hierárquica da população de soluções, os métodos para manter diversidade na população e reiniciá-la quando se perde diversidade, e apresenta, também, os operadores de recombinação, mutação e busca local utilizados.

O Capítulo 3 analisa os operadores de busca local, *crossover*, mutação e o comportamento do algoritmo memético para algumas instâncias através de uma análise de *fitness landscape*.

O Capítulo 4 tem como objetivo principal descrever o algoritmo comparativo utilizado, a forma de geração das instâncias experimentais e das instâncias com resultados ótimos conhecidos e apresentar os resultados computacionais obtidos.

CAPÍTULO 1
O PROBLEMA DE *NO-WAIT FLOWSHOP* (NWFSP)

Com o surgimento da indústria de manufatura moderna junto com as metodologias de produção *just-in-time* e inventário zero para a redução dos custos da produção, o problema de *no-wait flowshop* ganhou grande importância na comunidade científica principalmente na indústria química, farmacêutica e de laminação de metais, pois esses problemas podem ser modelados como um problema de *no-wait flowshop*.

Vários são os métodos utilizados para a resolução desse problema e incluem técnicas de eliminação implícita (*branch & bound*, regras de eliminação), heurísticas, metaheurísticas, regras de despacho e outros. Os mais usados são os métodos de heurísticas e metaheurísticas devido a eficiência para encontrar soluções próximas do ótimo para instâncias grandes e em pouco tempo.

1.1. DEFINIÇÃO DO NWFSP

O problema de *no-wait flowshop* consiste de um conjunto M de *m* máquinas e um conjunto J de *n* tarefas, sendo que cada tarefa possui *m* operações que são processadas nas *m* máquinas. Cada máquina pode processar somente uma operação ao mesmo tempo e uma tarefa só pode ser processada em uma máquina por vez.

Cada máquina também possui um tempo de preparação da máquina s_{ij}^k tal que $1 \leq i, j \leq n$. Sendo s_{ij}^k o tempo necessário para preparar a máquina *k* para processar uma tarefa *j* depois de uma tarefa *i*. Esse tempo de preparação não é o mesmo quando a ordem é invertida de *i* para *j* caracterizando-se assim em um problema assimétrico. A preparação da máquina pode ser realizada enquanto a tarefa seguinte a ser processada na máquina estiver sendo processada em uma outra máquina.

A principal característica do NWFSP é que a operação *i+1* de uma tarefa tem que ser processada logo após a operação *i* sendo $1 < i < m\text{-}1$. Isso caracteriza a restrição de *no-wait* do problema que não permite que haja tempo de espera no processamento de uma tarefa de uma

máquina para a próxima. O único tempo de espera que pode ocorrer é no início do processamento da tarefa na primeira máquina. Consequentemente, o tempo total de processamento de uma tarefa se caracteriza pela somatória dos tempos de processamento de cada operação em cada máquina. Essa restrição no tempo de espera geralmente ocorre em um ambiente caracterizado pela tecnologia do processo, i.e., quando, por exemplo, uma variação de temperatura pode influenciar na degeneração do material, ou pela falta de capacidade de armazenamento entre as máquinas. Desse modo, o NWFSP poderia ser chamado de um problema tipo flowshop sem espera. Preferimos, porém, nesta tese manter o nome em inglês.

O objetivo é achar uma seqüência das tarefas de forma que elas possam ser executadas em todas as máquinas satisfazendo a data de liberação r_i, $1 \leq i \leq n$, das tarefas no chão de fábrica e minimizando o *makespan*. O tempo de liberação das tarefas é o momento em que as tarefas ficam disponíveis para o processamento e o *makespan* é o tempo necessário para se completar todas as tarefas.

O *flowshop* pode ser determinístico ou estocástico e dinâmico ou estático. O problema determinístico assume que as tarefas são divididas por tipos e cada tipo possui um tempo de processamento fixo, enquanto no estocástico os tempos de processamento das tarefas variam de acordo com uma distribuição conhecida. O problema estático assume que existem n tarefas e um sequenciamento ótimo para ser achado, enquanto que no dinâmico o número de tarefas é incerto e varia com o tempo. O problema estudado aqui é o *no-wait flowshop* estático e estocástico.

1.2. APLICAÇÕES PRÁTICAS

As principais aplicações do *no-wait flowshop* são encontradas na indústria química e metalúrgica, mas outras indústrias como de alimentação, serviços, farmacêutica e células flexíveis de manufatura também podem utilizar o modelo de *no-wait flowshop*. O uso do modelo de *no-wait flowshop* é motivado pelo ambiente de produção *just-in-time* ou com inventário zero. Em tais ambientes procura-se reduzir custos com inventários e planejar a produção de forma a atender a demanda sem que haja excessos ou falta do produto, para que assim possa maximizar a utilização das máquinas e reduzir-se o custo de acordo com a demanda do mercado.

Na indústria de metais como alumínio e aço, a laminação de placas passa por uma série de processos que têm que ser realizados um após o outro. Primeiro, a placa é pré-aquecida e depois passa por uma série de rolamentos nos quais a espessura da placa é diminuída progressivamente até atingir sua especificação final. Nesse processo a placa tem que passar de uma máquina até a outra dentro da seqüência sem ficar esperando pela disponibilidade da próxima máquina. Qualquer espera pode causar o resfriamento do metal, causando, assim, problemas na laminação.

Na indústria química a restrição de *no-wait* pode ocorrer no envazamento de compostos químicos. Um composto químico depois de passar por um misturador tem que ser envazado logo em seguida, pois a exposição ao ar pode deteriorar o composto. Processo semelhante também ocorre na indústria de alimentação onde um alimento tem que ser enlatado logo após seu cozimento para garantir a qualidade do mesmo.

Outra aplicação do *no-wait flowshop* encontra-se na indústria de serviços onde o cliente tem uma alta intolerância a espera no processo.

Um exemplo de sequenciamento de tarefas com restrição de tempo de espera r_i, $1 \leq i \leq n$, igual a zero é mostrado na Figura 1. O exemplo mostra o sequenciamento de um problema para n=3 tarefas e m=3 máquinas e considera que todas as tarefas estão disponíveis ao mesmo tempo no chão de fábrica. A variável s^k_{ij} representa o tempo necessário para preparar a máquina k para processar a tarefa j depois que a tarefa i terminou seu processamento na máquina k e a variável p^k_i representa o tempo de processamento da tarefa i na máquina k.

Uma seqüência (solução) pode ser representada por $\sigma = (1, 2, \ldots, i\text{-}1, i, i\text{+}1, \ldots, n)$, onde i representa o nº da tarefa que aparece na i-ésima posição da seqüência.

Tabelas de tempo de preparação das máquinas e tempos de processamento

Máquina 1

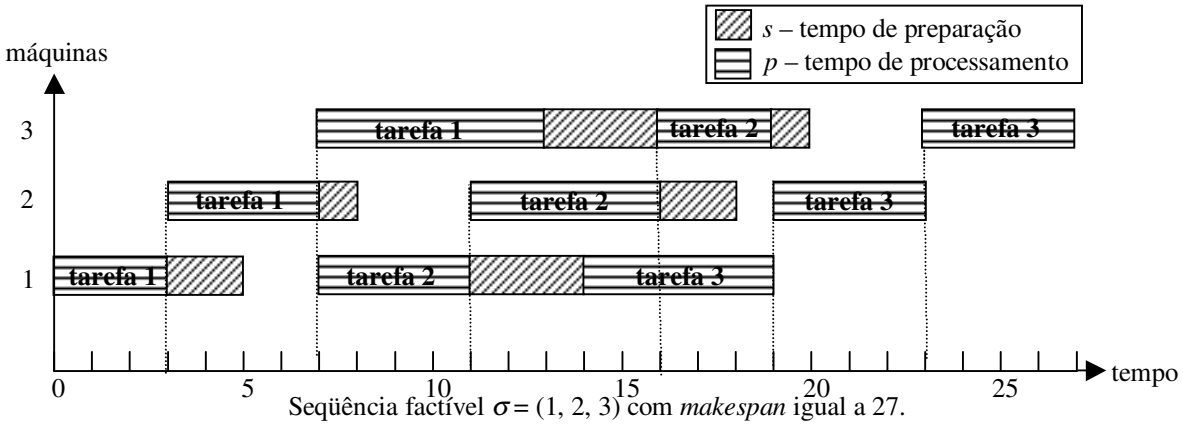
s^k_{ij}	1	2	3
1	-	2	1
2	2	-	3
3	4	1	-
p^k_i	3	4	5

Máquina 2

s^k_{ij}	1	2	3
1	-	1	3
2	1	-	2
3	3	2	-
p^k_i	4	5	4

Máquina 3

s^k_{ij}	1	2	3
1	-	3	2
2	2	-	1
3	1	3	-
p^k_i	6	3	4



Seqüência factível $\sigma = (1, 2, 3)$ com *makespan* igual a 27.

Figura 1: Sequenciamento factível de tarefas.

1.3. COMPLEXIDADE

Os problemas de otimização combinatória podem ser classificados como polinomial (P), polinomial não-determínistico (NP), polinomial não-determinístico completo (NP-completo) e polinomial não-determinístico difícil (NP-difícil).

Os problemas P são considerados de fácil manipulação e podem ser resolvidos em um tempo delimitado por um polinômio em função do tamanho da instância (conjunto de dados que representa o problema) em representação binária.

Os problemas NP são considerados de comportamento de difícil controle, consequentemente de difícil manipulação. Na classe NP se encontram os problemas de explosão combinatorial, para os quais provavelmente não existem algoritmos que os resolvam em tempo

polinomial. Os algoritmos existentes procuram uma solução factível, que pode ser ótima, para o problema. Esses algoritmos crescem exponencialmente em função do tamanho do problema.

A classe NP-completo é uma subclasse da NP e inclui os problemas de decisão (problemas cuja resposta é sim ou não). Um algoritmo que resolve um problema NP-completo resolve todos os outros problemas pertencentes a essa classe pois sempre existe uma transformação polinomial que relaciona um problema da classe NP-completo para um outro.

A classe NP-difícil é uma subclasse da NP-completo e inclui problemas de otimização cujo espaço de busca para soluções pode ser infinito. Um problema de decisão pode ser NP-difícil, mas não existem problemas de otimização que possam ser NP-completos. Se existir um algoritmo polinomial que resolva um problema NP-difícil então esse algoritmo também resolverá todos os problemas da classe NP-difícil, NP-completo e NP (Horowitz & Sahni [1978]).

Para mostrar que um problema está em uma dessas classes deve-se transformá-lo (de maneira polinomial) em um dos problemas pertencentes à classe.

O problema de *no-wait flowshop* foi provado ser NP-difícil para o caso de três ou mais máquinas e para duas máquinas se as tarefas não forem processadas em ambas as máquinas (Sahni & Cho [1979]). O problema com duas máquinas é resolvido em tempo polinomial se as tarefas forem processadas em ambas as máquinas (Gilmore e Gomory [1964]).

1.4. REVISÃO BIBLIOGRÁFICA

Gilmore e Gomory [1964] mostram como resolver um problema de *flowshop* quando m=2 em tempo polinomial considerando todas as tarefas com duas operações.

Wismer [1972] mostra o uso de um algoritmo *branch & bound* para resolver o problema de *no-wait flowshop*, que ocorre no processo de laminação de metais como alumínio e aço, através da redução do problema para o problema do caixeiro viajante assimétrico (ATSP) que é NP-difícil.

Sahni & Cho [1979] provaram que o problema de NWFSP com o objetivo de minimizar o tempo de término da produção (*makespan*) é NP-difícil mesmo quando restrito a duas máquinas, exceto quando o *flowshop* tiver exatamente duas operações por tarefa para todas as tarefas.

Dudek et al. [1992] afirmam que o estudo da complexidade nesse problema trouxe um grande impacto nas pesquisas. Os resultados das pesquisas não tiveram muito sucesso antes da teoria sobre problemas NP-completo pois havia um grande desejo pela busca de resultados ótimos usando algoritmos enumerativos ou exatos. Com o surgimento da teoria, ficou comprovado que o problema era de difícil manipulação favorecendo o uso de algoritmos não exatos como heurísticas e metaheurísticas. Dudek et al. também afirmam que o único problema de *flowshop* usado na prática é o de *no-wait flowshop* com o objetivo de minimizar o tempo total de processamento (*minimum flowtime*) encontrado em Wismer [1972].

Uma revisão dos problemas de NWFSP estudados na literatura foi apresentada por Hall & Sriskandarajah [1996]. O estudo desse problema é motivado pelos conceitos de inventário zero e produção *just-in-time* que surgem no processo de sistemas de manufatura moderno.

Fink & Voß [1998] propõem o uso de metaheurísticas (*simulated annealing* e *tabu search*) para o problema de *no-wait flowshop* possibilitando tratar instâncias maiores do que as estudadas na literatura até aquela data.

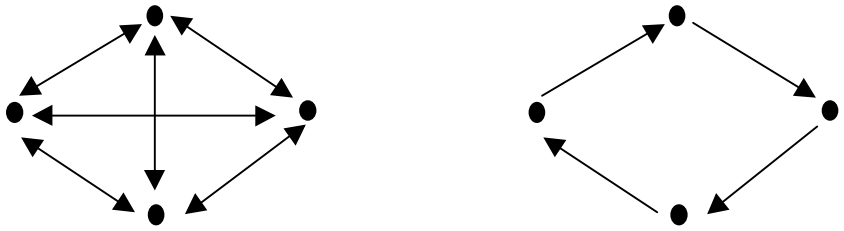
Bianco, Dell'Olmo & Giordani [1999] mostram a redução do NWFSP, com tempos de preparação dependentes da seqüência e datas de liberação, para o ATSP com restrições de tempo para visitar as cidades. Eles também propõem uma heurística construtiva e a geração de limitantes inferiores baseados na relaxação *lagrangiana* do modelo matemático do problema.

1.5. O PROBLEMA DO CAIXEIRO VIAJANTE ASSIMÉTRICO (ATSP)

O problema do ATSP consiste de um conjunto de n cidades e uma matriz de custo c_{ij} que representa o custo (eg., distância) de viajar da cidade i à cidade j sendo que $c_{ij} \neq c_{ji}$. O objetivo do caixeiro é sair de uma cidade e passar por todas as outras uma única vez e retornar à cidade de partida pelo menor custo possível.

O ATSP também pode ser formulado como um grafo completo. É dado um grafo G = (V,A), um conjunto V de vértices e um conjunto A de arestas que conectam cada vértice v a todos os outros do grafo. A cada aresta é associado um peso c_{ij} que representa o custo de ir de um vértice ao outro. O objetivo é encontrar o ciclo hamiltoniano de custo mínimo.

A Figura 2 mostra um grafo completo e uma rota possível do ATSP.



Grafo completo Rota factível

Figura 2: Grafo ATSP.

A formulação matemática de minimização do ATSP pode ser descrita da seguinte forma:

$$\text{Função Objetivo :} \qquad MIN \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\text{Sujeito a :} \qquad \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1,\ldots,n \qquad (1)$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1,\ldots,n \qquad (2)$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \qquad \forall S \subset V, S \neq \phi \qquad (3)$$

$$x_{ij} \in \{0,1\} \quad i, j = 1,...,n, i \neq j \qquad (4)$$

A variável inteira x_{ij} é uma variável de decisão que indica se a cidade *j* é visitada logo após a cidade *i*. Se $x_{ij} = 1$ então *j* é visitada logo após *i*, caso contrário $x_{ij} = 0$. A variável *n* representa o número de cidades a serem visitadas e o conjunto S é um subgrafo de G em que |S| representa o número de vértices desse subgrafo.

A restrição (1) indica que para cada cidade *j* existe apenas um arco incidente. A restrição (2) indica que para cada cidade *i* existe apenas um arco saindo.

A restrição (3) é usada para garantir que não haverá ocorrência de sub-rotas.

A restrição (4) indica o domínio das variáveis x_{ij}.

1.6. REDUÇÃO DO NWFSP PARA O PROBLEMA DO CAIXEIRO VIAJANTE ASSIMÉTRICO

O NWFSP quando reduzido para o ATSP se torna um grafo completo G = (V,A) sendo V um conjunto de n+1 vértices e A o conjunto de arcos que conectam os vértices. A introdução de uma tarefa fictícia, com p_0=0 e s_{0j}=s_{i0}=0 para $1 \leq i, j \leq n$ para todas as máquinas, se faz necessário para garantir uma seqüência factível do NWFSP. Então a solução do ATSP se torna uma rota hamiltoniana com n+1 cidades.

A tarefa fictícia tem por objetivo eliminar os custos de preparação da máquina incluídos no último arco percorrido pelo caixeiro viajante no problema do ATSP, pois uma seqüência factível do NWFSP não possui o custo de preparar uma máquina para processar a primeira tarefa depois que a última tarefa foi sequenciada (Figura 3). Sem a tarefa fictícia, esse custo de preparação é incluído no problema do ATSP. A tarefa fictícia também garante que o último arco da rota do ATSP corresponda ao tempo total para processar a última tarefa na seqüência do NWFSP. Na Figura 3, considere g sendo o tempo total para processar a última tarefa em todas as máquinas.

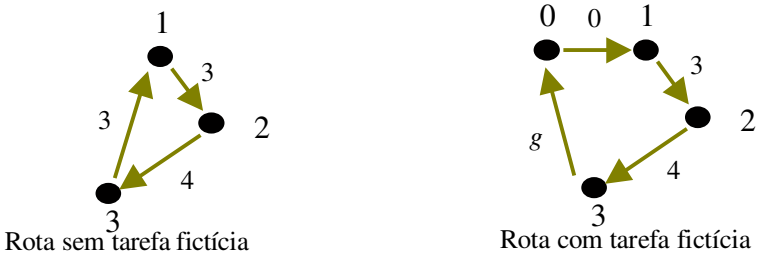


Figura 3: Influência da tarefa fictícia na rota do ATSP.

Para permitir que as operações das tarefas sejam processadas sem interrupções em todas as máquinas, a ordem das tarefas processadas tem que ser a mesma em todas as máquinas, o que caracteriza o problema de *no-wait flowshop* como um caso especial de *flowshop* permutacional. Para que isso aconteça, um atraso tem que ser imposto, quando necessário, na primeira operação de cada tarefa a ser executada. Esse atraso se torna o custo do arco do ATSP e é calculado da seguinte forma:

$$c_{ij} = \max_{1 \leq k \leq m} \left[s_{ij}^k + \sum_{h=1}^{k} (p_i^h - p_j^h) + p_j^k \right], \qquad \forall i,j \in \mathrm{J} \cup \{0\} \quad (i \neq j)$$

A função acima representa o atraso necessário para sequenciar j depois de i incluindo a

tarefa fictícia 0 no conjunto J de tarefas do problema. A variável s^k_{ij} representa o tempo de preparação da máquina *k* quando uma tarefa *j* é processada depois de uma tarefa *i* e a variável p^k_i representa o tempo de processamento da tarefa *i* na máquina *k*.

A Figura 4 mostra a rota hamiltoniana do sequenciamento da Figura 1.

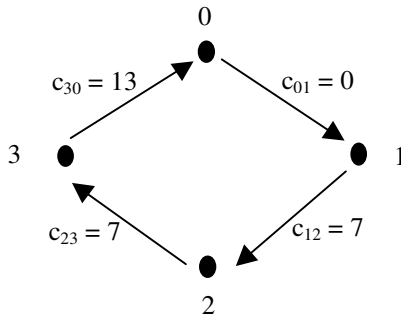
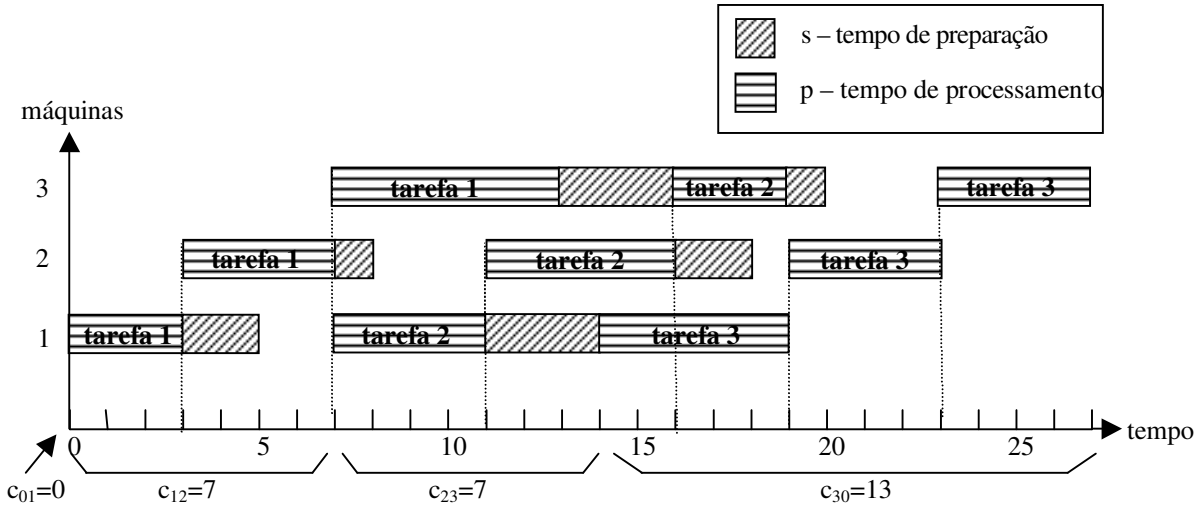


Figura 4: Rota Hamiltoniana.

A Figura 5 mostra como ficaria a seqüência factível do *no-wait flowshop* baseado na rota hamiltoniana da Figura 4. A tarefa fictícia sempre será a primeira e também será incluída no fim do sequenciamento. Desta forma, o ciclo hamiltoniana sempre representará um sequenciamento factível do *no-wait flowshop* e a somatória do atraso c_{ij} representa o valor da função objetivo. A matriz de tempos de preparação, o vetor de tempos de processamento e as datas de liberação são as mesmas da Figura 1.



Seqüência factível σ = (0,1, 2, 3)

Figura 5: Seqüência factível de tarefas incluindo a tarefa fictícia.

Na Figura 5 é nítida a relação entre o custo de uma cidade *i* à uma cidade *j* em relação ao sequenciamento das tarefas no problema NWFSP. Esse custo c_{ij} entre as cidades representa no problema NWFSP o tempo em que uma tarefa *j* tem que esperar para iniciar seu processamento, no chão de fábrica, depois que uma tarefa *i* iniciou seu processamento. Portanto, c_{ij} é um custo que inclui o tempo de processamento da tarefa *i*, o tempo de preparação entre *i* e *j* e um tempo de ociosidade necessário para garantir que a tarefa *j* inicie seu processamento sem sofrer interrupções entre as operações.

Quando o caixeiro chegar em uma cidade, ele deve esperar até o tempo r_i que determina quando uma cidade *i* está liberada para ser visitada. Ou seja, se ele chegar antes do tempo r_i então ele deve esperar até r_i. O ATSP com essa restrição se torna o problema do caixeiro viajante assimétrico com datas de liberações para se visitar as cidades (ATSP-RT).

CAPÍTULO 2
ALGORITMO MEMÉTICO PARA A SOLUÇÃO DO NWFSP

2.1. INTRODUÇÃO

Heurísticas são baseadas em princípios, idéias acumuladas, suposições, ou informação sobre o problema que visam reduzir o espaço de busca e de orientar o processo de escolha de soluções factíveis no domínio do problema. As heurísticas têm sido muito eficientes na redução do espaço de busca de problemas (Figura 6) onde algoritmos enumerativos levariam até anos para obter uma solução ótima. As heurísticas, especialmente as metaheurísticas, são aplicadas em problemas onde uma solução aproximada, cerca de 2% ou 3% distante do ótimo, traria mais benefícios do que uma solução exata, principalmente em relação a redução do tempo necessário para obtê-la.

Problemas como de sequenciamento de tarefas, onde as características das tarefas e as rotas das mesmas mudam constantemente e o tempo para se obter uma seqüência é crítico, é um bom exemplo para o uso de metaheurísticas; em contraste com problemas cuja rota não muda constantemente como o de um carteiro que sempre faz o mesmo caminho para entregar cartas.

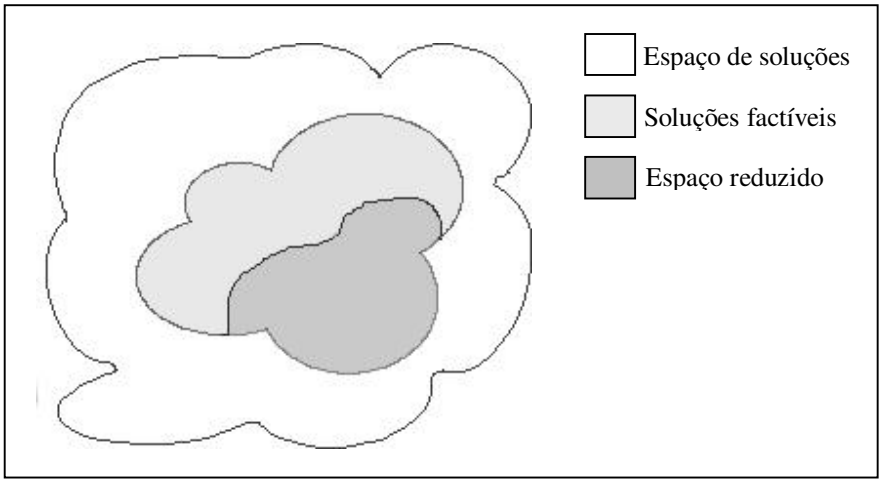


Figura 6: Espaço de busca do domínio do problema.

Algoritmo memético é uma metaheurística, uma estratégia em um nível mais alto que guia outras heurísticas em busca de soluções factíveis, pertencente à classe dos algoritmos populacionais. Algoritmo populacional usa vários indivíduos na procura de soluções factíveis no espaço de busca (Figura 7).

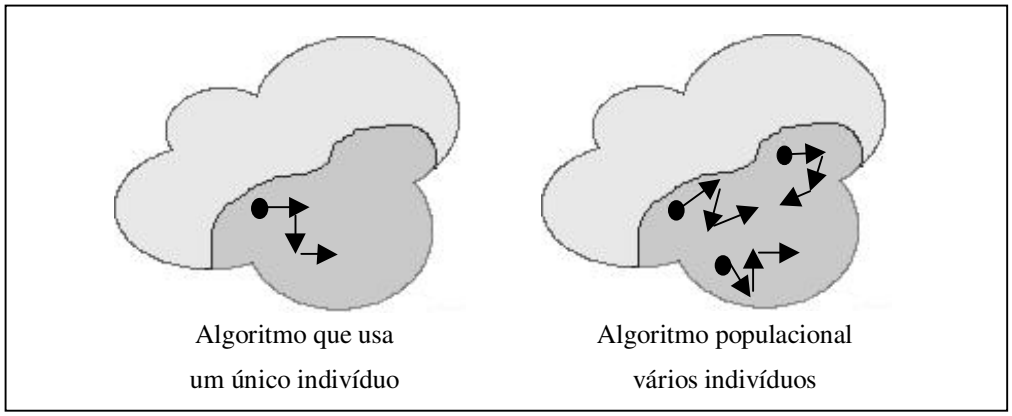


Figura 7: Algoritmo populacional vs. algoritmo não populacional.

O termo algoritmo memético surgiu pela primeira vez em 1989 quando Moscato procurou definir os limites e diferenças entre algoritmo memético e algoritmo genético. R. Dawkins [1976] em seu livro "*The Selfish Gene"* define o termo "meme" como uma unidade de imitação durante a transmissão cultural. Segundo Moscato [1989], algoritmo memético é uma definição mais apropriada para algoritmo genético híbrido, pois o termo híbrido tira o algoritmo genético de suas raízes da biologia. Algoritmo memético está mais associado a algoritmo evolutivo híbrido do que a algoritmo genético híbrido, primeiro por causa de sua representação que na maioria dos casos envolve inteiros, reais e também binários e depois pelo uso de heurísticas que exploram o espaço de busca através de movimentos nos indivíduos.

2.2. REPRESENTAÇÃO DA SOLUÇÃO

A escolha da representação correta para a solução do problema, bem como a estratégia de busca e os movimentos no espaço de soluções, pode criar ou evitar ótimos locais (Moscato

[1989]). Um algoritmo aplicado sucessivamente à uma solução inicial tende a levar as soluções subsequentes à um ótimo local. Uma vez atingido um ótimo local, a aplicação deste algoritmo não consegue melhorar a solução, fazendo o valor da função objetivo piorar. (Figura 8).

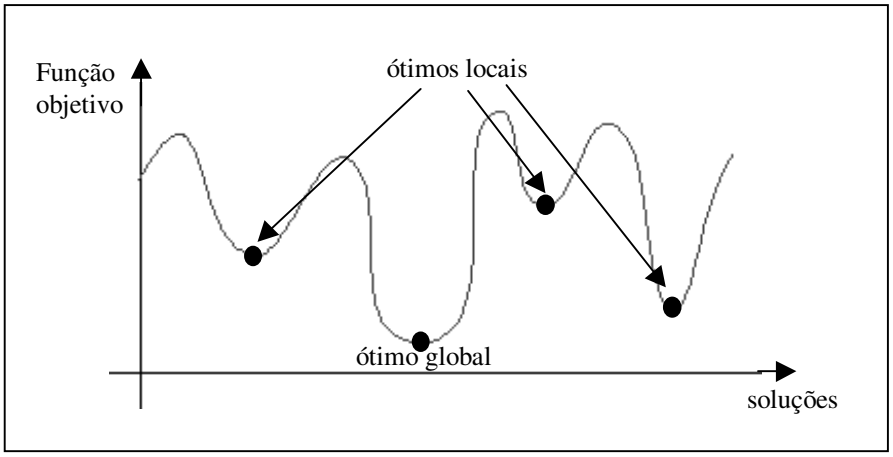
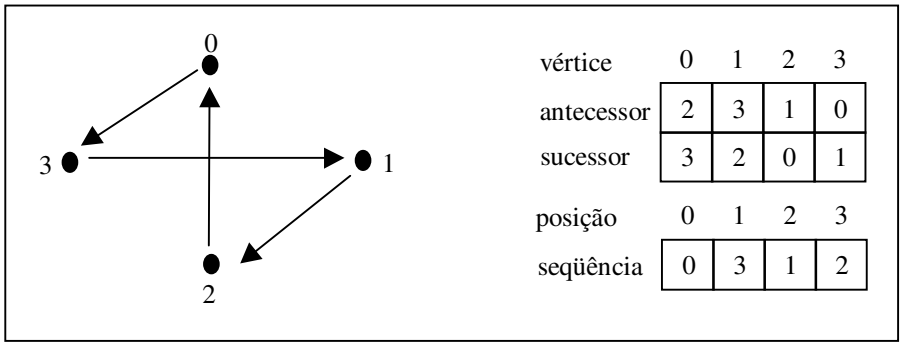


Figura 8: Ótimos Locais para um problema de minimização.

A representação escolhida para esse problema envolve dois vetores de tamanho *n*. Um vetor de vértices antecessores e sucessores a *i* e outro vetor com a seqüência da solução do problema (Figura 9). A escolha de vetores foi utilizada pois permite a troca de vizinhos em tempo O(1) para o primeiro vetor e o segundo vetor é atualizado quando a função objetivo for atualizada, ou seja, depois que a recombinação é efetuada e depois que a busca local é aplicada aos novos indivíduos ou indivíduos que sofreram mutação.



vértice	0	1	2	3
antecessor	2	3	1	0
sucessor	3	2	0	1

posição	0	1	2	3
seqüência	0	3	1	2

Figura 9: Representação da solução.

2.3. ESTRUTURA DA POPULAÇÃO

A população usada no algoritmo memético é composta de 13 agentes organizados em uma árvore ternária. Um agente funciona como um recipiente que guarda duas soluções do problema denominadas *Pocket e Current*. O *Pocket* é uma memória que guarda a melhor solução gerada em um agente através dos operadores de recombinação, busca local e mutação aplicados ao *Current* (Moscato e Norman [1992]).

A organização da população em uma árvore permite criar 4 subpopulações que podem ser comparadas a ilhas de conhecimento sobre o problema. Essas ilhas possuem indivíduos que competem e cooperam para a resolução do problema e, a cada geração do algoritmo, elas trocam informações entre si. Esse processo se assemelha muito à computação paralela onde cada subtarefa estaria associada as subpopulações, podendo ser executadas em computadores ou processadores distintos. A Figura 10 mostra a organização da população em uma árvore ternária bem como as subpopulações criadas em conseqüência da estrutura e da forma como os operadores são aplicados a ela.

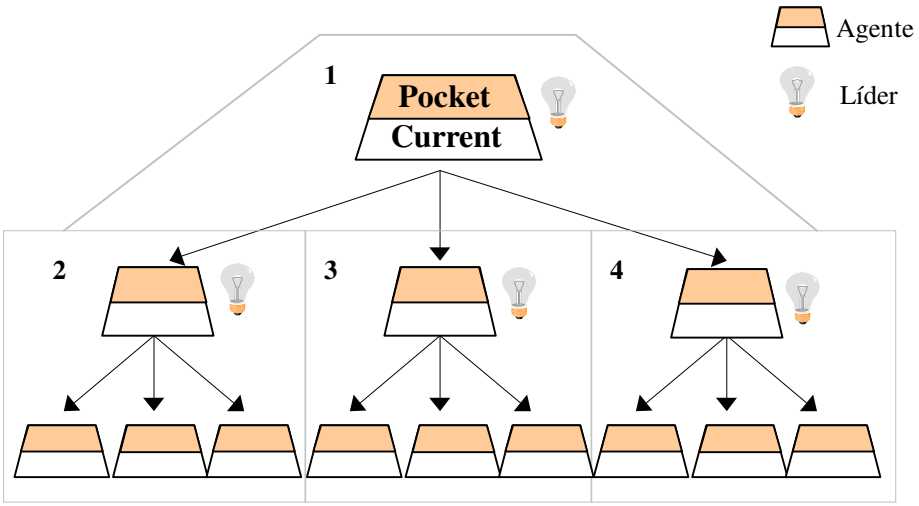


Figura 10: População organizada como uma árvore ternária de agentes.

As soluções *currents,* depois de passar pelos processos do algoritmo memético, podem trocar de lugar com as soluções *pockets* de seus respectivos agentes somente se o valor de sua função objetivo for melhor que a do *pocket*. A estrutura da população é organizada de tal forma que depois de cada geração do algoritmo os líderes de cada subpopulação possuem os *pockets*

com melhor valor de função objetivo dentre os seus subordinados e os *pockets* dos subordinados ficam ordenados em ordem não decrescente do valor de sua função objetivo.

2.4. OPERADORES

O algoritmo memético usa uma combinação de três operadores para a exploração do espaço de busca. Os operadores são de recombinação, evolução cultural (ou busca local) e mutação. Para a aplicação do algoritmo memético ao problema NWFSP vários operadores foram implementados e adicionados ao *framework* criado para resolver problemas de otimização combinatória. A escolha desses operadores é muito importante pois a sinergia entre eles é o que garante a convergência do algoritmo para uma solução ótima ou próxima dela.

Esses operadores realizam movimentos que alteram a configuração da solução. Quando esses movimentos são realizados, a distância de um vértice a outro não é apenas a distância da matriz do ATSP devido a restrição de data de liberação que exige que um caixeiro espere até que a cidade esteja disponível para ser visitada.

Então, para saber se um movimento que um operador realiza reduz ou não o *makespan* é necessário percorrer todos os elementos de uma solução, pois os movimentos, dos operadores do algoritmo memético aqui utilizados, alteram também o tempo que um caixeiro precisa esperar entre uma cidade e outra. Como o custo para descobrir a alteração que um movimento realiza na solução é muito alto, pois não depende somente do c_{ij} da matriz do ATSP, então é interessante realizar movimentos que valorizem cidades que tenham datas de liberação r_i próximas uma das outras ou que estejam próximas do tempo percorrido a_i pelo caixeiro viajante para chegar até determinada cidade i.

Para garantir que o caixeiro visite cidades com datas de liberação próximas, então uma distância d_{ij} que recebe a influência das datas de liberação é utilizada nos operadores do algoritmo memético. Essa distância pode ser descrita da seguinte forma:

Se a_i for conhecido então $d_{ij} = \max(c_{ij}, r_j - a_i)$; senão $d_{ij} = \max(c_{ij}, r_j - r_i)$, $0 \leq i,j \leq n$.

A variável d_{ij} é a nova distância de ir de i a j, a variável c_{ij} é a distância de i a j na matriz

do ATSP, a variável r_j é a data de liberação para se visitar a cidade j e a variável a_i é a data em que o caixeiro chegou na cidade i.

Como d_{ij} não representa a distância real entre uma cidade e outra, então é importante recalcular o valor da função objetivo, depois que um movimento é escolhido, para saber se aquele movimento realmente reduz o valor da mesma.

Considere uma seqüência σ = < 1, 2, 3, 4, 5, 6 >, o vetor de datas de liberação r ={0, 4, 10, 8, 12, 8} e que o custo entre uma cidade e outra fosse 2. Suponha que um movimento de inserção retirasse a cidade 4 de sua posição atual e a inserisse em uma posição que reduzisse o valor da função objetivo. O movimento escolhido, usando como base a distância d_{ij}, seria inserir a cidade 4 entre as cidades 2 e 3 e teria complexidade O(n) pois o custo do movimento é a diferença das distâncias dos arcos que saem e que entram para formar uma nova rota mais a complexidade de calculas a função objetivo. Se o movimento não usasse como base a distância d_{ij} e sim a distância c_{ij}, então o custo do movimento seria a complexidade de calcular o valor da função objetivo, O(n), vezes o número de vezes que a função objetivo precisa ser calculada para determinar a melhor posição de inserção da cidade 4, no pior dos casos é n.

2.4.1. Recombinação

A recombinação é uma forma de busca que usa um ou mais indivíduos para explorar o espaço de soluções do problema. Um operador de recombinação pode ser assexudada ou sexuada com dois ou mais indivíduos. No caso do NWFSP o operador usado é sexuado com dois indivíduos. Durante a recombinação os indivíduos trocam informações ou genes. Usualmente as informações/genes similares entre ambos é mantida na sua descendência.

Goldberg [1989] cita o matemático J. Hadamard [1949, p.29] que afirma o seguinte sobre a importância da justaposição de idéias para descobrir novas invenções:

> We shall see a little later that the possibility of imputing discovery to pure chance is already excluded. On the contrary, that there is an intervention of chance but also a necessary work of unconciousness, the latter implying and not contradicting the former… Indeed, it is obvious that invention or discovery, be in mathematics or anywhere else, takes place by combining ideas.

Hadamard afirma que a descoberta de novas invenções não se dá simplesmente por acaso, mas sim por uma combinação de idéias misturadas com alguma irracionalidade. A recombinação justamente adota esse comportamento ao combinar idéias parciais de soluções distintas para criar novas soluções.

2.4.1.1. *Strategic Arc Crossover* (SAX)

O operador SAX é uma adaptação feita por Buriol et al [1999], para o caso do caixeiro viajante assimétrico, do operador *Strategic Edge Crossover* (SEX) proposto por Moscato e Norman [1992] para o problema do caixeiro viajante simétrico.

Dado dois indivíduos para recombinação, o SAX cria um descendente em 3 passos:

a) Construção de uma estrutura auxiliar (*ArcMap*) usada para relacionar os vizinhos de cada cidade nas duas rotas;
b) Criação de seqüências a partir da estrutura *ArcMap*;
c) Aplicação de uma heurística para unir as seqüências geradas em (b).

Considere o exemplo mostrado na Figura 11. Sendo A e B, os pais representados por dois ciclos hamiltonianos.

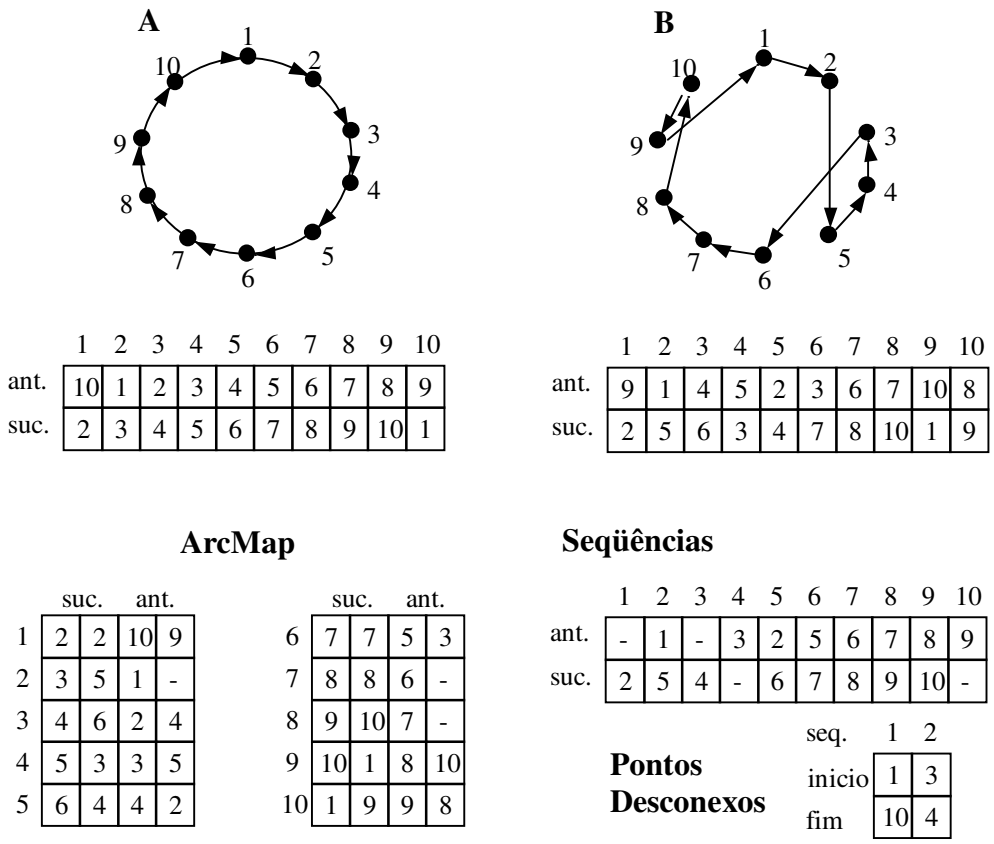


Figura 11: *Strategic Arc Crossover.*

O *ArcMap* representa uma tabela de *n* linhas, que representam os vértices, e 4 colunas, que representam os vizinhos sucessores e antecessores ao vértice *i* em cada uma das rotas.

A criação das seqüências ocorre da seguinte forma:

1) Um vértice *i* é escolhido aleatoriamente.
2) A cidade sucessora a *i* é escolhida em função do *ArcMap*. Uma cidade sucessora é escolhida aleatoriamente entre as cidades sucessoras e a ocorrência dela na coluna de sucessoras é eliminada em todas as linhas. Se *i* não tem cidade sucessora, então vá para o passo 1. A eliminação da cidade sucessora ocorre em O(1) pois se usa a coluna antecessora para isso.

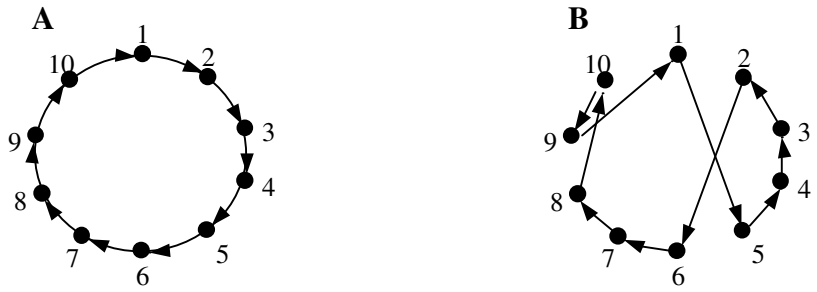
Depois que as seqüências foram criadas (Figura 11), faz-se o uso de uma heurística para conectar esses pontos desconexos. A heurística usada nesse algoritmo é uma heurística constutiva do vizinho mais próximo que valoriza a menor distância entre um ponto desconexo *i* que termina uma seqüência e os outros pontos desconexos que iniciam as outras seqüências.

2.4.1.2. *Partially Matched Crossover* (PMX)

PMX surgiu na tentativa de construir operadores de recombinação para o problema do caixeiro viajante cego. No problema do caixeiro viajante cego o objetivo também é achar a menor rota para viajar por *n* cidades, mas nesse problema ele não conhece a distância entre as cidades e só fica sabendo quando ele completa a rota (Goldberg [1989]).

No PMX, dois indivíduos são alinhados e dois pontos de *crossover* são escolhidos aleatoriamente. Um dos pais é escolhido aleatoriamente e o trecho entre os dois pontos é copiado para o filho na mesma posição em que se encontrava no pai.

As outras posições são preenchidas com os genes do outro pai através de trocas que respeitam a posição absoluta das cidades. A Figura 12 ilustra o *crossover* realizado por dois indivíduos A e B sendo que o indivíduo A é o escolhido aleatoriamente para copiar o trecho entre os pontos de *crossover* 4 e 6. Note que a cidade na posição 3 do filho é a cidade na posição 4 do indivíduoB.



pos / rota	1	2	3	4	5	6	7	8	9	10
A	1	2	3	4	5	6	7	8	9	10
B	1	5	4	3	2	6	7	8	10	9
filho	1	2	3	4	5	6	7	8	10	9

Figura 12: *Partially Matched Crossover.*

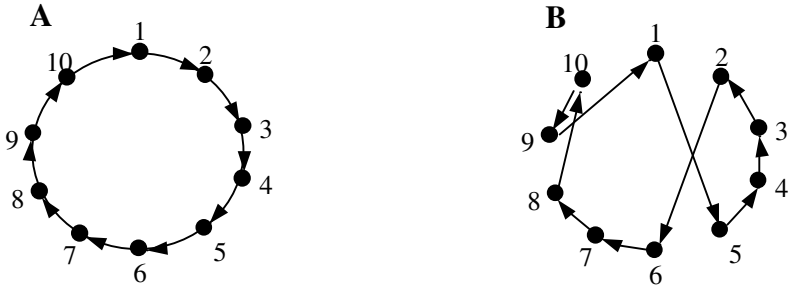
2.4.1.3. *Ordered Crossover* (OX)

OX também surgiu na tentativa de construir operadores para o caixeiro viajante cego (Goldberg [1989]). Mas diferentemente do PMX, que respeita a posição absoluta das cidades, o OX respeita a posição relativa entre as cidades.

O procedimento de construir um filho é muito parecido com o PMX e ocorre da seguinte forma:

1) Determine dois pontos de *crossover* aleatoriamente.
2) Selecione um pai cujo segmento entre os pontos escolhidos em (1) será copiado na mesma posição para o filho.
3) Os genes do pai que não foi escolhido em (1), exceto os genes que já se encontram no filho, são copiados, respeitando a ordem, para o filho a partir do segundo ponto de crossover.

Observe o exemplo da Figura 13 onde dois pais A e B cujos pontos de *crossover* são 4 e 6 geram um filho. Primeiramente, o trecho entre 4 e 6 de A é copiado para o filho. Depois, o filho é preenchido a partir da posição 7 com os genes do pai B começando da posição 7, passando pelo fim, indo para o começo e terminando em 6.



pos / rota	1	2	3	4	5	6	7	8	9	10
A	1	2	3	4	5	6	7	8	9	10
B	1	5	4	3	2	6	7	8	10	9
filho	1	3	2	4	5	6	7	8	10	9

Figura 13: *Ordered Crossover.*

2.4.1.4. *Distance Preserving Crossover* (DPX)

O DPX surgiu em 1996 quando Freisleben e Merz observaram que a distância entre ótimos locais é semelhante à distância entre uma rota ótima local e uma rota ótima global. Então surgiu a idéia de criar um operador de recombinação que, dado duas soluções, gera uma solução com distância a ambos os pais equivalente à distância entre os pais.

Em outras palavras, dado as soluções A e B considere um filho com arcos $A \cap B+S$, onde S é o conjunto de arcos que preenchem o grafo desconexo para formar uma rota. O conjunto S de arcos é escolhido usando uma heurística construtiva do vizinho mais próximo que considera $E-(A-B) \cup (B-A)$ arcos para fazerem parte da rota sendo E o conjunto de arcos de um grafo completo.

2.4.2. Evolução Cultural

Após a criação de um indivíduo pela recombinação, este sofre um processo de aprendizado ou evolução que explora o máximo de conhecimento, tornando o indivíduo pronto para se reproduzir e gerar novos indivíduos que herdarão suas características genéticas e culturais. Esse processo de evolução é usualmente realizado por algoritmos de busca local, cujo espaço vizinho ao indivíduo é explorado através de movimentos determinados pelo tipo de busca, mas que também pode ser realizado por outras heurísticas ou algoritmos exatos.

Um indivíduo atinge o máximo de aprendizado quando ele atinge o ótimo local do espaço que ele explora.

Cinco algoritmos de busca local foram testados no problema NWFSP. O algoritmo *Recursive Arc Insertion* (RAI), *Recursive Guided Insertion* (RGI), *Recursive Guided Insertion with Nearest Neighbor* (RGI/NN), inserção e trocas de todos os pares possíveis.

2.4.2.1. *Recursive Arc Insertion* (RAI)

RAI foi introduzido por Buriol et al. [1999] para o problema do caixeiro viajante assimétrico. A busca RAI trabalha recursivamente com a inserção de arcos nos pontos considerados críticos da rota. Os pontos considerados críticos são escolhidos de acordo com algum critério. No caso do NWFSP, o critério escolhido são os pontos de *crossover*, os pontos onde ocorre mutação e os vértices que ficaram desconexos durante o *crossover*. A Tabela 1 exemplifica melhor os pontos considerados críticos em cada operador de *crossover* utilizado nessa dissertação. A redução do espaço de busca aos pontos críticos da solução não faz com que a busca perca em qualidade pois a recursividade garante uma abrangência maior e com qualidade a outras áreas de vizinhança.

Tabela 1: Pontos críticos gerados pelos operadores de *crossover*.

Crossover	Pontos críticos
DPX	Ponto inicial e final de cada seqüência desconexa formada pelo operador.
SAX	Ponto inicial e final de cada seqüência desconexa formada pelo operador.
PMX	Pontos que sofrem trocas de posições por estarem dentro do segmento de *crossover* do primeiro pai.
OX	Pontos de *crossover* e pontos que se encontram dentro do trecho de *crossover* do primeiro pai selecionado mas que não se encontram no mesmo trecho no segundo pai. Seja < _ _ _ 4 5 2 6 _ _ _> o segmento de *crossover* do primeiro pai e < _ _ _4 3 2 6 _ _ _> o do segundo pai. Então os pontos de *crossover* são 4 ,5 e 6.

O movimento básico realizado pela busca local é o movimento 3-Opt. O movimento 3-Opt retira 3 arcos e insere 3 novos arcos com o objetivo de reduzir a rota percorrida pelo caixeiro viajante.

O algoritmo RAI utiliza duas matrizes bidimensionais de n linhas e 5 colunas que auxiliam nos movimentos da busca. A matriz *s-vizinhos(i)* contém os 5 vértices sucessores mais próximos de i dentre todos os n-1 vértices. A matriz *p-vizinhos(i)* contém os 5 vértices predecessores mais próximos de i dentre todos os n-1 vértices. Os vértices sucessores e predecessores a i estão ordenados em ordem não decrescente da sua distância a i.

O algoritmo pode ser descrito da seguinte forma:

1) Selecione um ponto crítico $i \in$ P (conjunto de pontos críticos) e P = P – $\{i\}$. Se P for vazio então pare. Faça *flag* = 0.
2) Escolha $j \in$ *s-vizinhos(i)* (Figura 14a). Se *j* for sucessor de *i* na rota então vá para (8).
3) Faça *a* = *suc(i)*, *b* = *pred(j)* (Figura 14b) e *m* = *j*. Calcule $\Delta_1 = d_{ia} + d_{bj} - d_{ij}$.
4) Faça *n* = sucessor de *m* (Figura 14c). Calcule $\Delta_2 = d_{ma} + d_{bn} - d_{mn}$.
5) Enquanto $\Delta_2 \geq \Delta_1$ e $n \neq i$ vá para (4).
6) Se *n* = *i* então vá para (8)
7) Execute o movimento, ou seja, insira os arcos *(i, j), (m, a) e (b, n)* e remova os arcos *(i, a), (b, j) e (m, n).* Faça P = P $\cup$ $\{a, b, n, m, j\}$ (Figura 14d).
8) Se *flag* = 0 então faça *j* = *i*, *flag* = 1 e escolha $i \in$ *p-vizinhos(j)* e vá para o passo (3) senão vá para o passo (1).

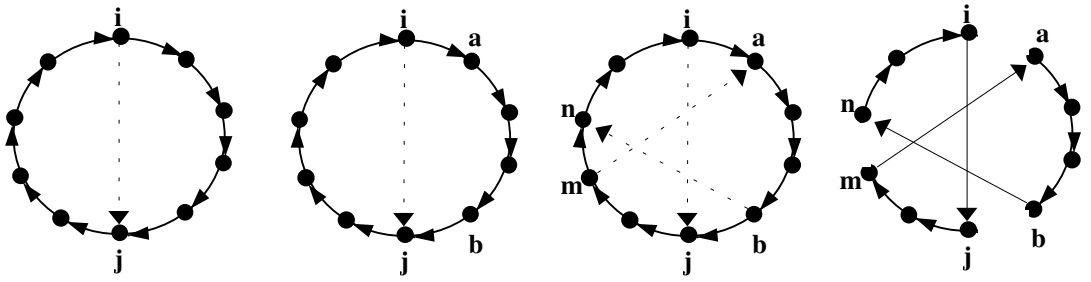


Figura 14: Procedimento RAI.

A escolha de um candidato a ser sucessor ou predecessor de *i* dentre os 5 mais próximos é feita aleatoriamente dentre as probabilidades que cada um tem de ser escolhido. As probabilidades foram escolhidas em testes empíricos e é feita da seguinte forma: a primeira vizinha tem 40% de chance de ser escolhida, a segunda 30%, a terceira 15%, a quarta 10% e a quinta 5% (Buriol et al. [1999]).

Note que o movimento só ocorre quando a somatória das distâncias dos 3 arcos que saem é maior do que a somatória das distâncias dos arcos que entram.

No caso do NWFSP as distâncias podem não refletir a real distância entre os vértices já que no NWFSP pode ocorrer um tempo de espera para satisfazer as datas de liberação e que não

são computados no algoritmo. Por conseqüência, a rota pode acabar sendo pior do que a que iniciou o movimento mesmo que $\Delta_2 < \Delta_1$. Para garantir que isso não ocorra, uma pequena modificação foi feita. Depois que a condição $\Delta_2 < \Delta_1$ é satisfeita, a função objetivo da nova rota é calculada e comparada com a original. Se houver melhora a recursividade continua, senão é escolhido um outro ponto crítico. O algoritmo é o mesmo que o descrito acima só que o item 7 sofre uma pequena modificação e o item (7a) é acrescentado.

7) Execute o movimento, ou seja, insira os arcos *(i, j), (m, a) e (b, n)* e remova os arcos *(i, a), (b, j) e (m, n)*.

7a) Calcule o custo da nova rota. Se o custo da nova rota for menor que o antigo então restaure a rota e vá para (1) senão faça $P = P \cup \{a, b, n, m, j\}$.

Além desses dois itens que modificam o algoritmo RAI original, o *s-vizinhos(i)* e *p-vizinhos(i)* passam a representar os vizinhos de *i* na solução em vez dos vizinhos mais próximos dentre todos os *n* vértices. Dada uma seqüência $\sigma = < 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15 >$, os *s-vizinhos(9)* são (11, 12, 13, 14, 15) e os *p-vizinhos(9)* são (7, 6, 5, 4, 3). 8 e 10 não são considerados vizinhos pois eles já são os vizinhos atuais e não alterariam a configuração da solução atual. Essa mudança faz com que o algoritmo não dependa de uma vizinhança estática que possa limitar o espaço de busca impedindo assim a convergência para um ótimo local. A desvantagem é que o espaço de busca, sendo maior, faz com que o algoritmo convirja mais lentamente.

2.4.2.2. *Recursive Guided Insertion* (RGI)

RGI é um procedimento recursivo e, assim como o RAI, também procura explorar os pontos críticos de uma solução de forma recursiva. Os pontos críticos são os pontos de *crossover,* de mutação e os pontos desconexos das seqüências quando geradas em um *crossover* (Tabela 1).

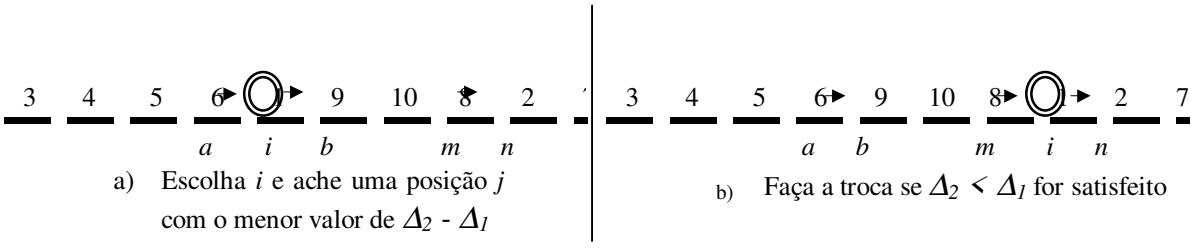
Os movimentos são de inserção de um ponto considerado crítico entre os outros vértices de uma rota ou entre as tarefas de uma seqüência. Nesse procedimento todos os pontos de inserção que podem melhorar a solução são considerados. A diferença entre este procedimento

para o RAI é que ele não trabalha com vizinhança e somente um ponto é reposicionado dentro da rota.

Esse procedimento surgiu com o intuito de se fazer mínimas perturbações na rota (movimentos que alterassem pouco a rota) devido a implicação que a restrição de datas de liberação traz ao problema. A implicação que essa restrição traz é que mesmo havendo uma diferença positiva entre arcos que saem e entram, o valor da função objetivo, quando recalculado, pode ser pior que o da solução original. Como o RAI além de reposicionar vértices pode também reposicionar trechos da rota a perturbação seria maior e mais movimentos seriam recusados.

O algoritmo RGI pode ser descrito da seguinte forma:

1) Seja F = custo(solução). Selecione um ponto crítico $i \in$ P (Conjunto de pontos críticos) e P = P – {i}. Se P for vazio então pare.
2) Faça a = *predecessor(i)* e b = *sucessor(i)* (Figura 15a).
3) Ache uma posição j tal que $1 \leq j \leq n$, $j \neq i$, calcule $\Delta_1 = d_{ai} + d_{ib} + d_{mn}$ e $\Delta_2 = d_{mi} + d_{in} + d_{ab}$, sendo m = vértice na posição j –1 e n = vértice na posição j (Figura 15b), reposicione i para essa posição, deslocando o vértice que estava nessa posição para a direita, se $\Delta_2 < \Delta_1$ para o menor $\Delta_2 - \Delta_1$ entre todas as posições j.
4) Calcule F_n = custo(solução nova). Se a condição em (3) for satisfeita e $F_n < F$, então faça P = P $\cup$ {a, b}
5) Vá para (1).



a) Escolha i e ache uma posição j com o menor valor de $\Delta_2 - \Delta_1$

b) Faça a troca se $\Delta_2 < \Delta_1$ for satisfeito

Figura 15: Procedimento RGI.

2.4.2.3. *Recursive Guided Insertion with Nearest Neighbor* (RGI/NN)

O RGI/NN é o mesmo algoritmo que o descrito anteriormente, exceto que em vez de considerar todas as posições para o reposicionamento de um ponto crítico somente os vizinhos mais próximos são considerados.

Para os movimentos serem efetuados, o algoritmo precisa de duas matrizes bidimensionais de n linhas e cinco colunas de vizinhos sucessores e predecessores. A matriz bidimensional *s-vizinhos(i)* contém os vértices sucessores a i e ordenados em ordem não decrescente de distância em relação a i. A matriz bidimensional *p-vizinhos(i)* contém os vértices predecessores a i e ordenados em ordem não decrescente de distância em relação a i.

O ponto crítico i é reposicionado ao lado do vértice sucessor ou predecessor que possa trazer o maior ganho na função objetivo.

O RGI/NN é o seguinte:

1) Seja F = custo(solução). Selecione um ponto crítico $i \in$ P (Conjunto de pontos críticos) e P = P – $\{i\}$. Se P for vazio então pare.
2) Faça a = *predecessor(i)* e b = *sucessor(i)* (Figura 15a).
3) Ache um vizinho j tal que $j \in$ *s-vizinhos(i)* $\cup$ *p-vizinhos(i)*, reposicione i como sucessor de j se vier de *p-vizinhos(i)* ou como predecessor de j se vier de *s-vizinhos(i)* se $\Delta_2 < \Delta_1$ para o menor $\Delta_2 - \Delta_1$. Sendo $\Delta_1 = d_{ai} + d_{ib} + d_{mn}$, $\Delta_2 = d_{mi} + d_{in} + d_{ab}$, m = novo predecessor de i e n = novo sucessor de i (Figura 15b).
4) Calcule F_n = custo(solução nova). Se a condição em (3) for satisfeita e $F_n < F$ então faça P = P $\cup$ $\{a, b\}$
5) Vá para (1).

2.4.2.4. Busca local de inserção

A busca de inserção procura reposicionar uma cidade *i* em *n*-1 posições sendo *n* o tamanho da instância. A inserção que trouxer maior benefício à função objetivo será a escolhida.

O algoritmo da busca de inserção é o seguinte:

1) Dado uma solução $x \in X \mid X$ conjunto de solução factíveis faça:
2) $x'' := x' := x,\ i := j := 0.$
3) Incremente *i.*
4) Incremente *j*.
5) Se a cidade *i,* quando inserida na posição *j*, melhorar *x* então atualizar *x´* com esse movimento.
6) Se $f(x') < f(x'')$, então $x'' := x'$
7) Se $j <= n$, então vá para passo 3 senão $x := x''$.
8) Se $i <= n$, então vá para passo 3.
9) Se *x* melhorou, então repita o procedimento acima; senão pare.

2.4.2.5. Busca local de trocas

Esta busca realiza trocas de forma ordenada de uma cidade com todas as outras. A troca que trouxer maior redução do custo será a escolhida. Considere a seqüência < 1 2 3 4 5 6 >. Primeiro a cidade 1 é trocada com todas as outras < 2 3 4 5 6 > depois a cidade 2 é trocada com < 3 4 5 6 > e assim por diante. Quando nenhuma troca melhorar a solução, então um mínimo local foi encontrado e o algoritmo pára.

2.4.3. Mutação

A mutação consiste em inserir ruído em uma solução e com isso proporcionar diversidade à população de indivíduos. Geralmente ela ocorre com uma pequena probabilidade, já que ruído

em excesso pode prejudicar a convergência do algoritmo.

Operadores de mutação atuam de forma aleatória na população, visando causar modificações não tendenciosas, que seriam causadas, por exemplo, se fosse empregado algum método heurístico que usasse conhecimento do problema com o intuito de melhorar a solução.

Os operadores estudados são o de inserção, o 3-Opt e o de trocas.

Inserção: A mutação de inserção escolhe uma posição *i* e insere o vértice que ocupa a posição *i* na posição *j*, sendo $i \neq j$, deslocando os vértices de *j* em diante para a direita se $i > j$ e a esquerda se $j < i$.

Trocas (Swap): A mutação de trocas escolhe duas posições *i* e *j* para $i \neq j$ e inverte a ordem dos vértices, ou seja o vértice da posição *i* vai para *j* e o de *j* vai para a posição *i*.

3-Opt: A mutação 3-Opt escolhe aleatoriamente um arco que entra e um arco que sai para conectar novamente os pontos desconexos. Ex. seja *(i, j)* o arco que entra; então deve-se achar aleatoriamente um arco *(m, n)* entre a rota *j* e *i* para sair e conectar com os pontos *a* e *b* que são o sucessor de *i* e o predecessor de *j*, tal que, *(n, a)* e *(b, m)* se tornam os novos arcos da rota (Figura 16).

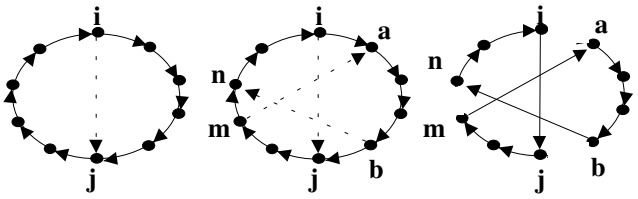


Figura 16: Movimento 3-Opt usado na mutação.

2.5. PSEUDOCÓDIGO

A estrutura apresentada aqui para um algoritmo memético pode variar de implementação à implementação e não é uma estrutura que tem que ser seguida à risca. Nessa seção será mostrado o algoritmo memético e depois algumas partes serão explicadas.

```
Inicialize(P)                                              //Inicialização
Para cada i ∈ P faça i = Busca_Local(i)
Para cada i ∈ P faça calcule função objetivo F_i = custo(i)
Repita
        Para j = 1 até #recombinações faça                 //Laço de recombinação
                S_par = SelecioneParaRecombinação(P)
                filho = Recombine(S_par)
                filho = Busca_Local(filho)
                F_filho = custo(filho)
                AcrescenteNaPopulação filho, P
        FimPara
        Para j = 1 até #mutações faça                      //Laço de mutação
                i = SelecioneParaMutação(P)
                aux = Mutação(i)
                aux = Busca_Local(aux)
                F_aux = custo(aux)
                AcrescenteNaPopulação aux, P
        FimPara
        Se diversidade(P) for falso então P = reinicialize(P)   //Diversidade
        Organiza(P)                                        //Reestruturação
Até critério_de_pararada for verdadeiro
```

2.5.1. Inicialização

A inicialização da população é feita usando uma heurística construtiva para gerar um indivíduo e um método aleatório para gerar os demais indivíduos. A heurística construtiva usada é a do vizinho mais próximo. A heurística do vizinho mais próximo escolhe um vértice inicial e constrói uma rota ligando sucessivamente um vértice a um outro vértice de menor distância dentre os que não fazem parte da rota.

2.5.2. Laço de recombinação

O laço de recombinação é responsável por escolher pais dentre a população para realizar a recombinação e gerar um filho. Esse filho gerado passa por um aprendizado através de uma busca local, um mecanismo de exploração da vizinhança. Depois que o filho evolui, ele é acrescentado à população original somente se não existir indivíduos semelhantes a ele, ou seja, se não houver indivíduos com o mesmo valor de função objetivo. Sempre que um indivíduo entra na população, um outro sai para manter o tamanho da população fixo. Esse critério para incluir indivíduos na população serve para garantir a diversidade da mesma.

2.5.3. Laço de mutação

O laço de mutação é responsável por selecionar indivíduos da população e realizar uma mutação criando um novo indivíduo. Quando um indivíduo é escolhido para sofrer mutação, uma cópia dele é feita e o operador de mutação é aplicado. A cópia, que sofreu mutação, passa pelo mesmo processo de aprendizado através da busca local. Depois que o indivíduo atingiu o seu ótimo local, ele é acrescentado à população no lugar do original, que deu origem a solução alterada, somente se não existir indivíduos com o mesmo valor de função objetivo.

2.5.4. Diversidade

A diversidade na população ocorre quando a maioria de seus indivíduos são distintos de acordo com algum critério. O critério utilizado nesta dissertação é em função do valor da função objetivo das soluções, que também é o *fitness* dos indivíduos. Indivíduos distintos são aqueles cujos valores da função objetivo são diferentes.

Quando a perda de diversidade ocorre, dizemos que os indivíduos de uma população possui valores de função objetivo iguais.

Como a comparação dos valores de função objetivo de todos os indivíduos de uma

população pode ter um alto custo computacional para determinar sua diversidade, outros critérios podem ser considerados nesse modelo de algoritmo memético. Critérios como: número de gerações que o algoritmo memético rodou, o número de gerações que um *pocket* ficou sem ser atualizado ou o número de novos indivíduos inseridos na população a cada geração do algoritmo memético.

Sempre que ocorrer perda de diversidade, a população sofre um processo de reinicialização de alguns ou todos os seus indivíduos. A manutenção da diversidade populacional é muito importante para o algoritmo memético ou qualquer metaheurística populacional, pois com a perda de diversidade o algoritmo tende a convergir mais lentamente para uma solução ótima global ou talvez nem convergir. Por usar uma população pequena, ao contrário de algoritmos genéticos que usam uma população maior, a manutenção da diversidade se torna muito importante. A população do algoritmo memético não pode ser muito grande devido ao uso da busca local que é responsável por cerca de 90% do tempo total de processamento do algoritmo.

2.5.5. Reestruturação

Ao final de cada geração, a população é reestruturada. Os *pockets* tem que ser atualizados, os líderes de cada subpopulação tem que ser atualizados e os subordinados tem que ser ordenados em ordem não decrescente do valor da função objetivo. O *pocket* troca de solução com o *current* sempre que o *current* tiver uma solução melhor que o *pocket*. O *pocket* dos líderes tem que ser sempre o melhor *pocket* de cada subpopulação de forma que o primeiro nó da árvore sempre terá a melhor solução.

2.5.6. Critério de Parada

O critério de parada utilizado nessa dissertação para o algoritmo memético é uma função relativa ao tamanho da população de agentes e relativa ao tamanho da instância. O algoritmo

também pára quando encontra uma solução ótima.

A Figura 17 descreve o comportamento da função 13**log*(13)**log*(*n*) como critério de parada para o algoritmo memético, onde 13 é o tamanho da população e *n* é o tamanho da instância. O crescimento logarítmico da função permite que o algoritmo rode um número grande de gerações para as instâncias de 17 à 55 tarefas, e um número de gerações não muito maior para instâncias maiores, em torno de 64 à 100 tarefas.

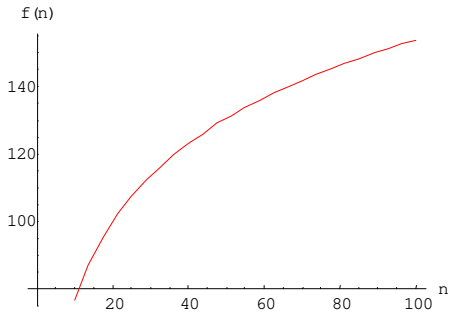


Figura 17: Critério de parada em função de 13**log*(13)**log*(**n**)

2.6. PLANO DE RECOMBINAÇÃO

O plano de recombinação é a estratégia usada para escolher indivíduos dentre a população para gerar novos indivíduos durante o processo de recombinação. A estratégia utilizada é a seguinte (veja Figura 10):

Current do líder ⇐ Recombine(*pocket* do filho 2, *pocket* do filho 3);

Current do filho 1 ⇐ Recombine(*pocket* do líder, *current* do filho 2);

Current do filho 2 ⇐ Recombine(*pocket* do filho1, *current* do filho 3);

Current do filho 3 ⇐ Recombine(*pocket* do filho2, *current* do filho 1).

O filho 1 é escolhido aleatoriamente entre os subordinados do líder da subpopulação, sendo assim filho1 pode ser o segundo subordinado, o terceiro ou o primeiro. Se filho1 for o segundo subordinado então o filho 2 será o terceiro e o filho 3 o primeiro. Esta estratégia foi escolhida como a melhor baseada em testes empíricos que avaliaram várias outras estratégias.

CAPÍTULO 3
ANÁLISE DE OPERADORES DE BUSCA LOCAL, RECOMBINAÇÃO, MUTAÇÃO, DIVERSIDADE E *FITNESS LANDSCAPE*

O estudo de *fitness landscape* tem sido de fundamental relevância para a compreensão da estrutura do problema e o comportamento dos algoritmos em tal estrutura. A estrutura de um problema pode ser definida pela representação genética utilizada no problema e os operadores de busca empregados nessa representação. *Landscapes* diferentes para um problema específico são criados por diferentes algoritmos, pois estes tendem a explorar o espaço de busca com movimentos distintos criando assim diferentes *landscapes* com convergências diferentes para o ótimo.

Vários pesquisadores estudaram a estrutura dos problemas através da análise de *fitness*. O pioneiro nesse estudo foi Sewall Wright [1932] com seu estudo da adaptabilidade de espécies biológicas ao meio ambiente. Outros estudos como o de Reeves [1998] para o problema de *flowshop*, Boese [1995] para o problema do caixeiro viajante e Merz & Freisleben [1998] para o problema de bipartição de grafos também contribuíram para o avanço do estudo de *fitness landscape* em problemas combinatoríais. Um trabalho importante que também surgiu no campo da biologia foi o de Weinberger [1990] que sugere um modelo matemático para o estudo das propriedades dos *landscapes*. Através desse modelo é possível estimar o ótimo global e a estrutura local do problema possibilitando assim um estudo mais exato do comportamento das heurísticas de busca aplicadas aos problemas combinatoríais.

Esse capítulo apresenta uma análise das buscas locais para o problema de *no-wait flowshop*, em seguida uma análise dos operadores de *crossover*, mutação, diversidade e por fim a análise de *fitness landscape*.

Todos os testes foram conduzidos usando instâncias com resultados ótimos conhecidos. A geração dessas instâncias é explicada no Capítulo 4. Essas instâncias variam de 17 a 100 tarefas e de 2, 5 e 10 máquinas. Para cada combinação de número de tarefas e máquinas foram geradas 4 instâncias, totalizando 168 instâncias. Devido ao grande número de instâncias, apenas as

instâncias de 17, 43, 70 e 100 tarefas para 10 máquinas serão consideradas nos testes realizados nesse capítulo.

3.1. ANÁLISE DE OPERADORES DE BUSCA LOCAL

Para analisar as 5 diferentes buscas locais e escolher a que melhor se adapta ao problema foram gerados aleatoriamente 1000 soluções iniciais para as buscas locais e comparado o ótimo local atingido por cada busca local para cada solução aleatória gerada.

Os pontos críticos usados para rodar as buscas RAI e RGI foram escolhidos aleatoriamente e são equivalentes a 25% do tamanho da instância. As buscas de inserção e trocas consideram todas as soluções do espaço de vizinhança.

Os gráficos abaixo mostram a porcentagem da melhor busca local encontrada em relação às outras quatro. A busca local que obteve os melhores resultados é a busca *Recursive Arc Insertion* (RAI). As porcentagens indicam o número de soluções onde a busca local RAI obteve ótimos locais de melhor qualidade em relação às outras. As porcentagens representam resultados médios gerados para as 4 instâncias de cada combinação de tarefas e máquinas.

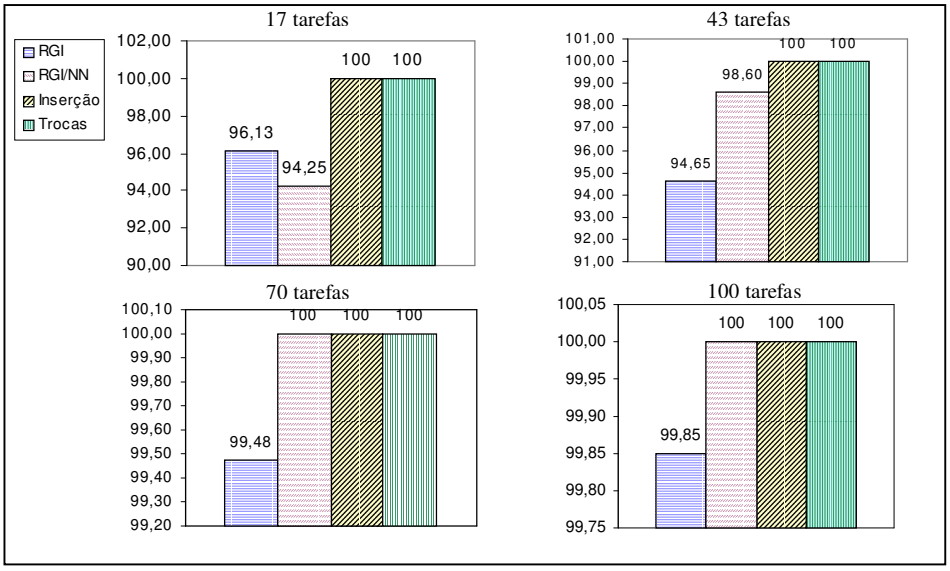


Figura 18: Análise da busca RAI em relação a outras buscas locais para 1000 soluções geradas aleatoriamente.

A Figura 18 mostra nitidamente a qualidade da busca RAI em relação às outras buscas locais. Conforme aumenta o tamanho da instância, a busca RAI se mostra ainda mais superior às outras, sendo até 100% superior em relação as buscas de inserção e de trocas para 17, 43, 70 e 100 tarefas. Em relação às buscas de inserção guiadas por pontos críticos, a busca RAI também se mostra nitidamente superior ficando em 100% para as instâncias de 70 e 100 tarefas e acima de 94% para 17 e 43 tarefas.

3.2. ANÁLISE DE OPERADORES DE *CROSSOVER*

Para escolher o operador de *crossover* que melhor se adaptasse ao problema de *no-wait flowshop* foi utilizada a busca local RAI no processo evolutivo do algoritmo memético. Esse procedimento foi repetido dez vezes para todas as 168 instâncias e as médias totais foram comparadas.

A Tabela 2 mostra os resultados médios obtidos por diferentes operadores de *crossover*. A população inicial para todos é a mesma pois o mesmo método de geração foi utilizado. A pequena variação de qualidade entre eles se deve ao grande número de instâncias e ao método de busca local ser muito eficiente, escondendo possíveis deficiências do método de *crossover*. **Pop.Inicial** representa o desvio médio da população inicial em relação ao ótimo, **Qual.** representa o desvio médio do algoritmo memético em relação ao ótimo, **Gerações** representa o número médio de gerações para encontrar o ótimo, **Otm/#exe** representa o número médio de vezes que o algoritmo memético encontrou o ótimo, **Tempo** é o tempo computacional médio para achar o ótimo e **CPU BL** é a porcentagem média de tempo gasto com a busca local.

Tabela 2: Resultados computacionais do algoritmo memético com busca local RAI usando diferentes métodos de *crossover*.

Crossover	Pop.Inicial (%)	Gerações	Qual.	Otm/#exe	Tempo (seg.)	CPU BL (%)
DPX	29,04	6,34	0,0431	9,82/10	5,15	98,2
OX	29,04	3,08	0,0016	9,99/10	2,48	98,22
SAX	29,04	5,17	0,0139	9,92/10	3,36	98,08
PMX	29,04	2,50	0,0008	9,99/10	2,62	98,69

O *crossover* PMX chega a ser quase 20 vezes melhor que o SAX e o DPX e duas vezes melhor que o OX para todas as instâncias (Tabela 2). Essa superioridade do PMX pode ser explicada pela sua característica de preservar a posição absoluta das tarefas na seqüência e pelos pontos críticos resultantes da aplicação desse operador, pontos tais que serão utilizados pela busca local para explorar o espaço de busca gerado pelo problema. Em contraste, os outros operadores de *crossover* procuram manter a distância relativa entre as tarefas e consequentemente não apresentam o mesmo desempenho que o *crossover* PMX .

3.3. ANÁLISE DE OPERADORES DE MUTAÇÃO E ANÁLISE DE DIVERSIDADE

O operador de mutação escolhido foi aquele que obteve melhor qualidade nos testes empíricos realizados usando a busca RAI e o *crossover* PMX. Os testes foram conduzidos com todas as instâncias e a probabilidade de realizar mutação em um indivíduo variou entre [5%,25%] em incrementos de 5. O operador 3-Opt com probabilidade de 5% foi o que obteve melhor qualidade e será o operador usado nos resultados computacionais.

Para manter a diversidade foi usado um critério de reinicialização dos agentes intermediários. A reinicialização dos agentes subordinados à raiz da árvore ternária ocorre a cada *n* gerações que a solução *current* não melhora a solução. Vários valores de *n* foram testados.

Esse critério, quando introduzido no algoritmo memético para o problema NWFSP, trouxe uma queda de qualidade e um atraso em algumas instâncias para obter a solução ótima. Portanto, o algoritmo não usará nenhum critério de diversidade além daquele que introduz novos indivíduos que não existam na população.

3.4. ANÁLISE DE *FITNESS LANDSCAPE*

A análise de *fitness landscape* tem por objetivo a análise do comportamento de determinados mecanismos que exploram o espaço de soluções de um problema. Essa análise ajuda a prever a performance do método e a criar novas estratégias de exploração do espaço de

soluções que sejam mais efetivas na resolução do problema. A técnica utilizada pela maioria dos pesquisadores envolve a geração de várias soluções e a plotagem do *fitness* ou custo dessas soluções vs. a distância da solução para uma solução ótima ou a melhor encontrada. Essas soluções são geralmente geradas por operadores de busca local. Uma solução aleatória é gerada e um operador de busca local é aplicado a essa solução. Quando a solução atingir um ótimo local ela é plotada no gráfico.

Ao contrário do método usado por outros pesquisadores, a análise de *fitness landscape* nessa dissertação foi feita durante todo o processo evolutivo do algoritmo incluindo operadores de *crossover*, busca local e mutação, já que esses métodos também participam da exploração do espaço de busca no algoritmo memético e o comportamento deles na exploração do espaço de soluções é de extrema importância para a geração de novas estratégias de busca.

As métricas usadas para calcular a distância entre soluções influenciam bastante a geração de *landscapes*, pois cada métrica define um novo *landscape* que pode ser ou não apropriado ao problema estudado.

A princípio, a distância entre duas soluções x_i e x_{i+1} em um *landscape* deveria ser medido pelo número de movimentos que o operador de busca local levaria para criar uma solução x_{i+1} a partir de x_i. Como, infelizmente, não existe nenhum algoritmo polinomial que calcule essa distância, foram então criadas várias métricas que procuram se aproximar dessa distância. A seguir algumas métricas são expostas, bem como sua aplicação prática.

Hamming: geralmente utilizada em representações binárias. A distância de Hamming calcula o número de bits que precisam ser mudados para transformar uma solução x em z. Seja x = <11100> e z = <01010> então $d_h(x,z) = 3$.

Acumulada: geralmente utilizada em problemas de sequenciamento onde a posição absoluta das tarefas é mais importante do que a posição relativa (Reeves [1997]). A distância acumulada é a soma do módulo da diferença entre o número de posições que uma tarefa está distante de uma outra tarefa em uma mesma solução e o número de posições entre as mesmas tarefas em uma outra solução para todos as combinações de duas tarefas. Também conhecida como distância de Johnson ou distância baseada em posição.

Adjacente: utilizada em problemas cuja distância relativa entre os elementos de uma solução é mais importante. Utilizada por Boese [1995] para calcular a distância entre soluções do

caixeiro viajante (TSP). A distância adjacente conta o número de arestas comuns em duas soluções.

Precedência: essa métrica é também utilizada onde a distância absoluta de um elemento da solução é mais importante que a distância relativa. Ela pode ser usada em substituição à distância acumulada, apesar de se levar mais tempo computacional para calculá-la do que a distância acumulada. Ela mede o número de vezes que um elemento precede um outro em ambas as soluções e esse número é subtraído de $n(n\text{-}1)/2$.

Todas as métricas foram testadas para o problema do *no-wait flowshop*, inclusive a distância de Hamming modificada para representações inteiras. A métrica adjacente, bem como a métrica de Hamming, não se mostraram adequadas para o problema do *no-wait flowshop,* havendo pouca ou nenhuma correlação entre distância e custo. Por sua vez, as métricas acumulada e a de precedência mostraram resultados muito similares e de boa correlação entre custo e distância do ótimo global. A métrica acumulada foi a escolhida devido à sua praticidade e rapidez para implementar e gerar os gráficos e, portanto, será a métrica utilizada nos resultados de análise de *fitness landscape*.

A Tabela 3 mostra os resultados médios da distância das soluções em relação ao ótimo e entre elas mesmas. Esses resultados mostram claramente que a solução ótima está mais perto de outras soluções ótimas locais do que elas estão perto delas mesmas, o que sugere uma característica globalmente convexa (Figura 19), onde os ótimos locais ocorrem agrupados em uma mesma região. Essa mesma característica foi observada por Boese et al. [1994]. Isso sugere que métodos que utilizam pontos de partida perto desses ótimos locais tendem a gerar boas soluções. O uso de métodos evolutivos como algoritmos genéticos ou meméticos sugere que a partir de um *crossover* que preserve a informação dos pais se gere uma prole que seja um bom ponto de partida para um método de melhoria.

Figura 19: Característica globalmente convexa do espaço de solução.

As Tabelas 3a, 3b e 3c mostram para cada instância a distância média de uma solução ótima local em relação à outra, a distância média das soluções ótimas locais em relação à solução ótima global, o erro percentual médio das soluções ótimas locais em relação ao ótimo global e a distância máxima que de uma solução em relação ao ótimo global.

Tabela 3a: Análise de distância acumulada para instâncias de 2 máquinas.

Instância	Distância Ótimo	Distância Outros	Média Custo (%)	Dist. Máx. Ótimo
br17	24,97	37,33	9,01	144,00
ftv33	64,17	98,60	5,79	549,00
ftv35	97,23	143,29	5,96	599,00
ftv38	73,78	118,51	6,02	694,00
p43	77,03	119,43	1,64	784,00
ftv44	91,87	147,77	4,92	881,50
ftv47	100,99	171,73	4,95	960,50
ry48p	119,81	169,53	3,83	1068,50
ft53	111,32	186,39	5,43	1161,00
ftv55	153,99	249,63	5,11	1434,00
ftv64	194,09	310,27	4,59	1877,00
ftv70	197,62	324,56	3,50	1880,00
ft70	179,06	309,36	4,39	2197,00
kro124p	224,90	411,86	2,72	3964,00

Tabela 3b: Análise de distância acumulada para instâncias de 5 máquinas.

Instância	Distância ótimo	Distância outros	Média Custo (%)	Dist. Máx. Ótimo
br17	20,61	32,32	6,56	125,00
ftv33	35,14	59,57	4,61	461,00
ftv35	41,51	67,14	5,01	572,00
ftv38	52,37	87,17	4,37	705,00
p43	52,16	88,59	2,21	750,50
ftv44	45,92	80,37	3,69	794,00
ftv47	59,11	105,44	4,25	930,00
ry48p	70,26	118,21	3,99	1002,50
ft53	56,67	104,14	4,72	1030,00
ftv55	132,69	198,44	3,95	1290,00
ftv64	133,42	230,27	4,57	1727,00
ftv70	107,71	191,50	2,77	2112,50
ft70	147,62	255,86	3,69	1926,50
kro124p	160,79	302,33	2,79	3965,50

Tabela 3c: Análise de distância acumulada para instâncias de 10 máquinas.

Instância	Distância ótimo	Distância outros	Média Custo (%)	Dist, máx, ótimo
br17	19,43	29,67	6,48	122,50
ftv33	24,41	43,33	3,32	410,50
ftv35	35,66	52,67	3,25	485,00
ftv38	32,94	58,03	3,68	570,50
p43	34,24	59,58	2,00	575,50
ftv44	42,73	75,41	3,34	690,50
ftv47	50,03	90,65	3,51	808,50
ry48p	51,46	83,58	2,97	875,00
ft53	36,98	69,79	3,19	990,00
ftv55	79,54	133,70	2,69	1007,50
ftv64	91,75	156,62	3,45	1580,50
ftv70	83,53	151,23	2,31	1880,50
ft70	78,16	142,20	2,56	1617,00
kro124p	137,66	254,97	2,12	4165,50

Os resultados das tabelas 3a, 3b e 3c mostram que a distância máxima é muito grande em relação à distância média, sendo em alguns casos 100 vezes maior. Devido a essa disparidade, principalmente para a instância **kro124p**, algumas soluções, cujo *fitness* ou a distância em relação ao ótimo é muito grande, foram cortadas do gráfico de *fitness landscape* para facilitar a visualização do comportamento da maioria dos outros pontos no gráfico. Devido ao grande número, apenas a análise das instâncias **ft70** e **kro124p** serão mostradas. Para essas, todo o processo evolutivo gera respectivamente 1860 e 2003 ótimos locais e como nem todos os ótimos locais são distintos então o gráfico pode ter menos pontos.

A Figura 20 mostra duas possíveis configurações de *fitness landscape*. O eixo *y* é o custo normalizado, ou seja, o custo de uma solução menos o custo da solução ótima. O eixo *x* é uma das métricas descritas acima, no caso do problema de *no-wait flowshop* a métrica utilizada é a distância acumulada.

A Figura 20a mostra uma configuração de *landscape* onde existe uma correlação entre distância e custo normalizado, ou seja, conforme a distância ao ótimo aumenta o custo normalizado também aumenta. Essa configuração mostra também uma convergência, ou um caminho, em direção ao ótimo, o que é considerada ideal para os problemas de otimização.

A Figura 20b mostra uma configuração de *landscape* onde a correlação praticamente não existe e a convergência para o ótimo não é visível. Esse tipo de configuração mostra a incapacidade do método utilizado para a exploração do espaço de busca em encontrar uma direção ao ótimo, sendo que o algoritmo explora uma região afastada do ótimo ou que não inclui a solução ótima.

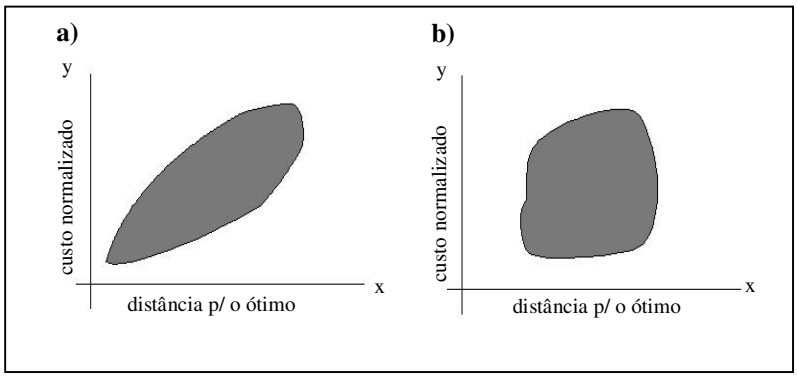


Figura 20: Possíveis configurações de *fitness landscape.*

A Figura 21 mostra a análise de *fitness landscape* para as 4 instâncias geradas com 70 tarefas e 2 máquinas. Cada instância possui parâmetros para geração de tempos de processamento bem como para geração de tempos de preparação diferentes. Mais detalhes sobre a geração dessas instâncias podem ser encontrados no Capítulo 4.

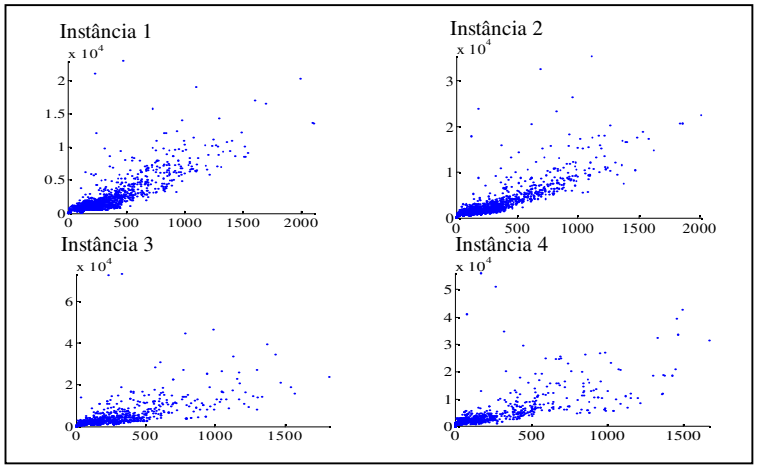


Figura 21: *Fitness Landscape* para 70 tarefas (ft70) e 2 máquinas.

As instâncias 1 e 2 mostram uma forte correlação entre custo e distância e uma convergência suave para o ótimo global. O caminho em direção ao ótimo é visível e a maioria dos pontos estão concentrados perto do ótimo.

A instância 3 também possui uma correlação dos pontos em direção ao ótimo, apesar de não ser visível esse caminho devido a um ponto com custo normalizado alto perto de *y* igual a 65000 e *x* igual a 500.

A instância 4 mostra a maioria dos pontos concentrada perto do ótimo, o que garante uma convergência rápida para o ótimo. Essa instância também mostra um concentração de pontos perto de custo normalizado 5000 e distância 200, o que pode causar que o algoritmo fique preso sem evoluir pois existe um desvio da convergência para o ótimo.

As instâncias 2, 3 e 4 da Figura 22 mostram uma rápida convergência para o ótimo. A diferença entre distância média entre os ótimos locais e a distância média entre eles e o ótimo global é pequena, mostrando que o algoritmo memético tem maior facilidade para convergir para instâncias com essas características. A instância 1 mostra um desvio, perto de custo normalizado 1000 e distância 300, que pode causar com que o algoritmo fique preso nessa região e não alcance o ótimo global.

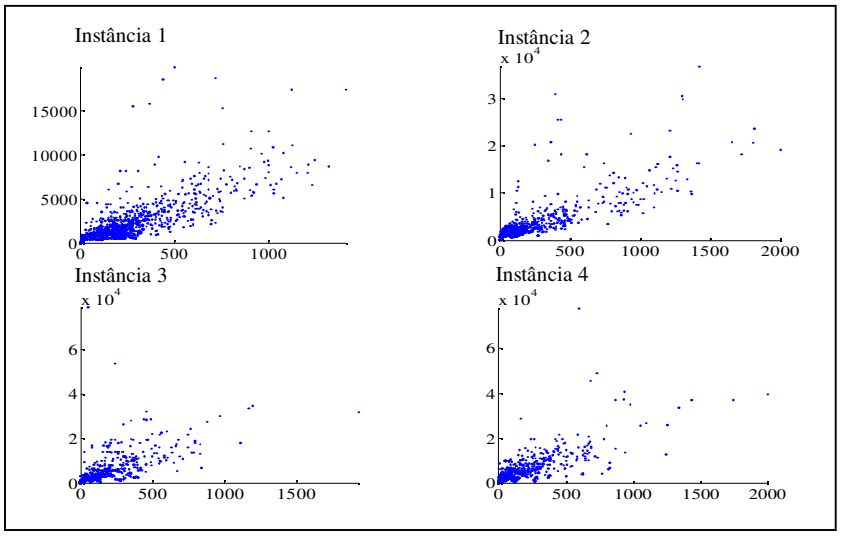


Figura 22: *Fitness Landscape* para 70 tarefas (ft70) e 5 máquinas.

Os gráficos da Figura 23 para instâncias 2, 3 e 4 mostram a ocorrência de menos pontos de ótimos locais em relação à instância 1, assim como na Figura 22. A convergência é rápida em

direção ao ótimo e a ocorrência de menos ótimos locais para as instâncias 2, 3 e 4 pode ocorrer devido ao grande número de bases de atração para os ótimos locais. Isso pode ser bom ou ruim, dependendo do método que explora esse espaço de soluções. Pode ser bom, no sentido de uma convergência rápida para o ótimo global, mas pode ser ruim, pois dependendo do método de busca, o movimento usado pode não conseguir sair das bases de atração de um ótimo local e o algoritmo pode ficar preso em algum ótimo local, não convergindo, assim, para o ótimo global.

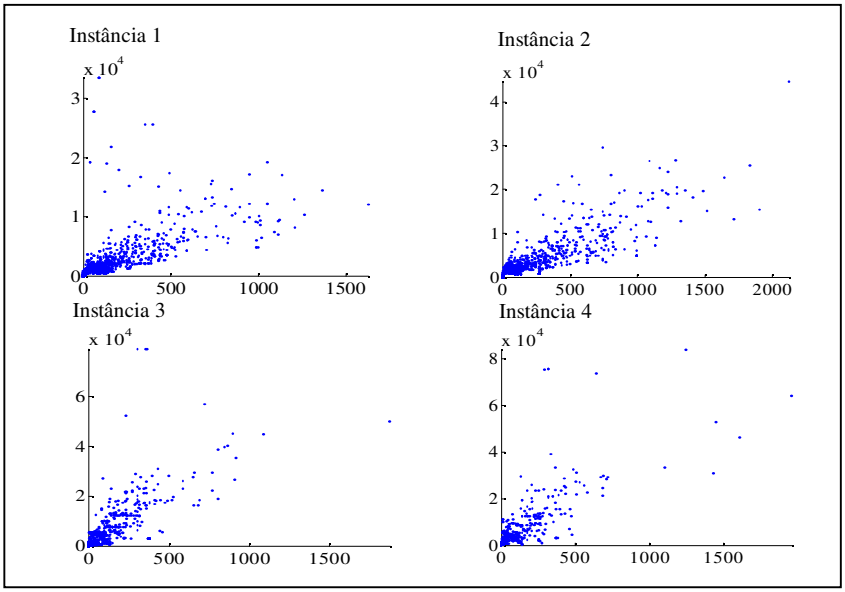


Figura 23: *Fitness Landscape* para 70 tarefas (ft70) e 10 máquinas.

A Figura 24 mostra a análise de *fitness landscape* para 100 tarefas e 2 máquinas. A convergência do algoritmo, ou o caminho para o ótimo global, para todas as instâncias é visível. A instância 3 possui pontos distantes e com custo normalizado perto do ótimo global causando a inclinação para a esquerda do caminho para o ótimo global. A instância 4 mostra um pequeno desvio que pode impedir a convergência do algoritmo memético.

Na Figura 25, para 5 máquinas, os ótimos locais, mais uma vez, continuam bastante perto do ótimo global, a ocorrência de ótimos locais não é pequena como aparenta devido a distância máxima e o custo normalizado para o ótimo global serem altos, e as instâncias 3 e 4 mostram ser mais fáceis que as 1 e 2 pois os ótimos locais estão mais próximos do ótimo global.

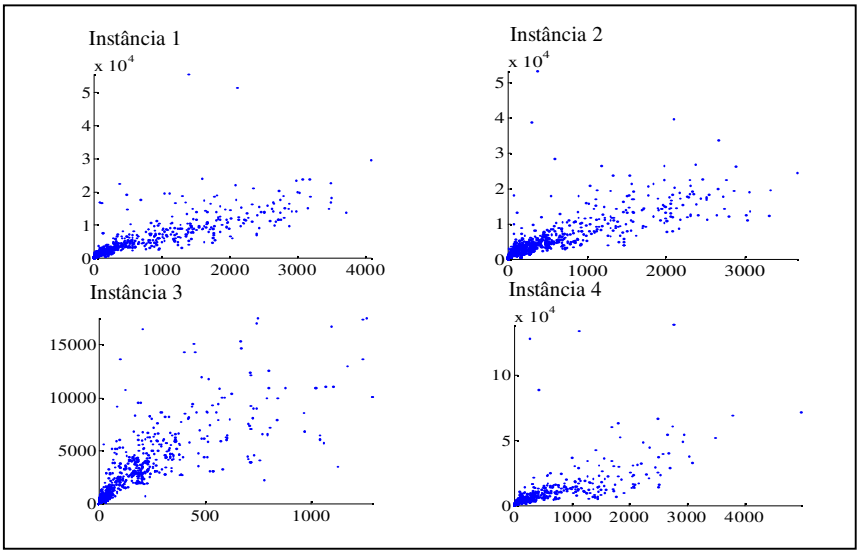


Figura 24: *Fitness Landscape* para 100 tarefas (kro124p) e 2 máquinas.

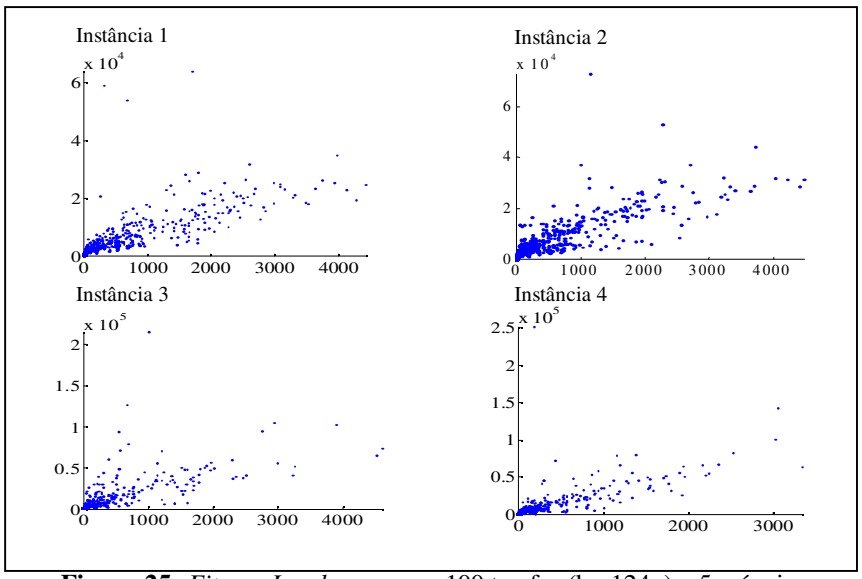


Figura 25: *Fitness Landscape* para 100 tarefas (kro124p) e 5 máquinas.

A Figura 26 mostra um vale na distância 200 para a instância 1. Para as instâncias 1 e 2, a maioria dos ótimos locais estão a menos de 2% do ótimo global enquanto que para as instâncias 3 e 4 a maioria dos ótimos locais estão a menos de 0,6% do ótimo global mostrando novamente a facilidade do algoritmo memético para as instâncias do tipo 3 e 4.

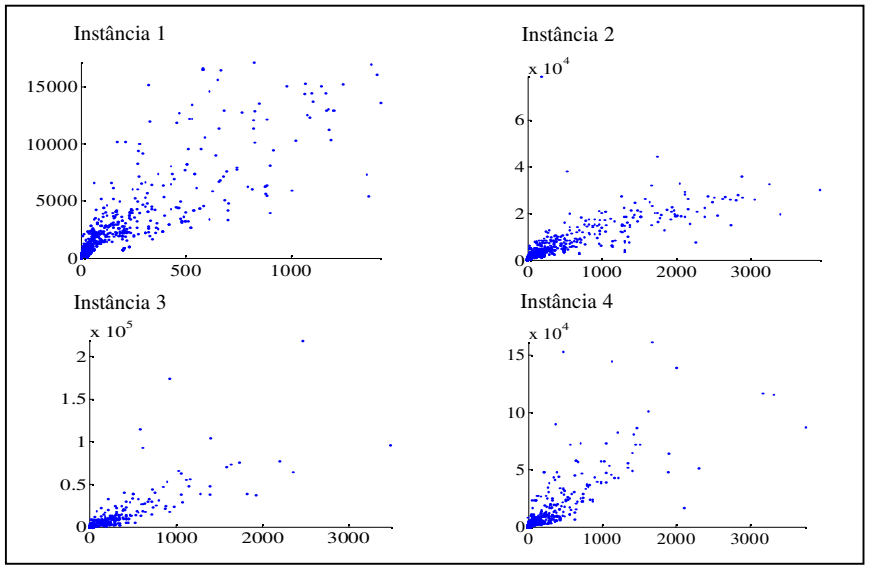


Figura 26: *Fitness Landscape* para 100 tarefas (kro124p) e 10 máquinas.

CAPÍTULO 4
RESULTADOS COMPUTACIONAIS

Os testes foram feitos usando 2 conjuntos de instâncias: 1) geradas aleatoriamente e 2) instâncias geradas a partir de instâncias conhecidas na literatura para o problema do caixeiro viajante assimétrico. Para as instâncias geradas aleatoriamente, os resultados do algoritmo memético são comparados aos resultados de um limitante inferior e também aos de um algoritmo construtivo. Para as instâncias geradas a partir do caixeiro viajante assimétrico, onde o ótimo global é conhecido, o resultado do algoritmo memético é comparado ao ótimo global e ao resultado obtido pela heurística construtiva. Os operadores utilizados no algoritmo memético são o crossover PMX, a busca local RAI e a mutação 3-Opt.

Neste capítulo, serão apresentados a obtenção de dois limitantes inferiores para o caixeiro viajante com datas de liberação, uma heurística construtiva e instâncias com resultados ótimos para o problema do *no-wait flowshop*.

4.1. ALGORITMO COMPARATIVO

Nesta dissertação uma heurística construtiva, *Best Insertion Heuristic* (BIH), proposta por Bianco et al. [1999] e dois limitantes inferiores, *Lower Bound 1* (LB1) e *Lower Bound 2* (LB2), também propostos por Bianco et al serão usados para comparar a qualidade e a performance do algoritmo memético para instâncias cujo ótimo global não é conhecido.

4.1.1. Limitantes Inferiores

Os limitantes inferiores são obtidos através da resolução em tempo polinomial da relaxação *lagrangiana* do modelo matemático do caixeiro viajante com datas de liberação para

visitar uma cidade (ATSP-RT).

Para melhor entender o LB1 e LB2, o modelo da ATSP-RT é apresentado a seguir.

Seja a variável x_{ij} uma variável de decisão, tal que $x_{ij} = 1$ se o vértice j é visitado imediatamente depois do vértice i e $x_{ij} = 0$ caso contrário; seja t_i uma variável que representa o tempo quando o vértice i é visitado, s uma variável que representa o tempo total de espera para visitar os vértices, r_i a data de liberação da cidade i, S é um subgrafo de G em que |S| representa o número de vértices desse subgrafo e V é o conjunto de vértices do grafo G.

(P)

Função Objetivo : $\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} + s$

Sujeito a :

$$t_i + \sum_{j \in V} c_{ij} x_{ij} - \sum_{k \in V} \sum_{j \in V} c_{kj} x_{kj} - s \leq 0, \qquad i \in V \qquad (1)$$

$$r_i - t_i \leq 0, \qquad i \in V \qquad (2)$$

$$(c_{ij} + T_{ij}) x_{ij} + t_i - t_j - T_{ij} \leq 0, \qquad i \in V, j \in V \backslash \{0\}, j \neq i \qquad (3)$$

$$\sum_{i \in V} x_{ij} = 1, \qquad j \in V \qquad (4)$$

$$\sum_{j \in V} x_{ij} = 1, \qquad i \in V \qquad (5)$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \qquad \forall S \subset V, S \neq \ddot{o} \qquad (6)$$

$$t_0 = r_0 = 0 \qquad (7)$$

$$s_{\min} \leq s \leq s_{\max} \qquad (8)$$

$$x_{ij} \in \{0,1\} \qquad i \in V, j \in V, i \neq j \qquad (9)$$

$$t_i \geq 0 \qquad i \in V \qquad (10)$$

A restrição (1) assegura que o tempo para visitar uma cidade j sucessora a i não seja maior que o tempo gasto para fazer a rota, para toda cidade $i \in$ V. As restrições (2) e (3) garantem que uma cidade só vai ser visitada depois que satisfazer a data de liberação para visitas. A variável T_{ij} é escolhida de forma que torne a restrição (4) verdadeira sempre que $x_{ij} = 0$. As restrições (5), (6) e (7) são as mesma do ATSP explicadas no Capítulo 1. A restrição (8) afirma que o tempo total esperado tem que estar entre um valor mínimo e máximo. Essa última restrição não é necessária

para a formulação do problema, mas ela ajuda na obtenção da solução do problema relaxado que vai gerar o limitante inferior.

4.1.1.1. *Lower Bound 1* (LB1)

O limitante inferior estudado aqui é baseado em uma relaxação *lagrangeana* do modelo do problema (P) sugerido acima. As restrições (1) e (2) são substituídas por uma restrição *surrogate* (2').

$$r_i + \sum_{j \in V} c_{ij} x_{ij} - \sum_{k \in V} \sum_{j \in V} c_{kj} x_{kj} - s \leq 0, \qquad i \in V \qquad (2')$$

A restrição (2') é dualizada, a restrição (3) é eliminada e as variáveis inteiras são relaxadas resultando no problema (LP).

(LP)

$$w = \max_{\lambda} L(\lambda) = \max_{\lambda} \left\{ \min_{x,\,s} \left\{ (1 - \sum_{k \in V} \lambda_k) s + \sum_{i \in V} \sum_{j \in V} (1 - \sum_{k \in V \setminus \{i\}} \lambda_k) c_{ij} x_{ij} \right\} + \sum_{k \in V} \lambda_k r_k \right\}$$

Sujeito a :

$$\sum_{i \in V} x_{ij} = 1, \qquad j \in \mathrm{V} \qquad (4)$$

$$\sum_{j \in V} x_{ij} = 1, \qquad i \in \mathrm{V} \qquad (5)$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \qquad \forall \mathrm{S} \subset \mathrm{V}, \mathrm{S} \neq \phi \qquad (6)$$

$$s_{\min} \leq s \leq s_{\max} \qquad (8)$$

$$x_{ij} \geq 0 \qquad i \in \mathrm{V},\ j \in \mathrm{V},\ i \neq j \qquad (9')$$

$$\lambda_k \geq 0 \qquad k \in \mathrm{V} \qquad (11)$$

Seja λ a variável *dual* do problema relaxado. Então, para o problema acima, *w* é otimizado quando, para um dado λ, existe um *x'* e um *s'* que resolve LP. Para um dado λ cujo *x'* e *s'* são ótimos no problema (LP), considere o Problema Lagrangiano Restrito (RLP), que para

um dado *x'* e *s'* ótimos em (LP) existe um λ que maximiza (RLP).

Para o modelo (RLP) abaixo, considere *u* e *v* os vetores das variáveis duais associadas às restrições (4) e (5), *y* o vetor das variáveis duais associadas à restrição de eliminação de subrotas (6) e Q_{ij} o subconjunto da restrição de eliminação de subrotas (6) ao qual a variável x_{ij} tem coeficiente diferente de zero. As restrições (13) e (14) forçam *u*, *v*, *y* e λ a satisfazer a folga complementária e a manter factibilidade do problema *dual*.

(RLP)

$$R(x',s') = \max_{\lambda}\left\{(1-\sum_{k\in V}\lambda_k)s' + \sum_{i\in V}\sum_{j\in V}(1-\sum_{k\in V\setminus\{i\}}\lambda_k)c_{ij}x'_{ij} + \sum_{k\in V}\lambda_k r_k\right\}$$

Sujeito a :

$$u_i + v_j + \sum_{l\in Q_{ij}} y_l + c_{ij}\sum_{k\in V\setminus\{i\}}\lambda_k \begin{cases} = c_{ij} & \text{if } x'_{ij} > 0 \quad i \in V, \\ \leq c_{ij} & \text{senão}, \quad j \in V, \end{cases} \quad j \neq i \qquad (12)$$

$$1-\sum_{k\in V}\lambda_k \begin{cases} \geq 0 \text{ if } s' = s_{min} \\ \leq 0 \text{ if } s' = s_{max} \end{cases} \qquad (13)$$

$$y_l \geq 0, \qquad l \in \bigcup_{ij} Q_{ij} \qquad (14)$$

$$\lambda_k \geq 0, \qquad k \in V \qquad (11)$$

O algoritmo LB1 repete a resolução de $L(\lambda)$ e $R(x',s')$ até quando nenhuma melhoria ocorrer no limitante inferior.

Algoritmo LB1

$\lambda = 0$; *melhorLB = -1; LB1 = 0*
Enquanto *LB1 > melhorLB* faça
    melhorLB = LB1
    Calcule $L(\lambda)$ e obtenha *x'* e *s'*
    Calcule *R(x', s')* e obtenha λ'
    LB1 = R(x', s')
    Se *LB1 > melhorLB1* então $\lambda = \lambda' + \Delta(\lambda')$
Fim do Enquanto

$\Delta(\lambda')$ é um pequena variação de λ' na direção de $R(x', s')$.

4.1.1.2. *Lower Bound 2* (LB2)

LB2 é baseada na relaxação da matriz de distância do ATSP-RT. Para cada vértice i a distância para qualquer vértice é igual a $d_i = \min_{j \in V \setminus \{i\}}[c_{ij}]$ e o ATSP-RT é resolvido em O($n \log n$) através da regra *earliest start time* (EST). EST visita todos os vértices a partir do vértice 0 em ordem não decrescente das datas de liberação para visitar as cidades.

4.1.2. Heurística construtiva - *Best Insertion Heuristic* (BIH)

A heurística construtiva proposta por Bianco et al. [1999] procura inserir tarefas em uma seqüência parcial já existente. O algoritmo é o seguinte:

Algoritmo BIH

$\sigma_0 = \varnothing$; $U = J$; $k = 0$
Enquanto $U \neq \varnothing$ faça
    Escolha uma tarefa $j \in U$ que quando inserida na seqüência σ_k em uma posição h traga o menor aumento de custo .
    Insira tarefa j na posição h da seqüência σ_k e faça σ_{k+1} ser a nova seqüência.
    Faça $U = U\{j\}$; $k = k + 1$
Fim do Enquanto.

O algoritmo pode ser executado em O(n^3), pois para escolher uma tarefa entre as não escalonadas e achar uma posição para inseri-la leva O(n^2) n vezes para o laço principal. Se o tempo de início de cada tarefa já escalonada for guardado em um vetor, então o custo de inserção de uma tarefa pode ser determinado em O(1). Depois que a melhor tarefa entre as não escalonadas for escolhida e a melhor posição de inserção for encontrada, o custo de inserir a nova tarefa na seqüência é no pior dos casos n, pois o vetor de tempos de início precisa ser atualizado para a próxima tarefa a ser inserida.

4.2. GERAÇÃO DE INSTÂNCIAS ALEATÓRIAS

As instâncias aleatórias para os quais serão calculados os limitantes LB1 e LB2 foram geradas para m = 2, 5 e 10 máquinas e n = 10, 50 e 100 tarefas. As datas de liberação de tarefas estão uniformemente distribuídas entre [1, R_{max}] onde R_{max} = 50, 100, 200, 300, 400 e 500. Instâncias com R_{max} baixo leva a situações onde várias tarefas ficam disponíveis no chão de fábrica ao mesmo tempo.

Os tempos de processamento p_i^k e os tempos de preparação s_{ij}^k foram escolhidos entre uma distribuição uniforme com intervalos [5,10] e [1,5].

Para cada combinação possível de parâmetros, máquinas, tarefas e data de liberação, foram geradas um máximo de 5 instâncias. Os resultados da Tabela 5 mostram a média dessas cinco instâncias e o pior resultado obtido entre as cinco instâncias que aparece entre parênteses (Qual.1 e Qual.2).

Para cada instância o algoritmo memético foi executado 10 vezes e o melhor resultado obtido em cada execução é comparado com o max(LB1,LB2). Os testes para as instâncias aleatórias foram feitos num computador AMD-Duron 600MHz e 128 MB de memória.

Nas Tabelas 4 a 6, **#Máq**. é o número de máquinas da instância, **#Tar.** é o número de tarefas da instância, $\mathbf{R_{max}}$ é o valor usado para a geração das datas de liberação, **Pop.Inicial** representa o desvio médio da população inicial em relação ao limitante inferior, **Qual.1** representa o desvio médio do algoritmo memético em relação ao limitante inferior, **Qual.2** representa o desvio médio da heurística construtiva em relação ao limitante inferior, **#Ger.** representa o número médio de gerações para encontrar o ótimo, **Otm/#exe** representa o número de vezes que o algoritmo memético encontrou o ótimo, **Tempo** é o tempo computacional médio para achar o ótimo e **CPU BL** é a porcentagem média de tempo gasto com a busca local.

Tabela 4: Resultados computacionais do algoritmo memético comparado a limitantes inferiores e a heurística BIH aplicado às instâncias experimentais com 2 máquinas.

#Máq.	#Tar.	R_{max}	IniPop (%)	#Ger.	Qual.1 (%)	Qual.2 (%)	Tempo (seg.)	CPU BL (%)
2	10	50	6,70	76,00	3,08(5,04)	5,23(8,74)	0,75	76,10
2	10	100	8,84	76,00	6,65(8,33)	7,78(11,67)	0,66	78,16
2	10	200	0,96	76,00	0,96(2,78)	1,41(2,78)	0,66	75,71
2	10	300	1,07	76,00	0,93(1,88)	1,05(2,50)	0,62	75,56
2	10	400	0,28	76,00	0,28(0,56)	0,28(0,56)	0,66	77,64
2	10	500	0,00	76,00	0,00(0,00)	0,00(0,00)	0,60	75,28
2	50	50	6,26	130,00	1,35(1,81)	3,45(3,65)	17,16	94,05
2	50	100	6,17	130,00	1,82(2,7)	3,71(4,28)	17,04	94,39
2	50	200	7,09	130,00	3,18(4,54)	5,91(6,63)	17,32	94,10
2	50	300	9,07	130,00	5,71(7,8)	7,56(8,45)	16,45	94,31
2	50	400	9,07	130,00	8,60(10,62)	8,85(10,40)	13,63	92,92
2	50	500	7,01	130,00	6,06(8,64)	5,57(7,00)	13,80	93,60
2	100	50	5,66	153,00	1,57(2,25)	3,26(3,50)	71,17	95,88
2	100	100	6,17	153,00	1,72(2,13)	3,54(3,90)	72,37	96,42
2	100	200	5,93	153,00	2,14(2,91)	3,72(3,82)	75,86	96,88
2	100	300	5,88	153,00	2,74(3,61)	4,53(5,01)	78,96	96,69
2	100	400	6,41	153,00	3,11(3,73)	4,79(5,01)	72,22	96,54
2	100	500	6,46	153,00	4,69(6,29)	5,42(5,71)	70,74	96,41
Média			5,60	116,41	2,96	4,29	26,00	88,23

Qual.1 = ((alg. memético – lim. inferior)/lim.inferior)*100
Qual.2 = ((BIH – lim. inferior)/lim. inferior)*100

Tabela 5: Resultados computacionais do algoritmo memético comparado a limitantes inferiores e a heurística BIH aplicado às instâncias experimentais com 5 máquinas.

#Máq.	#Tar.	R_{max}	IniPop (%)	#Ger.	Qual.1 (%)	Qual.2 (%)	Tempo (seg.)	CPU BL (%)
5	10	50	10,66	76,00	4,90(8,33)	5,53(8,33)	0,83	77,25
5	10	100	5,91	76,00	4,92(7,02)	5,60(8,55)	0,68	72,62
5	10	200	2,30	76,00	1,92(4,39)	2,20(4,39)	0,63	75,23
5	10	300	0,20	76,00	0,07(0,33)	0,07(0,33)	0,67	71,80
5	10	400	1,02	76,00	1,02(2,04)	1,02(2,04)	0,85	71,65
5	10	500	0,00	76,00	0,00(0,00)	0,00(0,00)	0,82	72,68
5	50	50	5,32	130,00	1,00(1,57)	2,94(3,32)	17,50	93,83
5	50	100	7,08	130,00	1,87(2,62)	3,26(4,23)	17,06	94,07
5	50	200	7,00	130,00	2,80(3,96)	4,89(6,13)	17,07	94,13
5	50	300	9,26	130,00	4,87(8,37)	7,03(9,16)	16,08	93,84
5	50	400	12,51	130,00	9,96(12,7)	10,92(13,10)	15,07	93,80
5	50	500	9,40	130,00	8,80(10,06)	8,23(9,09)	14,92	93,40
5	100	50	6,64	153,00	2,04(2,71)	3,77(4,27)	79,27	96,58
5	100	100	6,65	153,00	1,78(2,21)	3,89(4,24)	74,74	96,36
5	100	200	7,80	153,00	2,29(2,61)	4,02(4,35)	81,95	96,89
5	100	300	11,42	153,00	6,79(15,39)	8,94(17,27)	81,76	96,82
5	100	400	6,15	153,00	2,93(3,66)	4,12(4,98)	70,94	96,56
5	100	500	8,02	153,00	4,35(4,77)	5,03(5,28)	71,21	96,50
Média			6,44	116,41	3,25	4,38	27,02	87,28

Tabela 6: Resultados computacionais do algoritmo memético comparado a limitantes inferiores e a heurística BIH aplicado às instâncias experimentais com 10 máquinas.

#Máq.	#Tar.	R_{max}	IniPop (%)	#Ger.	Qual.1 (%)	Qual.2 (%)	Tempo (seg.)	CPU BL (%)
10	10	50	5,69	76,00	3,37(4,76)	4,33(7,14)	0,70	77,28
10	10	100	7,47	76,00	4,98(7,46)	6,01(8,96)	0,68	77,95
10	10	200	0,96	76,00	0,96(3,88)	1,50(3,88)	0,63	72,94
10	10	300	0,31	76,00	0,31(1,53)	0,31(1,53)	0,60	73,67
10	10	400	1,58	76,00	1,21(1,80)	2,57(3,59)	0,62	74,73
10	10	500	0,00	76,00	0,00(0,00)	0,00(0,00)	0,59	68,14
10	50	50	6,47	130,00	1,40(2,14)	3,06(3,57)	16,99	94,08
10	50	100	7,02	130,00	1,99(2,85)	3,34(4,01)	17,62	94,30
10	50	200	8,46	130,00	2,89(4,14)	4,69(5,95)	16,68	94,13
10	50	300	9,32	130,00	4,00(6,60)	5,59(7,30)	17,21	94,14
10	50	400	14,54	130,00	10,04(15,51)	10,38(14,26)	16,05	94,21
10	50	500	14,13	130,00	13,21(16,01)	12,93(14,29)	14,22	93,98
10	100	50	6,06	153,00	1,45(1,74)	2,89(3,01)	73,45	96,51
10	100	100	5,74	153,00	1,56(2,02)	3,14(3,49)	70,65	96,50
10	100	200	6,72	153,00	2,16(2,73)	3,21(3,64)	69,24	96,48
10	100	300	6,22	153,00	2,47(3,03)	3,98(4,96)	67,02	96,37
10	100	400	7,46	153,00	2,98(3,84)	4,59(4,67)	69,96	96,43
10	100	500	7,84	153,00	3,56(4,42)	4,80(5,14)	71,47	96,52
Média			6,22	116,41	2,96	4,04	25,20	87,64

Conforme R_{max} aumenta, as instâncias tendem a ficar mais fáceis pois elas ficam próximas de uma simples ordenação por datas de liberação. Isso pode ser observado para instâncias com 10 tarefas e 2, 5 e 10 máquinas, onde o algoritmo memético e o BIH encontram o ótimo com R_{max} = (400, 500). Percebe-se também que o limitante inferior é "forte", o que entende-se que ele está perto do ótimo, quando existe poucas tarefas e que ele tende a ficar cada vez mais distante quando as tarefas aumentam e quando o R_{max} aumenta para instâncias com 50 e 100 tarefas.

Para 50 e 100 tarefas, a heurística construtiva se aproxima da qualidade do algoritmo memético, quando o R_{max} varia entre 50 e 500, e chega a ser melhor que o algoritmo memético, quando R_{max} = 500 e 50 tarefas.

A heurística construtiva não chega a ser melhor que o algoritmo memético para 100 tarefas, mesmo quando R_{max} = (500), pois as datas de liberação são melhor distribuídas com um número de tarefas maior, favorecendo o algoritmo memético.

A piora do algoritmo memético para R_{max} = (500) pode ser explicada pela relaxação da restrição de datas de liberação e alteração da matriz de custo do ATSP-RT que passa a usar o

valor das datas de liberação para calcular a distância de um vértice a outro.

No geral, o algoritmo memético chega a ser melhor que a heurística construtiva em 44 instâncias, em outras 7 ambos obtêm o mesmo resultado e em 3 instâncias a heurística construtiva é melhor que o algoritmo memético.

Observa-se, também, que para as instâncias com 10 tarefas o algoritmo memético encontra o ótimo global logo na inicialização da população usando uma heurística simplista do vizinho mais próximo. Quando a média do tempo gasto pela busca local é menor que 90% é porque algumas instâncias (já que para cada conjunto de parâmetros (máquinas, tarefas e R_{max}) foram gerados no máximo 5 instâncias) a busca local não precisou ser aplicada, pois a população inicial já tinha uma solução com custo igual ao limitante inferior.

4.3. GERAÇÃO DE INSTÂNCIAS COM RESULTADOS ÓTIMOS CONHECIDOS

Para testar o desempenho do algoritmo memético em relação à outras heurísticas, bem como realizar análises de operadores e de *fitness landscapes,* é interessante o uso de instâncias cujo resultado ótimo seja conhecido. Para realizar tais testes, instâncias com resultados ótimos podem ser criadas a partir de instâncias do caixeiro viajante e do *Ordered Flowshop Sequencing Problem* (OFSP).

O problema OFSP possui duas características que o distinguem de outros problema de sequenciamento. A primeira característica envolve todas as tarefas de uma instância. Se uma tarefa possuir um tempo de processamento menor que uma outra tarefa qualquer que seja a máquina considerada, então o tempo de processamento da primeira tarefa tem que ser menor ou igual ao tempo de processamento da outra em todas as outras máquinas restantes. A segunda característica tem a ver com as máquinas da instância. A máquina com o menor tempo de processamento para uma tarefa tem que ter o menor tempo de processamento para todas as outras tarefas também.

O problema OFSP ocorre quando o tempo de processamento das tarefas está associado com o número de partes ou unidades que uma tarefa possui (i.e. tarefa com mais partes levaria mais tempo que as outras independentemente da máquina) ou com a complexidade da tarefa (i.e.

tarefas mais complexas podem requerer operações que leve mais tempo que tarefas menos complexas).

Segundo Smith et al. [1975] o problema OFSP pode ser resolvido por uma simples ordenação das tarefas. Se a última máquina possuir as tarefas com os maiores tempos de processamento, então as tarefas são ordenadas de forma não decrescente dos seus tempos de processamento. Se a primeira máquina possuir as tarefas com os maiores tempos de processamentos, então a ordenação das tarefas é feita de forma não crescente dos seus tempos de processamento.

Em 1979, Panwalkar e Woollam estenderam a prova desse caso especial que ocorre para *flowshop* para o *no-wait flowshop* (NWFSP).

Usando instâncias do *Ordered No-Wait Flowshop Sequencing Problem* (ONWFSP) com as matrizes do caixeiro viajante assimétrico (ATSP) para representar os tempos de preparação entre as máquinas é possível criar instâncias com resultados cujo ótimo global é conhecido e é o mesmo do ATSP.

Como há instâncias do ATSP que são amplamente usadas na literatura e por elas representarem dados reais, elas fornecem características relevantes para testar a performance e a qualidade dos algoritmos propostos para o NWFSP. Ao incluir essas instâncias como tempos de preparação para o problema do NWFSP, as características do problema ATSP são preservadas para o problema NWFSP.

Para gerar as instâncias com resultados ótimos conhecidos para NWFSP, o seguinte procedimento deve ser feito, a partir de uma instância do ATSP com rota ótima conhecida:

1) A matriz de tempos de preparação da 1ª máquina do NWFSP é igual à matriz de distância do ATSP. A matriz de tempos de preparação das outras máquinas é igual à da 1ª máquina vezes [1, *ps*] que multiplica todo s_{ij} que não pertence à rota ótima do ATSP, preservando assim a otimilidade da rota do ATSP.
2) Os tempos de processamento *p* das tarefas têm que ser em ordem não decrescente, de acordo com sua posição na rota ótima do ATSP. Suponha que *t* seja a rota ótima e *t(i)* a *i-ésima* tarefa dentro da rota, então $p_{t(1)} < p_{t(2)} < ... < p_{t(n)}$ para n tarefas.
3) Na primeira máquina os tempos são escolhidos aleatoriamente e distribuídos para as

tarefas de acordo com a ordem delas dentro da rota ótima. Os tempos podem variar entre 0 e *pp*médiaATSP* (Tabela 8) tal que *pp* é um número que multiplica o valor médio da matriz de distância do ATSP. Nas máquinas seguintes os tempos de processamento são de [0,20%] maior que na 1ª máquina. De forma que a última máquina tenha os maiores tempos de processamento.

4) O tempo de liberação varia de 0 ao tempo de início da cidade *i* na posição *t(i)* da rota. Para a última cidade da rota, posição *t(n)*, a data de liberação é igual ao tempo de início da cidade *n* na rota, para garantir que o caminho ótimo do NWFSP até *n* seja o mesmo que o caminho na rota ótima do ATSP de 0 até *n*.

A Tabela 7 mostra os valores medianos, médios e máximos calculados para cada instância do ATSP usada como matriz de tempos de preparação para o problema NWFSP. É interessante observar a diferença entre o máximo e o mediano, por haver uma diferença muito alta entre esses volores, diferente de qualquer distribuição uniforme gerada aleatoriamente, é importante o uso do valor médio como valor de referência para gerar os tempos de processamento. As instâncias do ATSP estão disponíveis na TSPLIB e representam casos de problemas reais estudados na literatura (Reinelt [1991]).

Tabela 7: Características das instâncias do ATSP.

Instância	#Tarefas	Max.	Média	Mediano
br17	17	74	14	8
ftv33	34	332	128	126
ftv35	36	332	135	133
ftv38	39	332	132	131
p43	43	5160	593	25
ftv44	45	332	136	135
ftv47	48	348	142	140
ry48p	48	2782	1138	1029
ft53	53	1834	492	372
ftv55	56	324	131	130
ftv64	65	348	135	133
ft70	70	2588	1030	921
ftv70	71	348	137	135
kro124p	100	4545	1910	1788

Instâncias com 2, 5 e 10 máquinas foram geradas. Para cada combinação de tarefas e

máquinas foram geradas 4 instâncias com parâmetros (*pp, ps*) diferentes. Os parâmetros para cada tipo de instância gerada para o algoritmo memético está descrito na Tabela 8.

Tabela 8: Parâmetros utilizados para geração de instâncias para o NWFSP.

Tipo	pp	ps
1	0,25	1,10
2	0,25	1,50
3	2	1,10
4	2	1,50

A instância tipo 1 com 2 máquinas é considerada mais difícil, pois o *ps* altera pouco a matriz de distância do ATSP e os tempos de processamento tem uma pequena influência na matriz de distância quando o NWFSP é reduzido para o problema do ATSP-RT. As instâncias do tipo 2 tem uma influência maior do parâmetro *ps,* alterando as características da matriz de distância do ATSP. As instâncias do tipo 3 e 4 perdem mais ainda as características da matriz de distância do ATSP já que os tempos de processamento têm grande peso durante o processo de redução do NWFSP para o ATSP e são afetados pelo parâmetro *pp*.

Nas Tabelas 9 à 20, **Pop.Inicial** representa o desvio médio da população inicial em relação ao ótimo, **Qual.1** representa o desvio médio do algoritmo memético em relação ao ótimo, **Qual.2** representa o desvio médio da heurística construtiva em relação ao ótimo, **#Ger.** representa o número médio de gerações para encontrar o ótimo, **Otm/#exe** representa o número de vezes que o algoritmo memético encontrou o ótimo, **Tempo** é o tempo computacional médio para achar o ótimo e **CPU BL** é a porcentagem média de tempo gasto com a busca local.

Tabela 9: Resultados computacionais para instâncias com resultados ótimos conhecidos para 2 máquinas e parâmetros do tipo 1.

Instância	IniPop (%)	#Ger.	Qual.1 (%)	Qual.2 (%)	Otm/ #exe	Tempo (seg.)	CPU BL (%)
br17	40,84	1,2	0(0)	23,94	10/10	0,19	96,89
ftv33	21,61	3,7	0(0)	15,46	10/10	1,14	97,71
ftv35	16,36	13,7	0(0)	13,49	10/10	2,71	93,39
ftv38	32,29	9,3	0(0)	16,46	10/10	2,78	96,29
p43	8,66	8,8	0(0)	12,02	10/10	2,98	97,04
ftv44	27,51	15,9	0(0)	24,87	10/10	3,38	94,77
ftv47	23,43	4,3	0(0)	6,07	10/10	2,35	99,11
ry48p	15,03	8,5	0(0)	28,33	10/10	3,14	96,62
ft53	47,6	4,1	0(0)	27,47	10/10	2,79	99,39
ftv55	21,66	6,2	0(0)	28,9	10/10	3,53	97,37
ftv64	26,66	11	0(0)	27,94	10/10	5,64	97,18
ft70	19,72	87,1	0,13(1,08)	19,05	8/10	16,01	94,9
ftv70	25,62	5,2	0(0)	25,62	10/10	5,71	98,48
kro124p	20,3	20,7	0(0)	26,73	10/10	16,65	97,87
Média	24,8	14,26	0,0095	21,17	9,86/10	4,93	96,93

Tabela 10: Resultados computacionais para instâncias com resultados ótimos conhecidos para 2 máquinas e parâmetros do tipo 2.

Instância	IniPop (%)	#Ger.	Qual.1 (%)	Qual.2 (%)	Otm/ #exe	Tempo (seg.)	CPU BL (%)
br17	47,89	0,4	0(0)	23,94	10/10	0,15	92,21
ftv33	38,9	0,2	0(0)	16,33	10/10	0,67	98,21
ftv35	19,7	3,4	0(0)	17,17	10/10	1,02	98,92
ftv38	21,63	2,5	0(0)	13,88	10/10	1,12	97,06
p43	13,32	14,4	0(0)	13,18	10/10	3,25	96,92
ftv44	33,77	3,9	0(0)	8	10/10	1,58	97,72
ftv47	37,29	3,1	0(0)	14,02	10/10	1,71	95,96
ry48p	20,39	5,4	0(0)	3,09	10/10	2,11	97,68
ft53	57,85	6,6	0(0)	42,53	10/10	2,6	97,31
ftv55	16,74	9,2	0(0)	29,72	10/10	3,21	97,91
ftv64	20,82	11,7	0(0)	25,14	10/10	4,89	97,63
ft70	17,98	23,5	0(0)	27,12	10/10	7,51	96,87
ftv70	31,37	13,8	0(0)	18,67	10/10	6,65	97,32
kro124p	13,37	14,4	0(0)	20,51	10/10	13,59	98,36
Média	27,93	8,04	0	19,52	10/10	3,58	97,15

Tabela 11: Resultados computacionais para instâncias com resultados ótimos conhecidos para 2 máquinas e parâmetros do tipo 3.

Instância	IniPop (%)	#Ger.	Qual.1 (%)	Qual.2 (%)	Otm/ #exe	Tempo (seg.)	CPU BL (%)
br17	21,51	0	0(0)	8,96	10/10	0,13	100
ftv33	16,35	0	0(0)	8,56	10/10	0,67	99,25
ftv35	6,71	0	0(0)	0,54	10/10	0,76	99,21
ftv38	13,68	0	0(0)	0,38	10/10	0,92	98,36
p43	6,84	0,7	0(0)	4,36	10/10	1,27	97,49
ftv44	10,65	0,1	0(0)	0	10/10	1,23	100
ftv47	13,22	0	0(0)	1,69	10/10	1,44	99,65
ry48p	12,08	0	0(0)	8,42	10/10	1,54	99,35
ft53	22,64	0	0(0)	13,77	10/10	1,8	99,72
ftv55	16,95	0	0(0)	9,78	10/10	2	99,15
ftv64	13,73	1	0(0)	5,15	10/10	3,08	99,35
ft70	13,28	7,8	0(0)	2,55	10/10	4,5	98,49
ftv70	15,01	0,1	0(0)	12,64	10/10	3,4	99,47
kro124p	7,49	3,1	0(0)	10,27	10/10	9,71	99,33
Média	13,58	0,91	0	6,22	10/10	2,32	99,2

Tabela 12: Resultados computacionais para instâncias com resultados ótimos conhecidos para 2 máquinas e parâmetros do tipo 4.

Instância	IniPop (%)	#Ger.	Qual.1 (%)	Qual.2 (%)	Otm/ #exe	Tempo (seg.)	CPU BL (%)
br17	22,58	0	0(0)	9,32	10/10	0,15	96,64
ftv33	25,81	0	0(0)	4,52	10/10	0,66	100
ftv35	8,53	0	0(0)	0,63	10/10	0,76	99,34
ftv38	15,22	0	0(0)	0,67	10/10	0,9	100
p43	10,34	12,9	0(0)	5,2	10/10	2,24	96,79
ftv44	12,57	0	0(0)	3,79	10/10	1,21	99,59
ftv47	16,73	0	0(0)	0	10/10	1,38	99,2
ry48p	9,81	0,2	0(0)	0	10/10	1,55	99,23
ft53	23,82	0	0(0)	15,22	10/10	1,72	98,78
ftv55	17,95	0,1	0(0)	11,34	10/10	2,02	99,7
ftv64	14,42	0,2	0(0)	6,34	10/10	2,76	99,17
ft70	13,73	3,9	0(0)	0,37	10/10	3,8	99,16
ftv70	16,36	1,8	0(0)	0,48	10/10	3,82	99,37
kro124p	8,63	3,5	0(0)	7,94	10/10	9,32	99,11
Média	15,46	1,61	0	4,70	10/10	2,31	99,01

Tabela 13: Resultados computacionais para instâncias com resultados ótimos conhecidos para 5 máquinas e parâmetros do tipo 1.

Instância	IniPop (%)	#Ger.	Qual.1 (%)	Qual.2 (%)	Otm/ #exe	Tempo (seg.)	CPU BL (%)
br17	42,5	0	0(0)	22,5	10/10	0,14	96,35
ftv33	36,9	0	0(0)	17,49	10/10	0,65	95,72
ftv35	22,39	0,3	0(0)	1,05	10/10	0,75	97,87
ftv38	33,24	0	0(0)	16,98	10/10	0,84	98,8
p43	15,2	2,6	0(0)	11,29	10/10	1,32	96,83
ftv44	35,84	0	0(0)	20,75	10/10	1,18	99,49
ftv47	39,5	0,1	0(0)	4,65	10/10	1,35	98,74
ry48p	42,49	2,6	0(0)	7,53	10/10	1,83	99,4
ft53	48,63	0	0(0)	24,73	10/10	1,7	99,71
ftv55	39,02	3,3	0(0)	12,83	10/10	2,42	99,05
ftv64	33,5	2,7	0(0)	25,04	10/10	3,23	99,07
ft70	26,9	10,4	0(0)	16,54	10/10	5,18	97,88
ftv70	37,82	2,1	0(0)	19,85	10/10	3,77	98,97
kro124p	29,19	15	0(0)	12,1	10/10	13,19	98,02
Média	34,51	2,79	0	15,24	10/10	2,68	98,28

Tabela 14: Resultados computacionais para instâncias com resultados ótimos conhecidos para 5 máquinas e parâmetros do tipo 2.

Instância	IniPop (%)	#Ger.	Qual.1 (%)	Qual.2 (%)	Otm/ #exe	Tempo (seg.)	CPU BL (%)
br17	46,25	0	0(0)	25	10/10	0,15	95,95
ftv33	16,62	0	0(0)	2,17	10/10	0,63	96,51
ftv35	44,02	0	0(0)	2,56	10/10	0,72	100
ftv38	21,6	0	0(0)	2,25	10/10	0,83	98,55
p43	30,8	2,6	0(0)	12,94	10/10	1,42	97,96
ftv44	29,73	0	0(0)	4,55	10/10	1,16	98,62
ftv47	41,88	0	0(0)	1,81	10/10	1,29	99,15
ry48p	25,38	0	0(0)	0	10/10	1,46	99,31
ft53	57,15	0,3	0(0)	42,1	10/10	1,68	99,64
ftv55	45,27	0,2	0(0)	21,48	10/10	1,91	99,06
ftv64	32,75	5,5	0(0)	15,45	10/10	3,62	99,23
ft70	25,29	3	0(0)	0	10/10	3,55	99,27
ftv70	38,12	2,7	0(0)	2,18	10/10	3,73	98,98
kro124p	32,51	5,3	0(0)	19,13	10/10	10,1	99,07
Média	34,81	1,4	0	10,83	10/10	2,30	98,66

Tabela 15: Resultados computacionais para instâncias com resultados ótimos conhecidos para 5 máquinas e parâmetros do tipo 3.

Instância	IniPop (%)	#Ger.	Qual.1 (%)	Qual.2 (%)	Otm/ #exe	Tempo (seg.)	CPU BL (%)
br17	20,16	0	0(0)	16,94	10/10	0,14	100
ftv33	21,55	0	0(0)	9,11	10/10	0,7	99,29
ftv35	18,52	0	0(0)	0,65	10/10	0,79	99,24
ftv38	23,81	0	0(0)	0,66	10/10	0,96	99,37
p43	19,15	0	0(0)	6,19	10/10	1,21	99,51
ftv44	26,17	0	0(0)	0,05	10/10	1,33	98,34
ftv47	21,53	0	0(0)	1	10/10	1,53	99,35
ry48p	30,48	0	0(0)	0,69	10/10	1,59	99,62
ft53	32,89	0	0(0)	2,25	10/10	1,92	98,96
ftv55	27,98	0	0(0)	4,43	10/10	2,14	99,25
ftv64	28,21	0	0(0)	0,03	10/10	2,99	99,03
ft70	24,41	0	0(0)	3,42	10/10	3,5	99,83
ftv70	28,17	0	0(0)	0,11	10/10	3,65	99,86
kro124p	25,8	0	0(0)	3,5	10/10	8,95	99,7
Média	24,92	0	0	3,5	10/10	2,24	99,38

Tabela 16: Resultados computacionais para instâncias com resultados ótimos conhecidos para 5 máquinas e parâmetros do tipo 4.

Instância	IniPop (%)	#Ger.	Qual.1 (%)	Qual.2 (%)	Otm/ #exe	Tempo (seg.)	CPU BL (%)
br17	20,97	0	0(0)	8,6	10/10	0,15	100
ftv33	25,03	0	0(0)	1,36	10/10	0,69	99,13
ftv35	19,28	0	0(0)	0,67	10/10	0,78	100
ftv38	24,15	0	0(0)	0,74	10/10	0,93	98,82
p43	20,34	2,3	0(0)	15,53	10/10	1,31	99,23
ftv44	27,32	0	0(0)	0	10/10	1,29	98,83
ftv47	21,99	0	0(0)	0	10/10	1,46	99,18
ry48p	33,07	0	0(0)	0	10/10	1,56	99,23
ft53	36,05	0	0(0)	1,05	10/10	1,84	99,13
ftv55	31,44	0	0(0)	6,74	10/10	2,12	99,72
ftv64	26,45	0	0(0)	0,12	10/10	2,93	99,28
ft70	26,98	0	0(0)	0	10/10	3,4	99,35
ftv70	33,98	0	0(0)	0,18	10/10	3,55	99,86
kro124p	25,9	0	0(0)	1,91	10/10	8,67	99,75
Média	26,64	0,16	0	2,64	10/10	2,19	99,39

Tabela 17: Resultados computacionais para instâncias com resultados ótimos conhecidos para 10 máquinas e parâmetros do tipo 1.

Instância	IniPop (%)	#Ger.	Qual.1 (%)	Qual.2 (%)	Otm/ #exe	Tempo (seg.)	CPU BL (%)
br17	32,63	0,1	0(0)	29,47	10/10	0,13	92,06
ftv33	52,77	0	0(0)	1,21	10/10	0,66	98,5
ftv35	30,73	0	0(0)	0,96	10/10	0,77	99,35
ftv38	32,98	0	0(0)	1,11	10/10	0,89	98,09
p43	24,47	0	0(0)	13,12	10/10	1,12	99,55
ftv44	30,47	0	0(0)	0,97	10/10	1,22	98,69
ftv47	57,31	0	0(0)	17,48	10/10	1,4	98,86
ry48p	35,64	0	0(0)	24,05	10/10	1,54	100
ft53	60,72	0	0(0)	21,4	10/10	1,77	99,66
ftv55	27,74	0	0(0)	14,82	10/10	2,01	100
ftv64	62,81	0,8	0(0)	15,36	10/10	2,94	99,15
ft70	30,36	0,9	0(0)	3,49	10/10	3,44	99,65
ftv70	28,34	0,5	0(0)	19,12	10/10	3,45	99,04
kro124p	44,1	8,7	0(0)	22,76	10/10	10,93	98,65
Média	39,36	0,79	0	13,24	10/10	2,31	98,66

Tabela 18: Resultados computacionais para instâncias com resultados ótimos conhecidos para 10 máquinas e parâmetros do tipo 2.

Instância	IniPop (%)	#Ger.	Qual.1 (%)	Qual.2 (%)	Otm/ #exe	Tempo (seg.)	CPU BL (%)
br17	46,32	0	0(0)	35,79	10/10	0,14	95,77
ftv33	51,82	0	0(0)	22,48	10/10	0,66	99,24
ftv35	18,47	0	0(0)	2,01	10/10	0,74	99,32
ftv38	41	0	0(0)	2,06	10/10	0,87	100
p43	34,64	0	0(0)	13,36	10/10	1,12	98,56
ftv44	25,37	0	0(0)	2,56	10/10	1,18	100
ftv47	57,88	0	0(0)	16,56	10/10	1,31	100
ry48p	57,75	0	0(0)	0	10/10	1,45	99,59
ft53	69,35	0	0(0)	22,73	10/10	1,71	99,71
ftv55	31,95	0	0(0)	13,12	10/10	1,91	99,11
ftv64	61,89	0	0(0)	26,46	10/10	2,64	99,36
ft70	26,25	0	0(0)	0	10/10	3,1	99,61
ftv70	41,2	0	0(0)	13,4	10/10	3,38	99,7
kro124p	36,85	0,8	0(0)	7,83	10/10	8,21	99,59
Média	42,91	0,06	0	12,74	10/10	2,03	99,25

Tabela 19: Resultados computacionais para instâncias com resultados ótimos conhecidos para 10 máquinas e parâmetros do tipo 3.

Instância	IniPop (%)	#Ger.	Qual.1 (%)	Qual.2 (%)	Otm/ #exe	Tempo (seg.)	CPU BL (%)
br17	20,09	0	0(0)	12,36	10/10	0,14	95,62
ftv33	31,23	0	0(0)	1,04	10/10	0,71	98,46
ftv35	28,93	0	0(0)	7,65	10/10	0,81	98,65
ftv38	29,73	0	0(0)	0,89	10/10	0,96	100
p43	30,51	0	0(0)	3,92	10/10	1,22	99,1
ftv44	28,01	0	0(0)	0,21	10/10	1,32	99,17
ftv47	42,32	0	0(0)	0,35	10/10	1,53	98,63
ry48p	31,3	0	0(0)	10,44	10/10	1,59	98,68
ft53	37,85	0	0(0)	3,64	10/10	1,95	99,13
ftv55	28,41	0	0(0)	0	10/10	2,21	99,73
ftv64	32,09	0	0(0)	0,02	10/10	3,05	99,84
ft70	32,24	0	0(0)	1,9	10/10	3,75	99,71
ftv70	30,07	0	0(0)	0,06	10/10	3,83	99,58
kro124p	28,71	0	0(0)	2,84	10/10	8,92	99,83
Média	30,82	0	0	3,24	10/10	2,28	99,01

Tabela 20: Resultados computacionais para instâncias com resultados ótimos conhecidos para 10 máquinas e parâmetros do tipo 4.

Instância	IniPop (%)	#Ger.	Qual.1 (%)	Qual.2 (%)	Otm/ #exe	Tempo (seg.)	CPU BL (%)
br17	20,87	0	0(0)	12,21	10/10	0,15	96,1
ftv33	32,32	0	0(0)	1,12	10/10	0,71	100
ftv35	30,29	0	0(0)	0,93	10/10	0,8	99,25
ftv38	28,86	0	0(0)	0,93	10/10	0,95	100
p43	30,52	0,4	0(0)	5,03	10/10	1,2	98,59
ftv44	30,97	0	0(0)	0,25	10/10	1,31	100
ftv47	45,47	0	0(0)	0	10/10	1,5	99,67
ry48p	36,89	0	0(0)	8,41	10/10	1,54	100
ft53	39,94	0	0(0)	0,36	10/10	1,92	98,96
ftv55	29,74	0	0(0)	0	10/10	2,16	98,71
ftv64	36	0	0(0)	0,08	10/10	3,1	99,42
ft70	34,39	0	0(0)	0	10/10	3,64	100
ftv70	31,82	0	0(0)	0,09	10/10	3,82	99,71
kro124p	29,96	0	0(0)	2,62	10/10	9,12	99,75
Média	32,72	0,03	0	2,29	10/10	2,28	99,3

Para todas as instâncias o algoritmo memético ou foi melhor que a heurística construtiva ou obteve resultados iguais.

A população inicial para as instâncias geradas a partir de instâncias do ATSP têm desvio percentual aproximado de 30% do ótimo, ao contrário das instâncias experimentais que têm desvio percentual aproximado de 6%, na média, do limitante inferior.

O algoritmo memético encontra o ótimo para essas instâncias cerca de dez vezes para cada dez execuções do algoritmo. Somente para uma única instância, **ft70** para 2 máquinas com parâmetros do tipo 1, o ótimo não foi encontrado duas vezes nem pelo algoritmo memético e nem pela heurística construtiva. O algoritmo memético se mostra superior à heurística construtiva para instâncias do tipo 1 e 2. Para as instâncias do tipo 3 e 4, ambos os métodos são competitivos com uma ligeira vantagem para o algoritmo memético.

A maior parte do tempo do algoritmo memético, cerca de 95%, é gasto com a execução da busca local, enquanto o restante é gasto para inicialização da população e a execução dos operadores de recombinação e mutação. A complexidade do *crossover* OX é O(n) e a mutação 3-Opt é menor do que O(n).

CONCLUSÃO

O problema NWFSP, que pode surgir com as características de produção *just-in-time* e inventário zero, é um caso especial do problema de *flowshop* encontrado nos ambientes de produção, principalmente na indústria química, farmacêutica, siderúrgica e de serviços. A resolução desse problema está associada a métodos aplicados ao problema do caixeiro viajante, já que o NWFSP pode ser reduzido ao problema do caixeiro viajante.

Sendo o problema do caixeiro viajante muito estudado na literatura, vários métodos foram desenvolvidos, contribuindo assim para que problemas que se reduzem a ele possam usufruir desse benefício e ser aplicados no ambiente de produção. Geralmente esses problemas estudados na literatura possuem poucas ou quase nenhuma restrições, ao contrário do problema apresentado nessa dissertação.

O problema de *flowshop* apresentado nessa dissertação possui tempos de preparação entre tarefas, datas de liberação e a restrição de armazenamento zero entre máquinas ou na máquina, que pode ocorrer devido a tecnologia do processo.

A redução desse problema para o problema do caixeiro viajante existe e pode ser feita em tempo polinomial. O problema reduzido é o problema do caixeiro viajante assimétrico com restrições de tempo para visitar as cidades (ATSP-RT).

Um algoritmo memético foi apresentado para resolver esse problema. Foram testados 4 operadores de *crossover*, 3 operadores de mutação e 5 buscas locais.

Os operadores de *crossover* que melhor se adaptam ao problema são aqueles que procuram preservar a posição absoluta das tarefas na seqüência ou das cidades na rota ótima do ATSP.

Os operadores de mutação são usados para manter a diversidade da população ou mudar drasticamente o espaço de vizinhança de uma solução, fazendo com que ela se situe em uma base de atração de um ótimo local diferente do qual ela estava. O operador que melhor se adapta, ou que mostra melhores resultados, ao NWFSP é o 3-Opt.

Para escolher a melhor busca local entre todas estudadas, foi feito uma análise de

qualidade entre elas sem a necessidade de executar todo o processo evolutivo do algoritmo memético. Soluções aleatórias foram geradas, o ótimo local gerado pelas buscas locais para cada solução aleatória foi então comparado e a busca local com ótimos locais predominantes foi escolhida.

Depois, uma análise de *fitness landscape* foi realizada para as instâncias mais difíceis para explicar o comportamento do algoritmo memético.

Instâncias aleatórias, com tempos de preparação, processamento e datas de liberação geradas aleatoriamente e distribuídos uniformemente em um intervalo, mostraram ser mais fáceis de ser resolvidas do que instâncias derivadas do problema do caixeiro viajante.

O algoritmo memético encontrou resultados melhores que a heurística construtiva testada na maioria das instâncias.

Trabalhos futuros podem ser conduzidos com operadores de crossover, busca local, ou mutação diferentes. O problema NWFSP pode ser estendido com restrições diferentes e/ou objetivos diferentes, aumentando assim sua complexidade e tornando-se interessante para o uso de heurísticas que usufruem de processamento paralelo assim como o algoritmo memético usufrui.

REFERÊNCIAS BIBLIOGRÁFICAS

Bianco, L., Dell' Olmo, P., and Giordani, S. (1998). "Flow shop no-wait scheduling with sequence dependent setup times and release dates", *INFOR*, **37(1)**.

Boese, K. D., Kahng, A. D., and Muddu, S. (1994). "A New Adaptive Multi-Start Techinique for Combinatorial Global Optimizations", *Operations Research Letters*, **16(2)**, pp. 175-181.

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

Bui, T. N. and Moon, B. R. (1996). "Genetic algorithm and graph partitioning", *IEEE Transactions on Computers* **45(7),** pp. 841-855

Buriol, L. S., França, P., and Moscato, P. (1999). "Recursive Arc Insertion: A New Local Search Embedded in a Memetic Algorithm for the Asymmetric Traveling Salesman Problem", *Journal of Heuristics*, to appear.

Buriol, L. S. (2000). "Algoritmo Memético para o Problema do Caixeiro Viajante Assimétrico como parte de um Framework para Algoritmos Evolutivos", Dissertação de mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação, *Universidade Estadual de Campinas* (UNICAMP).

Cheng, R.W. and Gen, M. (1997). "Parallel machine scheduling problems using memetic algorithms", *Computers & Industrial Engineering* **33(34),** pp.761-764

Dawkin, R. (1976). "The selfish gene", *Oxford University Press*, Oxford.

Dudek, R. A., Panwalkar, S. S. and Smith M.L. (1992). "The lessons of flowshop scheduling research", *Operations Research*, **40**, 7-13.

Fink A. and Voß S. (1998). "Applications of Modern Heuristic Search Methods to Continuous Flow-Shop Sequencing Problems", Working Paper, TU Braunschweig.

Freisleben, B. and Merz, P. (1996). "A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems", In *Proceedings of the 1996 IEE International Conference on Evolutionary Computation*, Nagoya, Japan, 616-621.

Gilmore, P. C. and Gomory R. E. (1964). "Sequencing a one state-variable machine: a solvable case of the traveling salesman problem", *Operations Research,* **12**, 655-679.

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

Hadamard, J. (1949). "The psychology of invention in the mathematical field", *Princeton University Press*, Princeton, NJ.

Hall, N.G. and Sriskandarajah, C. (1996). "A survey of machine scheduling problems with blocking and no-wait in process", *Operations Research,* **44**, 510-525.

Horowitz, E. & Sahni, S. (1978). "Fundamentals of Computer Algorithms", *Computer Science Press*, **11**, 551-552

Merz, P. and Freisleben, B. (1997). "A Genetic Local Search Approach to the Quadratic Assignment Problem", in T. Bäck (ed.), *Proceedings of the 7th international Conference on Genetic Algorithms* (ICGA' 97), Morgan Kaufmann, pp. 465-472

Merz, P. and Freisleben, B. (1998). "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.

Mendes, A., França P., and Moscato, P. (2001). "Memetic Algorithms to minimize tardiness on a single machine with sequence-dependent setup times", *Journal of Operational Research,* accepted.

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

Moscato, P. & Norman, M.G. (1992). "A Memetic Approach for the Traveling Salesman Problem Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems", in M. Valero, E. Onate, M. Jane, J. L. Larriba, B. Suarez (eds.). *Parallel Computing and Transputer Applications*, IOS Press, Amsterdam, pp. 177-186

Panwalkar, S. S. and Woollam, C. R. (1979). "Flowshop sequencing problem with no in-process waitin: a special case", *Journal of Operational Research Society*, **30**, pp. 661-664.

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

Reinelt, G. (1991), "TSPLIB - A Traveling Salesman Library", *ORSA Journal on Computing*, **3**, pp.376-384.

Sahni, S. & Cho, Y. (1979). "Complexity of scheduling shop with no wait in process", *Mathematics of Operations Research*, **4**, 448-457.

Smith, M. L., Panwalkar, S. S., and Dudek, R. A. (1975). "Flowshop Sequencing Problem with Ordered Processing Time Matrices", *Management Science*, **21(5),** pp. 544-549.

TSPLIB web page, http://www.crpc.rice.edu/softlib/tsplib/.

Weinberger, E. D. (1990). "Correlated and Uncorrelated Fitness Landscapes and How to Tell the Difference", *Biological Cybernetics*, **63**, pp. 325-336.

Wismer, D. A. (1972). "Solution of the flowshop scheduling problem with no intermediate queues", *Operations Research*, **20**, 689-697.

Wright S. (1932). "The Roles of Mutation, Inbreeding, Crossbreeding and Selection in Evolution", *Proceedings of the 6th International Congress on Genetics*, **1**, 356-366.