UFSM

Dissertação de Mestrado

ALGORITMOS EVOLUTIVOS PARA O PROBLEMA DE SEQÜENCIAMENTO DE TAREFAS EM MÁQUINAS PARALELAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQÜÊNCIA

Viviane Cátia Köhler

PPGEP

Santa Maria, RS, Brasil

2004

ALGORITMOS EVOLUTIVOS PARA O PROBLEMA DE SEQÜENCIAMENTO DE TAREFAS EM MÁQUINAS PARALELAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQÜÊNCIA

por

Viviane Cátia Köhler

Dissertação apresentada ao Curso de Mestrado do
Programa de Pós-Graduação em Engenharia de Produção,
Área de Concentração em Tecnologia de Informação,
da Universidade Federal de Santa Maria (UFSM, RS),
como requisito parcial para a obtenção do grau de
Mestre em Engenharia de Produção.

PPGEP

Santa Maria, RS, Brasil

2004

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Engenharia de Produção

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

ALGORITMOS EVOLUTIVOS PARA O PROBLEMA DE SEQÜENCIAMENTO DE TAREFAS EM MÁQUINAS PARALELAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQÜÊNCIA

elaborada por

Viviane Cátia Köhler

como requisito parcial para obtenção do grau de
Mestre em Engenharia de Produção

COMISSÃO EXAMINADORA:

_____
Prof. Dr. Felipe Martins Müller (UFSM)
(Presidente/Orientador)

_____
Prof. Dr. Luiz Satoru Ochi (UFF)

_____
Prof. Dr. Luis Felipe Dias Lopes (UFSM)

Santa Maria, 11 outubro de 2004

Desafio

Rir é correr o risco de parecer tolo.
Chorar é correr o risco de parecer sentimental.
Estender a mão é correr o risco de se envolver.
Expor seu sentimento é correr o risco
de mostrar seu verdadeiro eu.
Defender seus sonhos e idéias é correr
o risco de perder a amizade das pessoas.
Amar é correr o risco de não ser compreendido.
Confiar é correr o risco de se decepcionar.
Tentar é correr o risco de fracassar.
Porém, riscos devem ser corridos,
pois o maior perigo é não arriscar nada.
A pessoa que não corre nenhum risco
não faz nada, não tem nada,
não consegue nada, não é nada.
Ela até pode evitar sofrimentos e desilusões,
mas não se sente, não muda, não cresce e não vive.
Acorrentada por suas próprias atitudes,
ela torna-se escrava e priva-se da liberdade, por isso,
somente a pessoa que corre o risco
é totalmente ***Livre****.*

Autor Desconhecido

À Traudi (in memoriam), minha Mãe,

ETERNA incentivadora.

E ao Ari, meu Pai.

Dedico este trabalho.

AGRADECIMENTOS

À Universidade Federal de Santa Maria, pela oportunidade para a realização deste estudo, bem como pelo PPGEP, pelo acolhimento para a realização do curso e a CAPES pelo auxílio financeiro.

Ao Felipe Müller, pela orientação, paciência, amizade e compreensão.

Aos colegas do grupo de Otimização, pela amizade. Em especial, ao Olinto, Marcus e Haroldo, por suas valiosas contribuições no decorrer dos estudos.

Aos membros da banca, por terem aceitado participar da avaliação deste trabalho.

Aos meus pais, a quem devo o bem mais precioso: a Vida.

À minha irmã, Cláudia, e ao cunhado, Daniel, por todo apoio dado.

À minha sobrinha, Débora Luísa que me trouxe muitas alegrias com a sua chegada, dando um novo colorido para a vida.

Ao meu irmão, Jonas, que ajudou em vários momentos de "mudança". E a Mareni, que repartiu seu apartamento comigo.

À Patrícia Achterberg e a Jussara Rossato, pela amizade.

Ao Miguel Finger, pela atenção, amizade e carinho.

Aos meus demais amigos, *on-line* e *off-line*, que sempre ajudaram no que precisava.

E a todos que colaboraram de uma forma ou de outra para a realização deste trabalho.

A TODOS, muito Obrigada!

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

AM – Algoritmo Memético

DVRP – *Distance-constrained Vehicle*

EE – Estratégia Evolutiva

EEs – Estratégias Evolutivas

GENI – *Generalized Insertion*

GENIUS – *Generalized Insertion and Unstringing Insertion*

GRASP – *Greedy Randomized Adaptive Procedure*

PCV – Problema do Caixeiro Viajante

PRV – Problema de Roteamento de Veículos

OX – Order Crossover

US – Unstringing Insertion

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Engenharia de Produção
Universidade Federal de Santa Maria, RS, Brasil.

ALGORITMOS EVOLUTIVOS PARA O PROBLEMA DE SEQÜENCIAMENTO DE TAREFAS EM MÁQUINAS PARALELAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQÜÊNCIA

AUTORA: Viviane Cátia Köhler
ORIENTADOR: Felipe Martins Müller

Data e Local da Defesa: Santa Maria, 11 de outubro de 2004.

Este trabalho propõe três estratégias evolutivas para resolver o problema de seqüenciamento de *n* tarefas em *m* máquinas paralelas idênticas, buscando minimizar o tempo máximo de finalização (makespan). São considerados tempos de preparação dependentes da seqüência. Os métodos propostos são comparados com outras duas heurísticas de qualidade comprovada, uma baseada em Busca Tabu e outra baseada em Algoritmos Meméticos. Para algumas instâncias de pequeno porte, comparações são feitas com o valor ótimo obtido através de uma busca dicotômica. Para instâncias maiores, as comparações demonstram a robustez e a boa qualidade das soluções encontradas pelas estratégias evolutivas através da comparação com as outras heurísticas.

Palavras-chave: Estratégias evolutivas, metaheurísticas, máquinas paralelas idênticas.

ABSTRACT

Master's Degree Dissertation
Posgraduation Program in Engineering of Production
Federal University of Santa Maria, RS, Brazil

ALGORITMOS EVOLUTIVOS PARA O PROBLEMA DE SEQÜENCIAMENTO DE TAREFAS EM MÁQUINAS PARALELAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQÜÊNCIA

(Evolutionary Algorithms for Parallel Machine Scheduling Problems with Sequence Dependent Setup Times)

Author: Viviane Cátia Köhler
Advisor: Felipe Martins Müller

Date and place of defense: Santa Maria, October 11, 2004.

This work presents three evolutionary strategies to solve the problem of scheduling a given set of n jobs to m identical parallel machines with the objective of minimizing makespan. There is a sequence dependent setup times. We also compares our method with two other well succeeded heuristics, one is a tabu search based heuristic and the second is a memetic approach, which combines a population-based method with local search procedures. As benchmarks for small-sized instances, optimal values are used provide by a dichotomous search. For larger instances, the comparisons try to show the robust behavior in solution quality as well as in computational effort of our evolutionary strategy.

Key Words: *evolutionary strategies, scheduling problems, identical parallel machines.*

SUMÁRIO

1 INTRODUÇÃO

A otimização da seqüência e da execução de tarefas (*Sequencing and Scheduling Problems*) vem sendo cada vez mais estudada, devido ao crescimento do mercado e à escassez de recursos (máquinas, mão-de-obra, dinheiro, entre outros), gerando a necessidade de decidir a ordem em que as tarefas devem ser realizadas. Com isso, as decisões referentes à solução desses conflitos revelam questões altamente combinatórias, ou seja, como as tarefas devem ser arranjadas ou seqüenciadas de forma a maximizar os lucros da empresa.

A alocação das tarefas nas máquinas está voltada ao seqüenciamento operacional, e não às decisões táticas, tais como determinação de datas de entrega, ou às decisões estratégicas, como a aquisição de máquinas.

Problemas de otimização combinatória surgem quando é preciso selecionar, a partir de um conjunto discreto e finito de dados, o melhor subconjunto que satisfaça determinados critérios predefinidos. (Goldbarg & Luna, 2000)

Os problemas de otimização têm por finalidade encontrar uma solução, que é a minimização/maximização de um ou mais critérios de otimalidade (ou otimização), com o objetivo de encontrar um ponto ótimo (mínimo/máximo) de uma função definida sobre um domínio contínuo ou discreto. Embora os elementos do domínio possam, em geral, ser facilmente enumerados, a idéia ingênua de testar todos os elementos na busca pelo melhor mostra-se inviável na prática, pois o domínio é associado a uma dimensão finita elevada ou até infinita.

Segundo a teoria da complexidade computacional, sistematizada por Garey & Johnson (1979), os problemas de otimização combinatória

podem ser classificados de acordo com sua complexidade em P, NP, NP-completo e NP-difícil. Os problemas tratáveis computacionalmente são pertencentes à classe P. Já os demais são considerados intratáveis, ou seja, o número de operações primitivas (atribuição, soma, entre outras operações) executadas no melhor algoritmo conhecido para o problema cresce exponencialmente em função do tamanho da instância.

Segundo Müller (1993), a classificação do problema de seqüenciamento de tarefas em máquinas paralelas idênticas com tempos de preparação dependentes da seqüência, na teoria de complexidade computacional, é NP-difícil num sentido forte, pois inclui um *problema de caixeiro viajante*, que é NP-completo, como um caso especial.

Existem duas formas de resolução desses problemas, que são: os métodos exatos e os métodos heurísticos. Os métodos exatos, em geral, possuem um tempo de execução proibitivo, enquanto os métodos heurísticos possuem um tempo de execução razoável. No entanto, estes últimos possuem a desvantagem de não possuir a garantia de encontrar uma solução ótima global.

Devido à elevada complexidade computacional dos problemas de otimização combinatória [NP-difíceis (Garey & Johnson, 1979)], são utilizados algoritmos heurísticos para auxiliar na resolução. Nas situações em que os problemas são NP-difíceis, pode ser vantajoso sacrificar a otimalidade e ter uma solução boa em um tempo computacional razoável, que é o paradigma das heurísticas (perde num, mas ganha no outro). Convém observar que em uma heurística não se tem a garantia de encontrar um elemento do domínio, cujo resultado guarda uma relação preestabelecida com o valor ótimo. Atualmente, as empresas em muitos casos, necessitam saber em poucos segundos qual a melhor forma de seqüenciar as tarefas, de tal forma que obtenham maiores lucros sobre a produção. Desta forma, se valida à utilização de métodos heurísticos.

Para a resolução dos problemas de otimização, também são utilizados limitantes e condições dominantes para reduzir o espaço de busca. Um bom procedimento a ser adotado é avaliar todos os objetivos, examinar o núcleo do problema, identificar e descartar aspectos irrelevantes da investigação numa etapa de pré-processamento.

Dentre os métodos exatos para resolver problemas de otimização combinatória, podem-se citar: a programação linear inteira mista e a programação dinâmica e métodos de enumeração implícita, tais como *branch and bound*, entre outros.

As heurísticas construtivas são os procedimentos que, passo a passo, vão construindo uma solução para o problema. As heurísticas de melhoramento partem de uma solução inicial e procuram, através de trocas e realocações, melhorar a qualidade da solução. Como exemplo de heurísticas construtivas para o problema do caixeiro viajante, temos: vizinho mais próximo, vizinho mais distante, inserção mais próxima, inserção mais barata, inserção mais distante, entre outros. Para heurísticas de melhoramento para o problema do caixeiro viajante, podemos citar: GENIUS (Gendreau *et al*., 1992), 2-Opt, 2-Opt* (Potvin & Rousseau, 1995) e Or-Opt (Or, 1976) , entre outros.

Metaheurísticas são procedimentos mestres que guiam e modificam a operação de heurísticas subordinadas para produzir soluções aproximadas para problemas combinatórios difíceis, este métodos possuem ferramentas que permitem tentar escapar das armadilhas impostas pelos ótimos locais ainda distantes de um ótimo local e evitar também a formação de ciclos no processo de busca. Como exemplo de metaheurísticas, temos: Busca Tabu, *Simulated de Annealing*, GRASP (*Greedy Randomized Adaptive Procedure*), Estratégias Evolutivas, Algoritmos Genéticos, VNS, Ant Colony Systems, entre outras.

As heurísticas e metaheurísticas para problemas de otimização

combinatória, é importante executar, desenvolver e comparar, de modo sistemático, diferentes algoritmos, estratégias e parâmetros para o mesmo problema (Mendes *et al.*, 2002).

O problema em estudo nesta dissertação é o seqüenciamento de tarefas em máquinas paralelas idênticas com tempos de preparação dependentes (*setup times*) da seqüência. De acordo com a classificação de três campos proposta por Graham *et al.* (1979), ele foi categorizado como P|tpds|Cmax. Formalmente, esse problema de seqüenciamento considera um conjunto de *n* tarefas a serem alocadas em *m* máquinas paralelas idênticas, com o objetivo de minimizar o tempo máximo de finalização das tarefas (*makespan*).

Este problema de seqüeciamento aproxima-se freqüentemente das situações encontradas nas aplicações reais, em que os tempos de preparação são significativos, como os encontrados em indústrias químicas, de refrigerantes, têxteis e de papel.

São propostas, neste trabalho, três estratégias evolutivas que foram desenvolvidas a partir das idéias de Homberger & Gehring (1999), para resolver problemas de roteamento de veículos com janelas de tempo. A utilização das idéias desses autores se deu devido à semelhança entre os problemas.

A apresentação da dissertação está organizada da seguinte forma:

No capítulo 2, é definido o P|tpds|Cmax: conceito, uma formulação matemática e métodos de resolução. Também nesse capítulo é feita a revisão bibliográfica dos métodos heurísticos e exatos, propostos para o problema em questão, com o objetivo de situar adequadamente as contribuições propostas na dissertação.

O capítulo 3 inicia com uma revisão de literatura das Estratégias Evolutivas (*EEs*) e descreve como foram feitas as adaptações para o

P|tpds|Cmax.

O capítulo 4 apresenta os resultados obtidos pelas metaheurísticas utilizadas para comparação, busca tabu e memético, bem como os resultados das estratégias evolutivas propostas.

Na última parte há a conclusão e sugestões para trabalhos futuros.

2 PROBLEMA P|tpds|Cmax

O seqüenciamento em máquinas paralelas idênticas com tempo de preparação das tarefas (P|tpds|Cmax) é um problema de otimização combinatória pouco estudado, apesar de possuir inúmeras aplicações práticas. Isso se deve ao fato dele ser simples de descrever, mas muito difícil de se resolver.

A preparação (*setup time)* das tarefas consome tempo significativo da máquina, pois serve para prepará-la para receber a tarefa a ser executada. Pode-se citar como exemplo destas operações a preparação da máquina, a limpeza, a troca de peças, as ferramentas ou o desenho, aquecimento/resfriamento, os ajustes de um modo geral, entre outros.

O tempo de preparação pode representar, em certas situações, uma parcela significativa do tempo necessário para uma máquina executar um dado conjunto de tarefas. A redução desses tempos de preparação é freqüentemente alcançada quando uma fábrica adota um *lay out* que aloca as máquinas em células de produção, usando tecnologia de grupo e classificação por famílias de tarefas. (Burbidge, 1975 *apud* Müller, 1993)

O P|tpds|Cmax pertence à classe de problemas NP-difícil, ou seja, o tempo gasto para resolvê-lo pode ser exponencial em relação ao tamanho da instância. Devido a isso, a resolução do problema utilizando métodos heurísticos ganha maior importância, principalmente quando aplicado a instâncias grandes e/ou difíceis.

Este capítulo tem como objetivo conceituar o problema, apresentando uma formulação matemática e os métodos de resolução.

2.1 Definição

Dado um conjunto de ***n*** tarefas, $T_1, T_2,...,T_n$ e ***m*** máquinas paralelas idênticas, $M_1, M_2,...,M_m$, o problema P|tpds|Cmax consiste em alocar as tarefas nas máquinas de uma forma não preemptiva (uma vez determinado que uma tarefa esteja alocada a uma máquina, ela deverá permanecer na mesma até o seu término, sem interrupções), com o objetivo de minimizar o tempo de finalização de todas as tarefas (*makespan)*. Nesse problema, o tempo de processamento das tarefas T_j é composto da soma de duas parcelas: o tempo de execução dado por um número inteiro e positivo p_j e um tempo de preparação também inteiro e positivo para a tarefa T_j, se ela foi executada imediatamente após a tarefa *i*, chamado s_{ij}. Assim, o tempo de processamento de T_j logo após T_i pode ser expresso como: $t_{ij} = p_j + s_{ij}$. Para esse problema, considera-se que não existe nenhuma relação de simetria, ou seja, $s_{ij} \neq s_{ji}$.

2.2 Formulação matemática

Uma formulação matemática para o problema de seqüenciamento em máquinas paralelas idênticas é apresentada:

Função objetivo: Minimizar *C*

Sujeito a:

$$x_{kk} = 0, \qquad k = 1,...,n \qquad (1)$$

$$\sum_{\substack{i=2 \\ i \neq k}}^{n} x_{ik} = 1 \qquad k = 1,...,n \qquad (2)$$

$$\sum_{\substack{j=2 \\ j \neq k}}^{n} x_{kj} = 1 \qquad k = 1,...,n \qquad (3)$$

$$u_k \leq C \qquad k = 2,...,n \qquad (4)$$

$$u_k \geq t_{1k} x_{1k} \qquad k = 2,...,n \qquad (5)$$

$$u_k \leq M - (M - t_{1k}) x_{1k} \qquad k = 2,...,n \qquad (6)$$

$$u_k \geq u_i + t_{ik} - M + M(x_{ki} + x_{ik}) - (t_{ik} + t_{ki}) x_{ki} \qquad \begin{matrix} i = 2,...,n \\ k = 2,...,n \\ i \neq k \end{matrix} \qquad (7)$$

$$\sum_{j=2}^{n} x_{1j} \leq m \qquad (8)$$

$$x_{ij} \in \{0,1\} \qquad \begin{matrix} i = 1,...,n \\ j = 1,...,n \end{matrix} \qquad (9)$$

$$u_k \geq 0 \qquad k = 2,...,n \qquad (10)$$

$$C \geq 0 \qquad (11)$$

A função objetivo requer a minimização da carga da máquina mais carregada. Existem *n* tarefas e *m* máquinas. A variável inteira $x_{ij} = 1$ indica que a tarefa *j* será executada logo após a tarefa *i* e $x_{ij} = 0$. Caso contrário, a variável x_{11} é uma variável artificial da qual partem as seqüências de tarefas correspondentes a cada máquina. A variável u_k corresponde à soma parcial da carga da máquina na qual a tarefa *k* está alocada, considerando a seqüência de tarefas até *k.* M é um número "grande" e é utilizado para "ligar" e "desligar" as restrições (6) e (7).

As equações (1) afirmam que uma tarefa não pode ser predecessora ou sucessora dela mesma, enquanto que as equações (2) e (3) permitem que cada tarefa possui exatamente um sucessor e um predecessor, respectivamente.

Nas inequações (4) a variável *C* é limitada superiormente pelo valor do *makespan*. Observe-se que a última tarefa k_0 na seqüência da máquina que determina o *makespan* é associada ao valor u_{k_o} que corresponde à carga total desta máquina.

As desigualdades (5) e (6) determinam a tarefa que iniciará uma seqüência.

As restrições (7) servem para evitar ciclos na seqüência em que uma nova tarefa será adicionada.

As restrições (8) indicam que nunca poderão ser utilizadas mais que *m* máquinas para alocar as *n* tarefas.

Após definir formalmente o problema, apresentamos alguns métodos de resolução.

2.3 Métodos de resolução

Os problemas de otimização combinatória discutidos nesta dissertação utilizarão métodos exatos e heurísticos.

Os métodos exatos dão a garantia de encontrar uma solução ótima, contudo, o tempo de execução do algoritmo pode tornar-se "proibitivo", de acordo a dimensão da instância.

Devido aos elevados tempos exigidos pelo método exato e à necessidade de se obterem resultados de forma mais rápida, iniciou-se o estudo das heurísticas que tendem a encontrar uma resposta de boa qualidade ao problema. No entanto, não se tem certeza se este resultado representa o valor ótimo, ou seja, não se tem prova de convergência para a solução ótima. As heurísticas são procedimentos que resolvem o problema através de um enfoque intuitivo, em geral racional, no qual a estrutura do problema possa ser interpretada e explorada inteligentemente para

obter uma solução razoável. Os métodos heurísticos ajudam a escolher um caminho entre os vários possíveis em busca de um determinado objetivo, sem os compromissos de percorrer todos os caminhos ou de escolher o melhor deles a cada decisão, encontrando uma solução (factível ou não), não necessariamente ótima, computado em tempo aceitável.

Conforme Osman (1991), as heurísticas podem ser divididas em construtivas, de melhoramento, de programação matemática, particionamento, com restrição do espaço de soluções, com relaxação do espaço de soluções, com algoritmos compostos e metaheurísticas. Neste trabalho, serão implementados algoritmos construtivos e de melhoramento, que, inseridas em estratégias evolutivas, produzem procedimentos ainda não encontrada na literatura para a resolução deste problema.

Heurísticas construtivas são procedimentos iterativos que, passo a passo, iteração a iteração, vão adicionando componentes (tarefas, arcos, nós, variáveis, entre outros, de acordo com o problema que está sendo resolvido) à solução parcial para o problema. Somente ao final do algoritmo se obtém uma solução completa para o problema que pode ser, inclusive, infactível. O problema é factível quando todas as restrições estão sendo respeitadas; caso contrário, a solução é dita infactível.

A heurística de melhoramento parte de uma solução factível, buscando, através de uma estratégia, soluções de melhor qualidade. O algoritmo pára, quando, a partir da solução corrente, não se encontra uma solução melhor com as tentativas propostas pelo algoritmo de melhoramento. Nesse instante, diz-se que o algoritmo atingiu um ótimo local. Essa heurística também é encontrada com a denominação de heurística de busca local ou de vizinhança.

A metaheurística é um processo que organiza e direciona heurísticas subordinadas pela combinação de diferentes conceitos, tentando evitar parada prematura em ótimo local ainda distante de um ótimo global.

Metaheurísticas podem manipular uma solução completa, incompleta ou um conjunto de soluções, permitindo a deterioração controlada de soluções para diversificar a busca e escapar de um ótimo local de baixa qualidade.

As buscas utilizadas em metaheurísticas podem ser divididas em duas categorias: uma delas compreende os métodos que exploram várias soluções a cada iteração (algoritmos populacionais), e a outra compreende os métodos que exploram apenas um elemento de uma vizinhança a cada iteração.

Os algoritmos populacionais exploram uma população de soluções a cada iteração. Essa estratégia de busca é capaz de explorar várias regiões do espaço de soluções a cada execução. Assim, não se constrói uma única trajetória de busca, pois as novas buscas são combinações das soluções anteriores. Nesta categoria, atualmente, existem várias metaheurísticas. Dentre as mais conhecidas, podem-se citar: os algoritmos genéticos (Goldberg, 1989), os algoritmos meméticos (Moscato, 1989, 1999), *Scatter Search* (Glover, 1990), *Ant Colony Systems* (Stützle e Dorigo, 1999) e as estratégias evolutivas que foram inicialmente criadas para problemas de otimização no caso contínuo (Rechenberg, 1973) e que, atualmente, estão sendo aplicadas para problemas de otimização discreta (Cai & Thierauf,1995; Homberger & Gehring, 1999; Gupta & Greenwood,1996).

Algoritmos com apenas um elemento da vizinhança a cada iteração geram um caminho, ou uma trajetória de soluções, obtido pela transição de uma solução a outra, de acordo com os movimentos permitidos pela metaheurística. Como exemplo dessa classificação, têm-se: *Simulated Annealing* (Kirkpatrick *et al.*, 1983), Busca Tabu (Glover & Laguna, 1993; Fiechter, 1994), GRASP (*Greedy Randomized Adaptive Procedure*) (Feo & Resende, 1994), Redes Neurais (Potvin, 1993) e, recentemente, *Simu-*

lated Jumping (Amin, 1999).

2.4 Revisão de literatura do P|tpds|Cmax

O problema P|tpds|Cmax, em um caso particular de apenas uma máquina (*m=1*), é equivalente ao problema do caixeiro viajante – PCV (Baker, 1974). O PCV é assim definido: dado um caixeiro viajante, esse deve visitar *n* cidades, passando em cada uma delas uma vez e retornando à cidade de origem. Dadas as distâncias entre os pares de cidades, a tarefa do vendedor é encontrar uma rota que minimize a distância total percorrida. O PCV pode ser simétrico ($d_{ij} = d_{ji}$) ou assimétrico ($d_{ij} \neq d_{ji}$), onde d_{ij} é a distância entre as cidades *i* e *j*. No problema P|tpds|Cmax, com *m=1*, t_{ij} é o tempo de processamento da tarefa J_j, se ela é executada imediatamente após a tarefa J_i, ou seja, t_{ij} corresponde a d_{ij}. Assim, o P|tpds|Cmax, com *m = 1,* pode ser simétrico ou assimétrico (Müller, 1993).

Como o P|tpds|Cmax com $m = 1$ é equivalente ao PCV, poder-se-ia imaginar uma equivalência entre o P|tpds|Cmax com $m > 1$ e o problema dos *m* caixeiros viajantes (*m-PCV* ou *Minsum m-PCV*), que é definido como sendo o problema de determinar *m* rotas disjuntas, de modo que a distância total percorrida pelos *m* caixeiros viajantes seja minimizada. Isso permite dizer que o problema *Minsum m-PCV* é equivalente ao problema de seqüenciamento que tenha como objetivo minimizar o tempo total de processamento, ou seja, minimizar a distância total de todas as rotas. No caso aqui tratado, e mantendo a analogia descrita acima, o P|tpds|Cmax com $m > 1$ procuraria minimizar a rota mais longa (*makespan*).

2.4.1 Método Exato

Um método exato proposto para resolver o P|tpds|Cmax foi desenvolvido utilizando idéias de problemas de *m* caixeiros viajantes devido a certa semelhança observada deste problema com uma variante do problema de caixeiro viajante já resolvido por algoritmos exatos. Trata-se do Problema de Roteamento de Veículos (PRV) com restrição de tamanho da Rota – DVRP (*Distance-Constrained Vehicle Routing Problem*). O PRV pode ser visto como um *m-TSP,* onde a cada rota de caixeiro viajante está associada uma determinada capacidade. O DVRP é semelhante ao *Min-sum m-TSP,* exceto que, nele, cada rota é restrita a não exceder uma determinada distância máxima, dita *Dist*. Assim, um esquema ótimo pode ser concebido através da resolução de uma série de DVRPs, ajustando a restrição sobre as rotas (distância máxima de cada rota) a cada DVRP resolvido. Detalhando a idéia, assuma que C* é a solução ótima do P|tpds|Cmax. Se a solução do DVRP para um dado valor de *Dist* (distância máxima permitida para a rota mais longa) for factível, então *Dist* é um limitante superior de C*; se *S* é a soma de todos os arcos (ligações entre as tarefas, t_{ij}) usados na solução do DVRP, então $\left\lceil \frac{S}{m} \right\rceil$ é um limitante inferior de C*. Se o DVRP é infactível, então *Dist* +1 é o limitante inferior de C*.

Partindo destes conceitos, desenvolveram-se dois esquemas ótimos para a solução do problema P|tpds|Cmax: um baseado em uma *Busca Dicotômica* e outro baseado em um esquema de diminuição da distância máxima permitida de uma unidade por vez. Esse último é chamado de *Busca Subtrativa*. Neste trabalho, foi utilizada a *Busca Dicotômica* para encontrar a solução ótima das instâncias testadas, que pode ser descrita da seguinte forma:

Passo 1: Usando a heurística de Busca Tabu desenvolvida por Müller (1993), obtenha um limitante superior LS, a solução encontrada

pela metaheurística para o problema P|tpds|Cmax.

Passo 2: Resolva o DVRP associado, usando como restrição de máxima distância por rota o limitante LS; portanto, *Dist* = LS.

Passo 3: Faça LS assumir o valor do tamanho da rota mais longa na solução do DVRP, sendo *S* a soma do custo de todos os arcos (ligações entre tarefas, t_{ij}) usados na solução do DVRP; faça LI = $\left\lceil \frac{S}{m} \right\rceil$.

Passo 4: Se LS = LI, a solução ótima do P|tpds|Cmax foi obtida, então PARE. Caso contrário faça: *Dist* = $LI + \left\lceil \frac{LS - LI}{2} \right\rceil$.

Passo 5: Resolva o DVRP associado, usando *Dist* como restrição de máxima distância por rota.

- Se o problema é FACTÍVEL, faça:

LS = tamanho da rota mais longa na solução do DVRP.

LI = $\left\lceil \frac{S}{m} \right\rceil$

- Se o problema é factível, faça LI = *Dist* + 1.

Volte para o **Passo 4.**

O esquema acima apresentado necessita resolver O (log (LS-LI+1)) DVRPs (Sedewick; 1990), onde LS e LI são definidos no Passo 3.

2.4.2 Métodos Heurísticos para o P|tpds|Cmax

Apesar da aplicabilidade do P|tpds|Cmax, poucos algoritmos têm sido propostos para resolver esse problema (Mendes *et al*., 2002). Dearing e Henderson (1984) desenvolveram um método de programação inteira para alocação de tarefas aos teares em indústrias têxteis. Eles apre-

sentaram os resultados encontrados através de uma relaxação linear do modelo inteiro. Sumichrast & Baker (1987) propõem uma heurística baseada na resolução de uma série de subproblemas 0-1 inteiros que melhoraram os resultados de Dearing & Henderson (1984). Essas duas publicações foram feitas baseadas em tempos de preparação simétricos, satisfazendo a desigualdade do triângulo ($s_{ij} \leq s_{ik} + s_{kj}$). Neste caso, restrições de demanda também são consideradas. O método proposto por eles se mostrou apropriado para realizar o seqüenciamento em locais onde existe um grande número de tarefas distintas e altos custos de preparação. O método não resolve o problema de programação linear inteira associado como um todo, pois isso tornaria o tempo computacional proibitivo. Ao invés disso, ele considera somente as máquinas onde as tarefas necessitam ser trocadas, gerando uma série de problemas de programação linear 0/1. O método trabalha bem, quando não há um número grande de produtos deficientes (isto é, que não alcançam a demanda no horizonte de tempo determinado), pois isso aumenta o número de máquinas a serem consideradas, aumentando o número de problemas de programação linear 0/1 resolvidos.

No caso assimétrico, Guinet (1991) publicou vários algoritmos para problemas encontrados na indústria têxtil, onde um deles é semelhante ao P|tpds|Cmax. Porém, o objetivo é minimizar o somatório dos tempos de finalização de cada tarefa, ao invés do tempo máximo de processamento (*makespan*). À primeira vista, poder-se-ia concluir que os dois objetivos são equivalentes, mas nem sempre são. Isso pode ser observado no exemplo dado por Müller, 1993.

Parker, Deane & Holmes (1977) também publicaram um algoritmo com o objetivo de minimizar os tempos de preparação, porém os tempos de processamento das tarefas não estão a eles incorporados, como o caso desta dissertação. Eles impõem uma restrição de carga limitada à capacidade de cada máquina (essa restrição está baseada apenas nos

tempos de processamento, não levando em conta os tempos de preparação). O que eles propõem é uma equivalência com o Problema de Roteamento de Veículos (PRV) que é resolvido por uma heurística baseada no algoritmo de clássico de economias (*Savings*), proposta por Clarke & Wright (1964).

Müller (1993) propôs uma busca tabu, que explora o espaço de solução vizinha guiada para busca local, onde seu principal componente é o uso de memória adaptativa para criar uma busca mais flexível. A proposta de busca tabu é composta por três fases, onde:

Fase 1: Uma solução inicial é construída, alocando todas as tarefas em *m* máquinas.

Fase 2: Uma busca local controlada pela busca tabu é usada para encontrar uma melhor solução em termos de *makespan,* movendo as tarefas entre diferentes máquinas.

Fase 3: Esta fase final vai tentar melhorar a seqüência da máquina mais carregada, para melhorar o *makespan*.

Dentro do esquema de Busca Local e com a supervisão da metaheurística Busca Tabu, soluções vizinhas são exploradas, removendo uma tarefa de uma máquina e reinserindo-a em outra máquina. Na fase 3, um procedimento de pós-otimização é aplicado a cada máquina em separado.

As inserções são feitas usando a idéia proposta para caixeiro viajante simétrico por Gendreau, Hertz & Laporte (1992) de um procedimento geral de inserção denominado GENI (de *GENeralized Insertion*), que é menos míope e mais poderoso que o procedimento padrão de inserção (por exemplo, 2-opt). GENI permite que uma tarefa seja incluída no processador, contendo pelo menos um de seus vizinhos mais próximos, e cada inserção é executada simultaneamente com uma reotimização local

da seqüência de tarefas do processador em questão (Müller, 1993).

A fase de pós-otimização aplica o US (*Unstringing – Stringing*) também proposto pelos mesmos autores de GENI. O procedimento US remove, sucessivamente, tarefas de uma máquina, usando o reverso de GENI e a reinsere na mesma máquina usando o GENI.

O GENI é eficiente, pois o número de inserções candidatas fica restrito à vizinhança do elemento a ser adicionado. Outro fator importante é que a progressão da busca é limitada aos movimentos mais promissores, uma vez que a reordenação dos elementos e suas reconexões são feitas levando-se em conta a proximidade dos mesmos.

Na busca tabu desenvolvida por Müller *et al*. (1993), foi utilizada a busca local GENIUS adaptada para o P|tpds|Cmax, que, dentre outras características, é assimétrico, ou seja, $t_{ij} \neq t_{ji}$, solução vizinha obtida pela remoção de uma tarefa da máquina mais carregada e inserindo em outra máquina. A vizinhança é construída usando os conceitos de vizinhança local e global, onde se considera como vizinho global da tarefa T_i as suas mais próximas tarefas sucessoras e predecessoras dentre todas as tarefas, exceto ela própria (T_i).

Outra publicação para o P|tpds|Cmax foi feita por Mendes *et al*. (2002) que propõe uma algoritmo genético híbrido, também conhecido como genético com busca local ou algoritmo memético (AM) (Moscato, 1989). Em um AM, todos os indivíduos de uma população evoluem enquanto se procura um mínimo local de uma certa vizinhança. Após passos de recombinação e mutação, uma busca local é aplicada a soluções resultantes. Uma introdução mais completa e formal para AM pode ser encontrada em Moscato (1999).

O AM de Mendes *et al*. (2002), proposto para o P|tpds|Cmax, utiliza uma população estruturada hierarquicamente, organizada como uma ár-

vore ternária de indivíduos agrupados em quatro subpopulações ou grupos. Ao contrário do que acontece com as populações não estruturadas, as possibilidades de *crossover* ficam restritas. Outros estudos têm mostrado que a utilização de populações estruturadas apresenta mais eficiência quando comparadas com populações não estruturadas. (Buriol *et al*. 1999)

A estrutura populacional utilizada possui uma única população estruturada segundo uma árvore ternária. Desta forma, podemos dividi-la em subgrupos de 4 indivíduos, cada um composto por um líder e três seguidores (veja Figura 1). Em cada subgrupo, o líder é sempre o indivíduo mais adaptado dos quatro. Esta hierarquia faz com que indivíduos dos níveis superiores sejam em geral melhor adaptados que os dos níveis inferiores. Pode-se concluir facilmente que a melhor solução estará sempre presente com o líder do subgrupo "raiz" da árvore populacional.

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

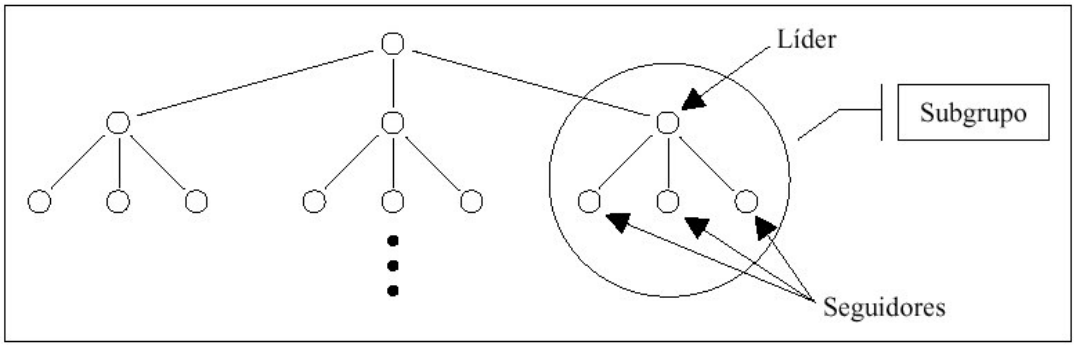

Figura 1 – Estrutura populacional em árvore ternária.

A recombinação usada por Mendes *et al*. (2002) foi o *Order Crossover* (OX). A mutação utilizada foi a tradicional, baseada na troca de tarefas. Como este problema tem o objetivo de minimizar o *makespan*, a função de *fitness* pode ser dada como: $f_i = (makespan)^{-1}$, onde *makespan*

é relativo à solução *i*.

A recombinação é permitida entre o líder e seus seguidores, que são selecionados aleatoriamente.

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

Utiliza-se um mecanismo de intensificação sobre o melhor indivíduo: ele é escolhido aproximadamente dez por cento a mais que os outros líderes.

Cada novo indivíduo gerado tem duas opções de inserção na população principal. Se seu *fitness* for melhor que o do líder do subgrupo, ele toma o seu lugar. Caso contrário, ele toma o lugar do seguidor que participou do *crossoveer* que lhe deu origem, independentemente de seu *fitness*. Se o novo indivíduo já está presente na população, ele não é inserido. Uma vez inseridos todos os novos indivíduos não-duplicados, inicia-se a etapa de atualização da população, caracterizada pela reestruturação da mesma. Como a árvore deve manter a hierarquia entre líderes e seguidores, verifica-se se algum indivíduo se tornou melhor adaptado que o líder do seu subgrupo. Caso isso ocorra, eles trocam de posição. Ao efetuar essa comparação para todos os subgrupos, mantém-se a consistência hierárquica da população.

Algoritmos de busca local se baseiam na definição de uma vizinhança que estabelece uma relação entre soluções no espaço de configurações. Mendes *et al*. (2002) propõe duas vizinhanças, além de diversas políticas de redução. A vizinhança de uma solução pode ser descrita

como o conjunto de soluções que são alcançáveis através da execução de qualquer movimento definido pelo mecanismo de geração da vizinhança.

Uma das vizinhanças propostas por Mendes *et al*. (2002) foi a troca *all-pairs*. Ela consiste em trocar de forma ordenada todos os pares de tarefas em uma dada solução. Seja a seqüência < D_ E_ F_ G_ H_ I_ J_ K >. Inicialmente trocamos as posições (D,E), gerando a solução < E_ D_ F_ G_ H_ I_ J_ K >. Se a troca for vantajosa, confirmamos a mudança e continuamos o processo. Caso contrário, a troca é desfeita. Continuando na seqüência, trocamos (D,F), (D,G), até o par (D,K). O processo é reiniciado a partir da segunda posição: E. Trocamos (E,F), (E,G), até (E,K). O processo termina na troca das posições (J,K). Ao terminar uma passagem completa, verificamos se alguma mudança foi vantajosa. Em caso positivo, realizamos uma nova passagem, pois a solução não é necessariamente um mínimo local. Se nenhuma troca foi confirmada em uma passagem completa, então a solução é um mínimo local e podemos parar.

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

Os parâmetros escolhidos empiricamente por Mendes *et al*. (2002) foram o tamanho da população, a razão de *crossover* e a mutação. De acordo com o número de tarefas, temos, na tabela abaixo, os parâmetros definidos por Mendes *et al.*

Tabela 1 - Parâmetros utilizados no AM.

n	*Tamanho da população*	*Crossover*	*Mutação*
20	121	0,2	0,1
40	40	0,2	0,1
60	13	0,3	0,0
80	13	0,2	0,0

Fonte: Mendes *et al*. (2002)

Ravetti (2003) analisa e resolve um problema real de seqüenciamento de tarefas com máquinas paralelas e tempos de preparação dependentes da seqüência, através da programação linear inteira e mista, de programação por restrições e da metaheurística GRASP. Ele define três problemas, considerando diferentes graus de simplificação do caso real, os quais são analisados, e testa algumas variações do GRASP.

Neste capítulo, foi apresentada a definição e a formulação matemática para o P|tpds|Cmax. Também há a revisão bibliográfica dos métodos utilizados para resolver o problema em questão. No próximo capítulo, serão apresentadas as estratégias evolutivas propostas para o P|tpds|Cmax.

3 ESTRATÉGIAS EVOLUTIVAS

Neste capítulo, será feita a revisão parcial de literatura das estratégias evolutivas e a descrição de como foi realizada a adaptação das *EE* propostas por Homberger & Gehring (1999) para o problema P|tpds|Cmax. Também será apresentado um exemplo numérico de todas as metaheurísticas utilizadas neste trabalho.

3.1 Revisão de Literatura das Estratégias Evolutivas

Em 1964, no *Herman Föttinger Institute for Hydrodynamics* (Universidade Técnica de Berlim, Alemanha), teve origem a *estratégia evolutiva (EE)*. O problema original em estudo era o de encontrar formas otimizadas para objetos inseridos em túneis de vento. Estratégias usando o método do gradiente não foram bem sucedidas. Dois estudantes, Ingo Rechenberg e Hans-Paul Schwefel, tiveram a idéia de efetuar alterações aleatórias nos parâmetros que definiam a forma do objeto, baseados na idéia da seleção natural, o que resultou na teoria da velocidade de convergência para o mecanismo denominado (1+1), um esquema simples de seleção-mutação trabalhando em um único indivíduo que gera um único descendente por geração através da mutação Gaussiana. Mais tarde, esta teoria evoluiu para o chamado mecanismo $(\mu+1)$, no qual uma população de μ indivíduos se recombina de maneira aleatória para formar um descendente, o qual, após sofrer mutação, substitui (se for o caso) o pior elemento da população. Ainda que este mecanismo nunca tenha sido largamente usado, ele permitiu a transição para os mecanismos chamados $(\mu+\lambda)$ e (μ,λ), já no final dos anos 70. Na primeira, os μ ancestrais e os λ descendentes convivem, enquanto na segunda, os μ ancestrais "morrem", deixando apenas os λ descendentes "vivos".

A abordagem *EE* é adequada a uma vasta gama de problemas de

otimização, porque ela não necessita de muitas informações sobre o problema. Ela é capaz de resolver problemas multidimensionais, multimodais e não-lineares sujeitos a restrições lineares ou não-lineares. O algoritmo básico pode ser descrito como segue:

1. Uma dada população composta de μ indivíduos. Cada um é caracterizado pelo seu genótipo, consistindo de *n* genes, que determina de modo não ambíguo a aptidão para a sobrevivência.

2. Cada indivíduo da população produz λ/μ descendentes, na média, de modo que um total de indivíduos novos são gerados. O genótipo dos descendentes difere ligeiramente dos genótipos de seus ancestrais.

3. Apenas os μ melhores indivíduos dos λ gerados permanecem vivos, tornando-se os ancestrais na próxima geração.

Note-se que até recentemente a *EE* era conhecida apenas na comunidade de engenharia e tida como alternativa a soluções-padrão para problemas de otimização.

Estratégias Evolutivas são diferentes de Algoritmos Genéticos. Estes dois paradigmas foram desenvolvidos independentemente (um na Alemanha e outro nos Estados Unidos, respectivamente) e são filosoficamente muito diferentes. Os algoritmos genéticos enfatizam modelos com operadores genéticos, como os observados na natureza (a recombinação), enquanto estratégias evolutivas enfatizam operadores de mutação que mantêm a característica fenótipa entre pai e filho (Gupta e Greenwood, 1996).

3.2 Estratégias Evolutivas Propostas

A estratégia evolutiva proposta neste trabalho foi baseada nas idéias de Homberger e Gehring (1999) para resolver problemas de roteamento de veículos com janelas de tempo. A adaptação das idéias se deu devido à semelhança entre os problemas de roteamento e seqüenciamento, P|tpds|Cmax. Estamos propondo três *EEs* denominadas *ESched1*, *ESched2* e *ESched3,* que serão apresentados na seqüência.

As matrizes das Figuras 2 e 3 são quadradas de (*n*+*m*), sendo *n* o número de tarefas, e *m*, o de máquinas.

Na matriz da Figura 2, por exemplo, o tempo de preparação da máquina, para executar a tarefa 6, após a tarefa 5, é 2.

As colunas maiores que *n* são iguais a 0, porque as máquinas não são limpas após a execução da última tarefa.

Como as máquinas são paralelas idênticas, o tempo de preparação (limpeza) de uma é igual ao das outras (o que se mostra pela igualdade das linhas maiores que *n*). Esse tempo existe porque as máquinas têm de ser limpas antes da execução da primeira tarefa (eis que não o são após a última).

A diferença existente entre as matrizes das Figuras 2 e 3 consiste em que, nesta, acrescenta-se o tempo de execução da tarefa.

Na matriz da Figura 3, por exemplo, o tempo de preparação da máquina para a tarefa 6, acrescido do tempo de execução dessa tarefa, após a tarefa 5, é 6.

A instância que será utilizada como exemplo no decorrer da explicação dos métodos propostos utiliza duas máquinas e vinte tarefas, sendo o tempo de processamento das tarefas p_j = {45, 25, 52, 35, 60, 4, 41, 63, 20, 88, 69, 91, 65, 94, 8, 91, 6, 52, 71, 15}, e a matriz dos tempos de preparação da máquina para executar a tarefa:

s_{ij}	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	--	4	5	7	4	0	0	0	2	2	3	5	6	8	3	0	0	0	8	1	0	0
2	1	--	8	5	4	6	0	0	7	6	2	0	7	0	6	4	5	4	1	3	0	0
3	5	3	--	9	9	4	6	8	0	6	2	1	9	6	3	7	3	4	3	9	0	0
4	9	9	3	--	0	9	1	8	0	2	0	1	5	7	6	8	9	1	0	1	0	0
5	9	5	0	0	--	2	7	2	9	4	9	1	5	7	2	3	6	4	8	4	0	0
6	7	9	2	5	1	--	5	7	4	8	0	2	2	7	1	4	8	1	7	7	0	0
7	0	0	2	9	9	8	--	5	4	8	0	3	5	1	8	0	8	7	0	4	0	0
8	3	3	2	3	8	4	8	--	2	0	4	2	5	5	9	4	7	6	7	0	0	0
9	9	0	5	4	8	8	1	3	--	7	7	1	6	6	7	2	6	5	8	8	0	0
10	0	4	0	2	2	7	8	0	0	--	9	8	2	4	5	5	5	2	8	9	0	0
11	7	9	7	4	4	8	0	8	7	5	--	2	0	2	1	3	3	4	0	8	0	0
12	7	9	5	7	5	4	5	5	7	5	3	--	0	6	9	9	6	6	5	2	0	0
13	9	4	9	2	7	0	2	8	4	0	4	4	--	0	1	5	7	1	9	3	0	0
14	4	3	6	4	1	1	2	6	1	6	8	8	2	--	3	7	1	9	5	8	0	0
15	4	6	5	1	3	7	6	7	9	6	7	1	5	8	--	7	5	6	8	2	0	0
16	9	1	0	3	0	3	8	1	9	0	1	6	4	9	1	--	2	2	2	5	0	0
17	3	9	2	8	7	9	2	6	7	7	7	4	6	5	8	7	--	7	6	5	0	0
18	6	5	7	6	1	6	6	7	2	1	4	3	9	5	7	2	7	--	5	2	0	0
19	6	9	8	1	1	0	6	0	1	7	9	3	7	4	3	0	9	6	--	9	0	0
20	0	8	5	3	8	2	9	3	8	6	3	4	6	8	1	9	5	2	1	--	0	0
21	2	2	1	1	3	9	9	1	7	8	5	6	9	2	9	8	2	2	2	6	--	--
22	2	2	1	1	3	9	9	1	7	8	5	6	9	2	9	8	2	2	2	6	--	--

Figura 2 – Matriz do tempo de preparação das máquinas para alocar as tarefas.

t_{ij}	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	--	29	57	42	64	4	41	63	22	90	72	96	71	102	11	91	6	52	79	16	0	0
2	46	--	60	40	64	10	41	63	27	94	71	91	72	94	14	95	11	56	72	18	0	0
3	50	28	--	44	69	8	47	71	20	94	71	92	74	100	11	98	9	56	74	24	0	0
4	54	34	55	--	60	13	42	71	20	90	69	92	70	101	14	99	15	53	71	16	0	0
5	54	30	52	35	--	6	48	65	29	92	78	92	70	101	10	94	12	56	79	19	0	0
6	52	34	54	40	61	--	46	70	24	96	69	93	67	101	9	95	14	53	78	22	0	0
7	45	25	54	44	69	12	--	68	24	96	69	94	70	95	16	91	14	59	71	19	0	0
8	48	28	54	38	68	8	49	--	22	88	73	93	70	99	17	95	13	58	78	15	0	0
9	54	25	57	39	68	12	42	66	--	95	76	92	71	100	15	93	12	57	79	23	0	0
10	45	29	52	37	62	11	49	63	20	--	78	99	67	98	13	96	11	54	79	24	0	0
11	52	34	59	39	64	12	41	71	27	88	--	93	65	96	9	94	9	56	71	23	0	0
12	52	34	57	42	65	8	46	68	27	93	72	--	65	100	17	100	12	58	76	17	0	0
13	54	29	61	37	67	4	43	71	24	88	73	95	--	94	9	96	13	53	80	18	0	0
14	49	28	58	39	61	5	43	69	21	94	77	99	67	--	11	98	7	61	76	23	0	0
15	49	31	57	36	63	11	47	70	29	94	76	92	70	102	--	98	11	58	79	17	0	0
16	54	26	52	38	60	7	49	64	29	88	70	97	69	103	9	--	8	54	73	20	0	0
17	48	34	54	43	67	13	43	69	27	95	76	95	71	99	16	98	--	59	77	20	0	0
18	51	30	59	41	61	10	47	70	22	89	73	94	74	99	15	93	13	--	76	17	0	0
19	51	34	60	36	61	4	47	63	21	95	78	94	72	98	11	91	15	58	--	24	0	0
20	45	33	57	38	68	6	50	66	28	94	72	95	71	102	9	100	11	54	72	--	0	0
21	47	27	53	36	63	13	50	64	27	96	74	97	74	96	17	99	8	54	73	21	-	-
22	47	27	53	36	63	13	50	64	27	96	74	97	74	96	17	99	8	54	73	21	-	-

Figura 3 – Matriz do tempo de preparação das máquinas para alocar as tarefas acrescido do tempo de execução destas.

3.2.1 Algoritmo Construtivo

Antes de explicar o funcionamento das *EEs,* será apresentado o algoritmo construtivo que foi utilizado pelas estratégias. Como nas estratégias evolutivas é necessário um conjunto de soluções do problema, que é a população inicial do algoritmo, será necessário criar um mecanismo que ofereça a possibilidade de gerar soluções, e no caso aqui proposto

tem-se a restrição de não permitir indivíduos iguais na população inicial. Após a criação da população inicial, será excluída a restrição que não permite indivíduos iguais.

É utilizada, no método construtivo, uma lista de vizinhos mais próximos, que, neste caso, é uma lista ordenada de forma crescente dos tempos de preparação das tarefas. Caso ocorra mais de um vizinho mais próximo com o mesmo valor, será testado inicialmente o de menor índice. A lista de vizinhos mais próximos tem tamanho *n*.

O algoritmo construtivo proposto é o seguinte:

Passo 1: Seleciona aleatoriamente *m* tarefas;

Passo 2: Cada tarefa será alocada em uma máquina, ou seja, inicializará a máquina.

Passo 3: Faça $i = 1$;

Passo 4: Se $i = n$, então PARE; se não, vá para o Passo 5.

Passo 5: Faça *M-* ser a máquina menos carregada, a máquina que contenha o menor tempo de utilização.

Passo 6: Encontre T_j, que é o vizinho mais próximo da última tarefa alocada em *M-*.

Passo 7: Se T_j ainda não foi alocado em nenhuma das máquinas, aloque a tarefa em *M-*. Caso contrário, vá para o Passo 6 e encontre o próximo vizinho mais próximo.

Passo 8: Faça $i = i + 1$ e vá para o Passo 4.

No exemplo que segue, será utilizado $n + 1$, $n + 2$, ... , $n + m$ como indicadores de inicialização da máquina.

Exemplo do algoritmo construtivo:

Passo 1: Seleciona aleatoriamente *m* tarefas;

Foram escolhidas aleatoriamente as tarefas 2 e 12. Cada uma delas inicializará uma máquina.

Passo 2: Cada tarefa será alocada em uma máquina, ou seja, inicializará a máquina.

Máquina 1: 21 - 2 (Custo = 27)

Máquina 2: 22 - 12 (Custo = 97)

Passo 3: Faça $i = 1$;

Passo 4: Se $i = n$, então PARE; se não, vá para o Passo 5.

Passo 5: Faça *M-* ser a máquina menos carregada.

M- = 1

Passo 6: Encontre T_j, que é o vizinho mais próximo da última tarefa alocada em *M-*;

$T_j = 7$

Passo 7: Se T_j ainda não foi alocado em nenhuma das máquinas, aloque a tarefa em *M-*. Caso contrário, vá para o Passo 6 e encontre o próximo vizinho mais próximo.

Máquina 1: 21 - 2 - 7 (Custo = 27 + 41 = 68)

Máquina 2: 22 - 12 (Custo = 97)

Passo 8: Faça $i = 2$ e vá para o Passo 4.

Passo 4: Se $i = n$, então PARE; se não, vá para o Passo 5.

Passo 5: Faça *M-* ser a máquina menos carregada.

M- = 1

Passo 6: Encontre T_j, que é o vizinho mais próximo da última tarefa alocada em *M-*.

T_j = 1

Passo 7: Se T_j ainda não foi alocado em nenhuma das máquinas, aloque a tarefa em *M-*. Caso contrário, vá para o Passo 6 e encontre o próximo vizinho mais próximo.

Máquina 1: 21 - 2 - 7 - 1 (Custo = 68 + 45 = 113)

Máquina 2: 22 - 12 (Custo = 97)

Passo 8: Faça *i = 3* e vá para o Passo 4.

Passo 4: Se *i* = n, então PARE; se não, vá para o Passo 5.

Passo 5: Faça *M-* ser a máquina menos carregada.

M- = 2

Passo 6: Encontre T_j, que é o vizinho mais próximo da última tarefa alocada em *M-*.

T_j = 13

Passo 7: Se T_j ainda não foi alocado em nenhuma das máquinas, aloque a tarefa em *M-*. Caso contrário, vá para o Passo 6 e encontre o próximo vizinho mais próximo.

Máquina 1: 21 - 2 - 7 – 1 (Custo = 113)

Máquina 2: 22 - 12 - 13 (Custo = 97 + 65 = 162)

Passo 8: Faça *i* = 4 e vá para o Passo 4.

Este procedimento se repete até *i* = n, que resulta na alocação de todas as tarefas, que, neste caso, gerou a seguinte alocação:

Máquina 1: 21 - 2 - 7 - 1 - 6 - 11 - 19 - 8 - 20 - 15 - 4 - 5 -18 Custo = 496

Máquina 2: 22 - 12 - 13 - 10 - 3 - 9 - 16 - 17 - 14 Custo = 522

Resultando um *makespan* igual a 522.

3.2.2 ESched1

Para as estratégias evolutivas, é necessário definir o tamanho da população inicial *pop*, o tamanho da população, *popmax*, e seja *imax* o número de iterações evolutivas. Considerando o conjunto *MoveSet* = {2-opt*, Or-OPT, *two-interchange*}, onde o algoritmo *two-interchange* executa a remoção de duas tarefas J_i e J_j de duas diferentes máquinas M_i e M_j, respectivamente, realoque J_i e M_j e J_j em M_i utilizando inserção mais barata.

O Or-opt desenvolvido, utilizado neste trabalho, não testa a possibilidade de inversão da rota, e o procedimento fica da seguinte forma: selecionam-se três tarefas adjacentes e se tenta alocar esta seqüência em outro local, mas na mesma máquina. Após testar com três tarefas, será feita com duas tarefas. Caso exista uma troca, esta é feita, e se volta a fazer com três tarefas. Caso não tenha nenhuma troca, faz-se com uma tarefa. Se tiver uma troca com uma tarefa, esta será feita, e inicia-se o processo de procura com três tarefas. O procedimento pára quando não existem mais possibilidades de troca. Também é testada a possibilidade da troca dos indicadores de inicialização da máquina neste procedimento. Para melhor entendimento do procedimento utilizado, veja o exemplo abaixo:

Supomos uma máquina que possua a seguinte alocação:

21-19-6-5-3-9-2-12-13-14-17-20-21 Custo = 512

Primeiramente, é testada a seqüência 21-19-6, de modo que seja alocada entre 5 e 3, resultando num custo de 521, que é superior ao custo atual. Dessa forma, a troca não será realizada. Testa-se também esta seqüência de 3 tarefas entre 3 e 9, ou seja, testam-se todas as possibilidades de alocar 21-19-6 entre duas tarefas da máquina. Após testar esta seqüência de três tarefas, será testada a seqüência 19, 6 e 5, e se tentará alocar esta ordem de tarefas entre 3 e 9. Isso resultará num custo de 523,

que é superior ao custo atual. Dessa forma, a troca não será feita. Depois, entre 9 e 2, e assim por diante. Depois, testa-se outra seqüência de três tarefas, 6, 5 e 3, e assim por diante.

O 2-OPT* testa todas as possibilidades que cortam a seqüência das tarefas em um ponto em cada rota e as reconecta. É testada a possibilidade de reconexão entre as duas seqüências das máquinas, sem inverter o sentido original da seqüência das tarefas. Como exemplo para o 2-opt*, considere a seguinte alocação das tarefas nas máquinas:

Máquina 1: 21-1-6-11-7-2-8-20-15-4-5-16-17-18-21

Máquina 2: 22-13-10-3-9-12-19-14-22

O único caso em que ocorre uma troca é rompendo a seqüência da tarefa 8 e 20 na máquina 1, e, na máquina 2, entre as tarefas 9 e 12. Com a reconexão, teremos as seguintes seqüências de tarefas nas máquinas.

Máquina 1': 21-1-6-11-7-2-8-12-19-14-21 Custo = 516

Máquina 2': 22-13-10-3-9-20-15-4-5-16-17-18-22 Custo = 523

A descrição através de fluxograma de ***ESched1*** pode ser feita da seguinte forma:

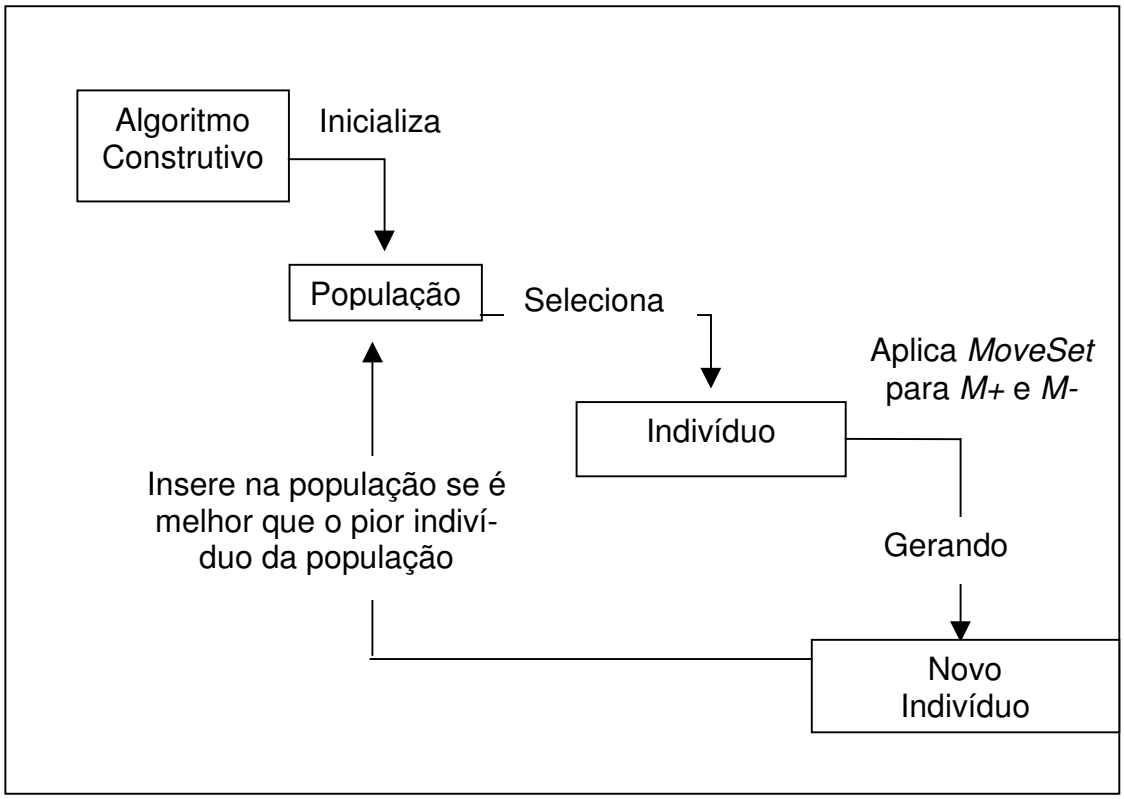

Figura 4 – Fluxograma de *ESched1*.

ESched1 pode ser formalmente descrito como segue:

Passo 1: Crie uma população inicial de tamanho *pop,* utilizando o Algoritmo Construtivo.

Passo 2: Faça *iter* =1.

Passo 3: Se *iter* > *imax,* então PARE. Caso contrário, escolha aleatoriamente um indivíduo I_j da população.

Passo 4: Faça *M+* e *M-* serem a máquina mais carregada e menos carregada do indivíduo I_j, respectivamente.

Passo 5: Escolha, aleatoriamente, do conjunto *MoveSet*, um algo-

ritmo de melhoramento.

Passo 6: Execute o algoritmo selecionado no passo 5 para *M+* e *M-*, gerando um novo indivíduo I_n.

Passo 7: Se (*pop + iter)* ≤ *popmax*, então insira I_n na população. Caso contrário, se I_n é melhor que o pior indivíduo na população, então troque-o por I_n.

Passo 8: Faça *iter* = *iter* +1 e vá para o **Passo 3**.

ESched1, de acordo com o exemplo dado acima, é resolvido da seguinte forma:

pop = 5; *popmax* = 10; *imax* = 15;

Passo 1:

Indivíduo 1:

Máquina 1: 21-2-7-1-6-11-19-8-20-15-4-5-18 Custo = 496

Máquina 2: 22-12-13-10-3-9-16-17-14 Custo = 522

Indivíduo 2:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-11-7-1-8-10-4-18-16-15 Custo = 503

Indivíduo 3:

Máquina 1: 21-1-6-11-7-2-8-20-15-4-5-16-17-18 Custo = 530

Máquina 2: 22-13-10-3-9-12-19-14 Custo = 500

Indivíduo 4:

Máquina 1: 21-2-7-1-6-11-10-8-20-15-4-18-16 Custo = 543

Máquina 2: 22-5-3-9-12-13-14-17-19 Custo = 470

Indivíduo 5:

Máquina 1: 21-3-9-2-7-1-8-20-15-12-13-14 Custo = 522

Máquina 2: 22-19-6-11-10-4-5-16-17-18 Custo = 492

Passo 2: *iter* =1.

Passo 3: Se *1 > 15,* então PARE. Caso contrário, escolha aleatoriamente um in-

divíduo I_j da população.

Indivíduo selecionado é o 2.

Indivíduo 2:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20-21 Custo = 512

Máquina 2: 22-11-7-1-8-10-4-18-16-15-22 Custo = 503

Passo 4: Faça *M+* e *M-* serem a máquina mais carregada e menos carregada do indivíduo I_j, respectivamente.

M+ = 1 e *M-* = 2

Passo 5: *MoveSet* selecionado aleatoriamente = Or-Opt

Passo 6: Execute o algoritmo selecionado no passo 5 para *M+* e *M-*, gerando um novo indivíduo I_n.

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Depois de verificar que não existe nenhuma melhora na máquina mais carregada, utilizando o Or-opt, o mesmo procedimento será aplicado para a máquina menos carregada.

Máquina 2: 22-11-7-1-8-10-4-18-16-15

Verificou-se que existe uma troca de 11-7-2 entre as tarefas 4 e 18, dando um custo de 498. E, como o custo com essa troca diminui, ela é feita, e a seqüência fica da seguinte forma:

Máquina 2: 22-8-10-4-11-7-1-18-16-15

Após o Or-opt, o indivíduo ficou assim:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-8-10-4-11-7-1-18-16-15 Custo = 498

Passo 7: Como (1 + 5)<= 10, então é inserido I_n na população.

Indivíduo 6:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-8-10-4-11-7-1-18-16-15 Custo = 498

Passo 8: *iter* = 2 e vá para o **Passo 3**.

Passo 3: Escolhido aleatoriamente:

Indivíduo 3:

Máquina 1: 21-1-6-11-7-2-8-20-15-4-5-16-17-18 Custo = 530

Máquina 2: 22-13-10-3-9-12-19-14 Custo = 500

Passo 4: *M+* = 1; *M-* = 2

Passo 5: Algoritmo de melhoramento sorteado: 2-OPT*

Máquina 1: 21-1-6-11-7-2-8-20-15-4-5-16-17-18

Máquina 2: 22-13-10-3-9-12-19-14-

Passo 6: O único caso em que ocorre uma troca é o rompimento da seqüência da tarefa 8 e 20 na máquina 1 e, na máquina 2, entre a tarefa 9 e 12.

Máquina 1: 21-1-6-11-7-2-8-20-15-4-5-16-17-18

Máquina 2: 22-13-10-3-9-12-19-14

Com a reconexão, teremos as seguintes seqüências de tarefas nas máquinas.

Máquina 1': 21-1-6-11-7-2-8-12-19-14 Custo = 516

Máquina 2': 22-13-10-3-9-20-15-4-5-16-17-18 Custo = 523

Passo 7: Como (2+5) <= 10, então insira I_n na população.

Indivíduo 7:

Máquina 1: 21-1-6-11-7-2-8-12-19-14 Custo = 516

Máquina 2: 22-13-10-3-9-20-15-4-5-16-17-18 Custo = 523

Passo 8: Faça *iter* =3 e vá para o **Passo 3**.

Passo 3: Indivíduo selecionado:

Indivíduo 1:

Máquina 1: 21-2-7-1-6-11-19-8-20-15-4-5-18 Custo = 496

Máquina 2: 22-12-13-10-3-9-16-17-14 Custo = 522

Passo 4: *M+* = 2 e *M-*=1

Passo 5: Algoritmo de melhoramento: Or-Opt

Passo 6: As seqüências testadas são 13-10-3 entre as tarefas 9-20, 20-15, 15-4, 4-5, 5-16, 16-17, 17-18 e 18-22. Depois, seleciona-se a seqüência 10-3-9 e testa-se entre 20-15, 15-4 e 5-16. O único caso que diminui o custo é inserir a seqüência 10-3-9 entre 5 e 16. Após essa inserção, é testada novamente a primeira seqüência de três tarefas.

Selecione a seqüência das três tarefas 10-3-9;

Máquina 2: 22-13-10-3-9-20-15-4-5-16-17-18

Máquina 2': 22-13-20-15-4-5-10-3-9-16-17-18

A nova troca é da seqüência 13-20-15 entre 5 e 10.

Máquina 2: 22-13-20-15-4-5-10-3-9-16-17-18

Máquina 2': 22-4-5-13-20-15-10-3-9-16-17-18

Após testar todas as possibilidades da seqüência de três tarefas, o mesmo é feito para duas tarefas, e a troca que pode ser feita com melhora é a seqüência 20-15, e esta vai ser inserida entre 4 e 5.

Máquina 2: 22-4-5-13-20-15-10-3-9-16-17-18

Máquina 2': 22-4-20-15-5-13-10-3-9-16-17-18

Após essa troca, inicia o processo de procura com a seqüência de três tarefas, e a troca encontrada é a seguinte:

Máquina 2: 22-4-20-15-5-13-10-3-9-16-17-18

Máquina 2': 22-4-5-13-10-3-9-16-17-18-20-15

Outra troca possível é seguinte:

Máquina 2: 22-4-5-13-10-3-9-16-17-18-20-15

Máquina 2': 22-4-5-3-9-16-17-13-10-18-20-15

E, na máquina menos carregada, a troca possível é:

Máquina 1: 21-1-6-11-7-2-8-12-19-14

Máquina 1': 21-1-6-11-7-19-14-2-8-12

E a outra troca é:

Máquina 1: 21-1-6-11-7-19-14-2-8-12

Máquina 1': 21-1-7-19-6-11-14-2-8-12

Passo 7: Se (*iter* + *pop*) <= *popmax,* então insira I_n na população. Caso contrário, se I_n é melhor que o pior indivíduo na população, então troque pelo I_n.

Indivíduo 8:

Máquina 1:21-1-7-19-6-11-14-2-8-12 Custo = 512

Máquina 2: 22-4-5-3-9-16-17-13-10-18-20-15 Custo = 508

Passo 8: Faça iter = 4 e vá para o Passo 3.

Passo 3: Escolhido aleatoriamente:

Indivíduo 3:

Máquina 1: 21-1-6-11-7-2-8-20-15-4-5-16-17-18 Custo = 530

Máquina 2: 22-13-10-3-9-12-19-14 Custo = 500

Passo 4: *M+* = 1 e *M-*= 2

Passo 5: Algoritmo de melhoramento selecionado: *two-interchange.*

Passo 6: Indivíduo 3 antes do *two-interchange*:

Máquina 1: 21-1-6-11-7-2-8-20-15-4-5-16-17-18 Custo = 530

Máquina 2: 22-13-10-3-9-12-19-14 Custo = 500

As tarefas escolhidas foram a 20 da máquina 1 e a 3 da máquina 2. Depois, foi feita a inserção mais barata da tarefa 3 na máquina 1 e da tarefa 20 na máquina 2.

Logo após o *MoveSet*, o indivíduo ficou assim:

Máquina 1: 21-1-6-11-7-2-8-15-4-5-16-17-3-8 Custo = 574

Máquina 2: 22-13-10-9-12-20-19-14 Custo = 461

Passo 7: Se (4 + 5) <= 15, então insira I_n na população. Caso contrário, se I_n é melhor que o pior indivíduo na população, então troque pelo I_n.

Indivíduo 9:

Máquina 1: 21-1-6-11-7-2-8-15-4-5-16-17-3-8 Custo = 574

Máquina 2: 22-13-10-9-12-20-19-14 Custo = 461

Passo 8: Faça *iter* = *5* e vá para o **Passo 3**.

Passo 3: Escolhido aleatoriamente:

Indivíduo 6:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo: 512

Máquina 2: 22-8-10-4-11-7-1-18-16-15 Custo: 498

Passo 4: *M+* = 1 e *M-* =2.

Passo 5: *MoveSet* = Or-Opt.

Passo 6: *M+* não tem troca, e *M-* tem a seguinte:

Máquina 2: 22-8-10-4-11-7-1-18-16-15

Máquina 2': 22-8-10-1-18-16-15-4-11-7

I_n → Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo: 512

Máquina 2: 22-8-10-1-18-16-15-4-11-7 Custo: 497

Passo 7: Se (5+5) <= 10, então:

Indivíduo 10:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-8-10-1-18-16-15-4-11-7 Custo = 497

Passo 8: Faça *iter* = 6 e vá para o **Passo 3**.

Passo 3: Escolhido aleatoriamente:

Indivíduo 3:

Máquina 1: 21-1-6-11-7-2-8-20-15-4-5-16-17-18 Custo = 530

Máquina 2: 22-13-10-3-9-12-19-14 Custo = 500

Passo 4: *M+* = 1 e *M-* =2.

Passo 5: *MoveSet* = Or-Opt.

Passo 6: *M+* e *M-* têm trocas, que são as seguintes:

Máquina 1: 21-1-6-11-7-2-8-20-15-4-5-16-17-18

Máquina 1' : 21-1-6-11-7-2-8-20-15-4-18-5-16-17

Máquina 2: 22-13-10-3-9-12-19-14

Máquina 2' : 22-9-12-13-10-3-19-14

Máquina 2: 22-9-12-13-10-3-19-14

Máquina 2' : 22-3-19-9-12-13-10-14

In → Máquina 1: 21-1-6-11-7-2-8-20-15-4-18-5-16-17 Custo = 525

Máquina 2: 22-3-19-9-12-13-10-14 Custo = 491

Passo 7: Se (*6+5)* <= *10*, então insira I_n na população. Caso contrário, se I_n é melhor que o pior indivíduo na população, então troque pelo I_n.

Será trocado o indivíduo 9 pelo I_n.

Indivíduo 9:

Máquina 1: 21-1-6-11-7-2-8-20-15-4-18-5-16-17 Custo = 525

Máquina 2: 22-3-19-9-12-13-10-14 Custo = 491

Passo 8: Faça *iter* = 7 e vá para o **Passo 3**.

Passo 3: Escolhido aleatoriamente:

Indivíduo 2:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-11-7-1-8-10-4-18-16-15 Custo = 503

Passo 4: *M+* = 1 e *M-* =2.

Passo 5: *MoveSet* = Or-Opt.

Passo 6: *M+* não tem troca, e *M-* tem troca, que é a seguinte:

Máquina 2: 22-11-7-1-8-10-4-18-16-15

Máquina 2' 22-8-10-4-11-7-1-18-16-15

I_n → *Máquina 1:* 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-8-10-4-11-7-1-18-16-15 Custo = 498

Passo 7: Se (*7+5) <= 10*, então insira I_n na população. Caso contrário, se I_n é melhor que o pior indivíduo na população, então troque pelo I_n.

Será trocado o indivíduo 4 pelo I_n.

Indivíduo 4:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-8-10-4-11-7-1-18-16-15 Custo = 498

Passo 8: Faça *iter* = 8 e vá para o **Passo 3**.

Passo 3: Escolhido aleatoriamente:

Indivíduo 4:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-8-10-4-11-7-1-18-16-15 Custo = 498

Passo 4: *M+* = 1 e *M-* =2.

Passo 5: *MoveSet = two-interchange*.

Passo 6: *M+* e *M-* têm trocas, que são as seguintes:

Máquina 1: 21-19-6-18-5-3-9-2-12-14-17-20 Custo = 506

Máquina 2: 22-8-10-13-4-11-7-1-16-15 Custo = 511

I_n → *Máquina 1*: 21-19-6-18-5-3-9-2-12-14-17-20 Custo = 506

Máquina 2: 22-8-10-13-4-11-7-1-16-15 Custo = 511

Passo 7: Se (*8+5) <= 10*, então insira I_n na população. Caso contrário, se I_n é melhor que o pior indivíduo na população, então troque pelo I_n.

Será trocado o indivíduo 3 pelo I_n.

Indivíduo 3:

Máquina 1: 21-19-6-18-5-3-9-2-12-14-17-20 Custo = 506

Máquina 2: 22-8-10-13-4-11-7-1-16-15 Custo = 511

Passo 8: Faça iter = 9 e vá para o Passo 3.

Passo 3: Escolhido aleatoriamente:

Indivíduo 9:

Máquina 1: 21-1-6-11-7-2-8-20-15-4-18-5-16-17 Custo = 525

Máquina 2: 22-3-19-9-12-13-10-14 Custo = 491

Passo 4: *M+* = 1 e *M-* =2.

Passo 5: *MoveSet* = Or-opt.

Passo 6: *M+* e *M-* não têm trocas.

I_n → *Máquina 1:* 21-1-6-11-7-2-8-20-15-4-18-5-16-17 Custo = 525

Máquina 2: 22-3-19-9-12-13-10-14 Custo = 491

Passo 7: Se (*9+5) <= 10*, então insira I_n na população. Caso contrário, se I_n é melhor que o pior indivíduo na população, então troque pelo I_n.

Não será trocado o indivíduo I_n pelo pior, porque não existe indivíduo pior que I_n.

Passo 8: Faça iter = 10 e vá para o Passo 3.

Passo 3: Escolhido aleatoriamente:

Indivíduo 7:

Máquina 1: 21-1-6-11-7-2-8-12-19-14 Custo = 516

Máquina 2: 22-13-10-3-9-20-15-4-5-16-17-18 Custo = 523

Passo 4: *M+* = 2 e *M-* = 1.

Passo 5: *MoveSet = two-interchange*.

Passo 6: *M+* e *M-* têm trocas, que são as seguintes:

Máquina 1: 21-1-6-11-7-2-8-12-19-14 Custo = 516

Máquina 1': 21-1-6-11-7-16-8-12-19-14 Custo = 583

Máquina 2: 22-13-10-3-9-20-15-4-5-16-17-18 Custo = 523

Máquina 2': 22-13-10-3-9-2-20-15-4-5-17-18 Custo = 453

I_n → *Máquina 1*: 21-1-6-11-7-16-8-12-19-14 Custo = 583

Máquina 2: 22-13-10-3-9-2-20-15-4-5-17-18 Custo = 453

Passo 7: Se (*10+5) <= 10*, então insira I_n na população. Caso contrário, se I_n é melhor que o pior indivíduo na população, então troque pelo I_n.

Não será trocado o indivíduo I_n pelo pior, porque não existe indivíduo pior que I_n.

Passo 8: Faça iter = 11 e vá para o Passo 3.

Passo 3: Escolhido aleatoriamente:

Indivíduo 2:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-11-7-1-8-10-4-18-16-15 Custo = 503

Passo 4: *M+* = 1 e *M-* =2.

Passo 5: *MoveSet* = Or-opt.

Passo 6: *M+* não tem troca, e *M-* tem troca, que é a seguinte:

Máquina 2: 22-11-7-1-8-10-4-18-16-15

Máquina 2': 22-8-10-4-11-7-1-18-16-15

I_n → *Máquina 1*: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-8-10-4-11-7-1-18-16-15 Custo = 498

Passo 7: Se *(11+5) <= 10*, então insira I_n na população. Caso contrário, se I_n é melhor que o pior indivíduo na população, então troque pelo I_n.

Será trocado o indivíduo 9 pelo I_n.

Indivíduo 9:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-8-10-4-11-7-1-18-16-15 Custo = 498

Passo 8: Faça iter = 12 e vá para o Passo 3.

Passo 3: Escolhido aleatoriamente:

Indivíduo 4:

Máquina 1: 21-2-7-1-6-11-19-8-20-15-4-5-18 Custo = 496

Máquina 2: 22-12-13-10-3-9-16-17-14 Custo = 522

Passo 4: *M+* = 2 e *M-* = 1.

Passo 5: *MoveSet* = Or-opt.

Passo 6: *M+* e *M-* têm trocas, que são as seguintes:

Máquina 2: 22-12-13-10-3-9-16-17-14

Máquina 2': 22-3-12-13-10-9-16-17-14

Máquina 1: 21-2-7-1-6-11-19-8-20-15-4-5-18

Máquina 1' : 21-2-7-1-6-11-19-8-20-15-5-4-18

Máquina 1: 21-2-7-1-6-11-19-8-20-15-5-4-18

Máquina 1' 21-2-7-1-6-11-19-8-20-15-4-18-5

I_n → *Máquina 1*: 21-2-7-1-6-11-19-8-20-15-4-18-5 Custo = 494

Máquina 2: 22-3-12-13-10-9-16-17-14 Custo = 518

Passo 7: Se (*11+5) <= 10*, então insira I_n na população. Caso contrário, se I_n é melhor que o pior indivíduo na população, então troque pelo I_n.

Será trocado o indivíduo 5 pelo I_n.

Indivíduo 5:

Máquina 1: 21-2-7-1-6-11-19-8-20-15-4-18-5 Custo = 494

Máquina 2: 22-3-12-13-10-9-16-17-14 Custo = 518

Passo 8: Faça iter = 13 e vá para o Passo 3.

Passo 3: Escolhido aleatoriamente::

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-8-10-4-11-7-1-18-16-15 Custo = 498

Passo 4: *M+* = 1 e *M-* =2.

Passo 5: *MoveSet* = 2-opt*

Passo 6: *M+* e *M-* têm trocas, que são as seguintes:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20

Máquina 2: 22-8-10-4-11-7-1-18-16-15

Máquina 1' 21-19-6-5-3-9-2-12-13-14-17-15

Máquina 2' 22-8-10-4-11-7-1-18-16-20

I_n → *Máquina 1*: 21-19-6-5-3-9-2-12-13-14-17-15 Custo = 508

Máquina 2: 22-8-10-4-11-7-1-18-16-20 Custo = 509

Passo 7: Se (*12+5) <= 10*, então insira I_n na população. Caso contrário, se I_n é melhor que o pior indivíduo na população, então troque pelo I_n.

Será trocado o indivíduo 1 pelo I_n.

Indivíduo 1:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-15 Custo = 508

Máquina 2: 22-8-10-4-11-7-1-18-16-20 Custo = 509

Passo 8: Faça iter = 14 e vá para o Passo 3.

Passo 3: Indivíduo 1 escolhido:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-15 Custo = 508

Máquina 2: 22-8-10-4-11-7-1-18-16-20 Custo = 509

Passo 4: *M+* = 2 e *M-* =1.

Passo 5: *MoveSet* = Or-opt

Passo 6: *M+* e *M-* têm trocas, que são as seguintes:

Máquina 2: 22-8-10-4-11-7-1-18-16-20

*Máquina 2' 2*2-8-10-4-18-16-11-7-1-20

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-15

*Máquina 1' 2*1-3-9-2-12-13-14-17-19-6-5-15

I_n → *Máquina 1*: 21-3-9-2-12-13-14-17-19-6-5-15 Custo = 507

Máquina 2: 22-8-10-4-18-16-11-7-1-20 Custo = 507

Passo 7: Se (*13+5) <= 10*, então insira I_n na população. Caso contrário, se I_n é melhor que o pior indivíduo na população, então troque pelo I_n.

Será trocado o indivíduo 5 pelo I_n.

Indivíduo 5:

Máquina 1: 21-3-9-2-12-13-14-17-19-6-5-15 Custo = 507

Máquina 2: 22-8-10-4-18-16-11-7-1-20 Custo = 507

Passo 8: Faça iter = 15 e vá para o Passo 3.

Passo 3: Indivíduo 1 escolhido:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-15 Custo = 508

Máquina 2: 22-8-10-4-11-7-1-18-16-20 Custo = 509

Passo 4: *M+* = 2 e *M-* = 1.

Passo 5: *MoveSet* = Or-opt.

Passo 6: *M+* e *M-* têm trocas, que são as seguintes:

Máquina 2: 22-8-10-4-11-7-1-18-16-20

Máquina 2': 22-8-10-4-18-16-11-7-1-20

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-15

*Máquina 1' 2*1-3-9-2-12-13-14-17-19-6-5-15

I_n → *Máquina 1*: 21-3-9-2-12-13-14-17-19-6-5-15 Custo = 507

Máquina 2: 22-8-10-4-18-16-11-7-1-20 Custo = 507

Passo 7: Se (*15+5) <= 10*, então insira I_n na população. Caso contrário, se I_n é melhor que o pior indivíduo na população, então troque pelo I_n.

Será trocado o indivíduo 7 pelo I_n.

Indivíduo 7:

Máquina 1: 21-3-9-2-12-13-14-17-19-6-5-15 Custo = 507

Máquina 2: 22-8-10-4-18-16-11-7-1-20 Custo = 507

Passo 8: Faça iter = 15 e vá para o Passo 3.

E assim funciona o algoritmo até o passo *iter* = 15. A população, após as 15 iterações, é a seguinte:

Indivíduo 1:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-15 Custo = 508

Máquina 2: 22-8-10-4-11-7-1-18-16-20 Custo = 509

Indivíduo 2

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-11-7-1-8-10-4-18-16-15 Custo = 503

Indivíduo 3:

Máquina 1: 21-19-6-18-5-3-9-2-12-14-17-20 Custo = 506

Máquina 2: 22-8-10-13-4-11-7-1-16-15 Custo = 511

Indivíduo 4:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-8-10-4-11-7-1-18-16-15 Custo = 498

Indivíduo 5:

Máquina 1: 21-3-9-2-12-13-14-17-19-6-5-15 Custo = 507

Máquina 2: 22-8-10-4-18-16-11-7-1-20 Custo = 507

Indivíduo 6:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-8-10-4-11-7-1-18-16-15 Custo = 498

Indivíduo 7:

Máquina 1: 21-3-9-2-12-13-14-17-19-6-5-15 Custo = 507

Máquina 2: 22-8-10-4-18-16-11-7-1-20 Custo = 507

Indivíduo 8:

Máquina 1: 21-1-7-19-6-11-14-2-8-12 Custo = 512

Máquina 2: 22-4-5-3-9-16-17-13-10-18-20-15 Custo = 508

Indivíduo 9:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-8-10-4-11-7-1-18-16-15 Custo = 498

Indivíduo 10:

Máquina 1: 21-19-6-5-3-9-2-12-13-14-17-20 Custo = 512

Máquina 2: 22-8-10-1-18-16-15-4-11-7 Custo = 497

O *makespan* para esta resolução é **507**.

3.2.3 ESched2

Antes de descrever ***ESched2*** em detalhes, apresentamos algumas definições necessárias. O operador de recombinação que será usado no Passo 5 é o conhecido *Order-Based Crossover* que mostra resultados melhores que o *Cycle Crossover* (Glodberg, 1989).

O código de mutação necessita de uma lista de tarefas indexadas com tamanho 2*n*, onde *n* é o número total de tarefas. Cada tarefa indexada *i*, $1 \leq i \leq n$, aparece apenas duas vezes e em diferentes locais na lista mencionada.

Com o operador e o código C de mutação, um novo indivíduo é gerado com a lista, removendo a tarefa J_i de I_A na primeira aparência de *i* e reinserindo este quando aparece na segunda vez. Após a retirada da tarefa, é feita uma busca local com o Or-opt modificado. E, no momento de inserção da tarefa, é utilizada a inserção mais barata na máquina que resultar menor acréscimo no custo final.

O Or-opt modificado primeiro identifica a tarefa a ser removida, dita J_K, e seu predecessor e sucessor, ditos J_i e J_j, fazendo uma busca local na tarefa *permanecer alocado na máquina*.

A recombinação utilizada é *Order-Based Crossover*, a qual necessita de uma lista aleatória de zeros e uns, que definirá se a tarefa permanece ou não na posição do primeiro código de mutação. O código de mutação e a lista de zeros e uns têm o mesmo tamanho.

O código de mutação inicial é dado pela seqüência de alocação das tarefas nas máquinas, começando pela primeira máquina até a *m*-ésima máquina.

Para entendimento do método, considere C_1 e L_1, onde C_1 é o código de mutação, e L_1, a lista aleatória de zeros e uns.

C_1: 9-2-7-16-3-12-13-14-20-1-6-11-10-8-4-5-15-17-19-18-9-2-7-16-3-12-13-14-20-1-6-11-10-8-4-5-15-17-19-18

L_1: 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1

O método funciona da seguinte forma do exemplo em questão: no código de mutação C_1, temos, por primeiro, a tarefa 9 e, na mesma posição em L_1, tem-se o valor 1. Isso representa que a 9 irá continuar na primeira posição em C'_3. Depois, temos a tarefa 2, que também permanecerá na mesma localização. Na tarefa 7, também acontece a mesma coisa. Na tarefa 16, temos, na mesma posição em L_1, o valor 0, que implica esta tarefa ficar em uma lista provisória das tarefas não alocadas. Este procedimento continua até o fim da lista.

C'_3: 9-2-7- - -12- - -20- -6- -10-8-4-5-15-17-19-18- -2-7- -3-12-13- - -

-6- - - - -5- - -19-18

Tarefas não alocadas:16,3,13,14,1, 11, 9, 16, 14, 20, 1, 11, 10, 8, 4, 15, 17.

Após identificar a ordem e as tarefas não alocadas, utiliza-se C_2 para definir a ordem das tarefas não alocadas.

C_2: 3-9-2-7-1-6-11-19-8-20-15-17-14-4-5-12-13-10-18-16-3-9-2-7-1-6-11-19-8-20-15-17-14-4-5-12-13-10-18-16

Ordem que as tarefas serão alocadas: 3, 9, 1, 11, 8, 20, 15, 17, 14, 4, 13, 10, 16, 1, 11, 14, 16.

Será preenchido o código de mutação C'_3 de acordo com a ordem das tarefas não alocadas.

C'_3= 9-2-7- 3 - 9 -12-1-11-20-8-6-20-10-8-4-5-15-17-19-18-15 -2-7-17-3-12-13-14-4-13-6-10-16-1-11-5-14-16-19-18

Para entendimento do Or-Opt modificado para o ***ESched2***, utilizamos o seguinte exemplo:

Máquina 1: 21-12-13-10-3-15-4-5-18-19 Custo = 541

Máquina 2: 22-20-1-6-11-7-2-8-9-16-17-14 Custo = 490

C'$_3$=9-2-7-3-9-12-1-11-20-8-6-20-10-8-4-5-15-17-19-18-15-2-7-17-3-12-13-14-4-13-6-10-16-1-11-5-14-16-19-18

Or-Opt modificado para o ***Esched2*** funciona da seguinte forma: como a primeira tarefa do código de mutação é a 9, esta é retirada da máquina 2. Neste momento, tem-se uma lacuna entre a tarefa 8 e 16, e essa lacuna é a modificação do Or-opt. Ao invés de o Or-opt testar todas as possibilidades, ele fixará essa lacuna e tentará alocar uma seqüência de três ou duas ou uma tarefa entre a tarefa 8 e 16. Neste Or-Opt, é permitida a inversão da seqüência das tarefas.

Antes da remoção da tarefa 9, temos: Máquina 2: 22-20-1-6-11-7-2-8-9-16-17-14-22. A primeira troca possível é inverter o sentindo das tarefas após a lacuna, e este conjunto de três tarefas será alocado entre o indicador de inicialização da máquina e a tarefa 20. Após a remoção da tarefa 9 e a aplicação do Or-opt modificado, temos a seguinte alocação de tarefas na máquina:

Máquina' : 22-14-17-16-20-1-6-11-7-2-8-22

Após estas definições prévias, podemos apresentar formalmente a segunda estratégia evolutiva ***ESched2***:

A descrição através de fluxograma de ***ESched2*** pode ser feita da seguinte forma:

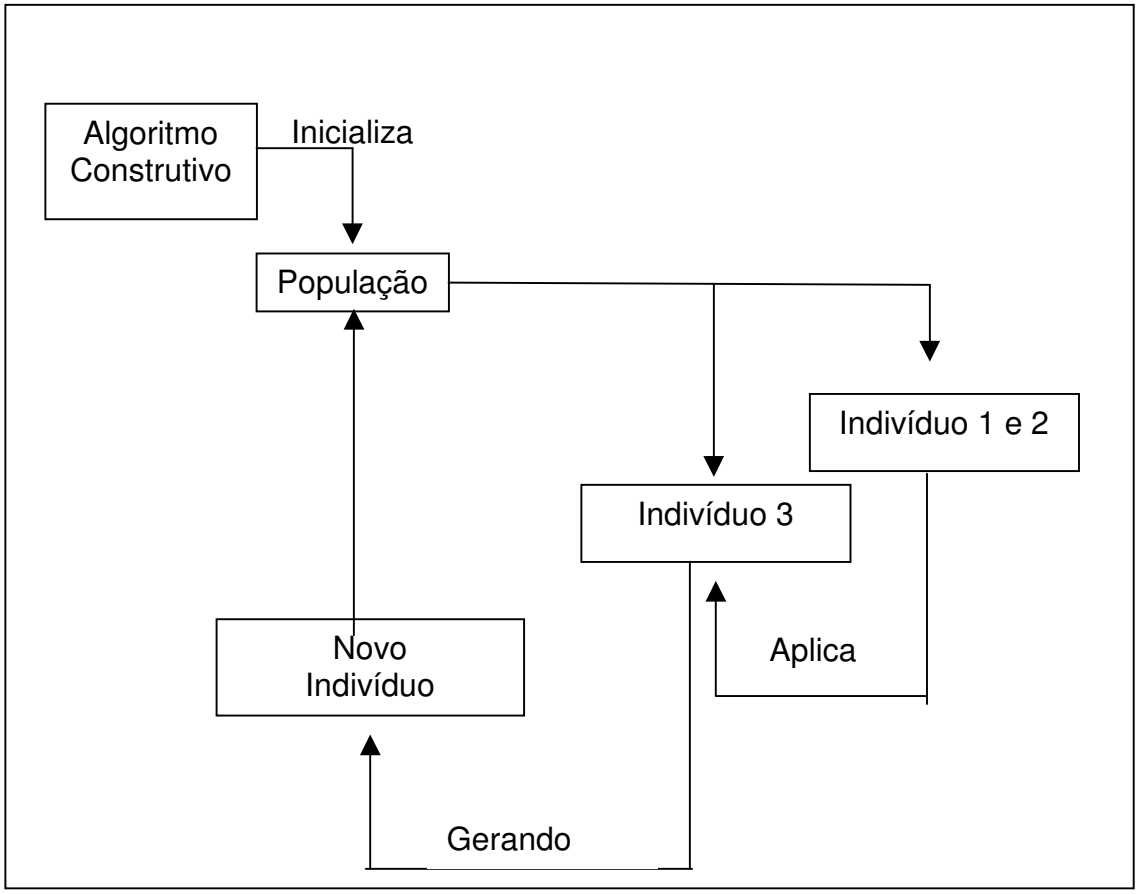

Figura 5 – Fluxograma de *ESched2*.

A descrição da ***ESched2*** passo a passo pode ser feita da seguinte forma:

Passos 1 e 2: Idênticos ao ***ESched1***.

Passo 3: Gere o código de mutação para cada indivíduo da população.

Passo 4: Se *iter* > *imax*, então PARE. Caso contrário, escolha, aleatoriamente, dois diferentes indivíduos, ditos I_1 e I_2. Considere C_1 e C_2 o código de mutação de I_1 e I_2, respectivamente.

Passo 5: Gere uma lista aleatória de zeros e uns, L_1 e um novo código de mutação C'_3 de um indivíduo resultante da recombinação de C_1 e C_2.

Passo 6: Escolha, aleatoriamente, da população, um indivíduo I_3 diferente de I_1 e I_2.

Passo 7: Realize a mutação em I_3 utilizando o código de mutação C'_3 resultando um novo indivíduo I'_3.

Passo 8: Se *iter* ≤ *popmax,* então insira I'_3 na população. Caso contrário, se I'_3 é melhor que o pior indivíduo na população, então troque pelo I'_3.

Passo 9: Faça *iter* = *iter* + 1 e vá para o **Passo 4**.

Será apresentado um exemplo para melhor entendimento do método proposto.

Passos 1 e 2: Idênticos ao ***ESched1***.

Então, gera-se uma população de *pop* = 5; *popmax* = 10; *imax* = 15;

Indivíduo 1:

Máquina 1: 21-3-9-2-7-1-6-11-13-14-16 Custo = 514

Máquina 2: 22-8-10-4-5-12-20-15-17-19-18 Custo = 513

Indivíduo 2:

Máquina 1: 21-9-2-7-16-3-12-13-14 Custo = 487

Máquina 2: 22-20-1-6-11-10-8-4-5-15-17-19-18 Custo = 544

Indivíduo 3:

Máquina 1: 21-3-9-2-7-1-6-11-19-8-20-15-17-14 Custo = 525

Máquina 2: 22-4-5-12-13-10-18-16 Custo = 488

Indivíduo 4:

Máquina 1: 21-12-13-10-3-15-4-5-18-19 Custo = 541

Máquina 2: 22-20-1-6-11-7-2-8-9-16-17-14 Custo = 490

Indivíduo 5:

Máquina 1: 21-6-5-3-9-2-12-13-14-17-20-15-18 Custo = 515

Máquina 2: 22-11-7-1-8-10-4-19-16 Custo = 510

Passo 3: Gere o código de mutação para cada indivíduo da população.

Indivíduo 1:

C_1: 3-9-2-7-1-6-11-13-14-16-8-10-4-5-12-20-15-17-19-18-3-9-2-7-1-6-11-13-14-16-8-10-4-5-12-20-15-17-19-18

Indivíduo 2:

C_2: 9-2-7-16-3-12-13-14-20-1-6-11-10-8-4-5-15-17-19-18-9-2-7-16-3-12-13-14-20-1-6-11-10-8-4-5-15-17-19-18

Indivíduo 3:

C_3: 3-9-2-7-1-6-11-19-8-20-15-17-14-4-5-12-13-10-18-16-3-9-2-7-1-6-11-19-8-20-15-17-14-4-5-12-13-10-18-16

Indivíduo 4:

C_4: 12-13-10-3-15-4-5-18-19-20-1-6-11-7-2-8-9-16-17-14-12-13-10-3-15-4-5-18-19-20-1-6-11-7-2-8-9-16-17-14

Indivíduo 5:

C_5: 6-5-3-9-2-12-13-14-17-20-15-18-11-7-1-8-10-4-19-16-6-5-3-9-2-12-13-14-17-20-15-18-11-7-1-8-10-4-19-16

Passo 4: Se *iter* > *imax*, então PARE. Caso contrário, escolha, aleatoriamente, dois diferentes indivíduos, ditos I_1 e I_2. Considere C_1 e C_2 o código de mutação de I_1 e I_2, respectivamente.

Indivíduos 2 e 3 foram os indivíduos selecionados aleatoriamente.

I_1 → *Máquina 1:* 21-9-2-7-16-3-12-13-14 Custo = 487

Máquina 2: 22-20-1-6-11-10-8-4-5-15-17-19-18 Custo = 544

C_1: 9-2-7-16-3-12-13-14-20-1-6-11-10-8-4-5-15-17-19-18-9-2-7-16-3-12-13-14-20-1-6-11-10-8-4-5-15-17-19-18

I_2 → Máquina 1: 21-3-9-2-7-1-6-11-19-8-20-15-17-14 Custo = 525

Máquina 2: 22-4-5-12-13-10-18-16 Custo = 488

C_2: 3-9-2-7-1-6-11-19-8-20-15-17-14-4-5-12-13-10-18-16-3-9-2-7-1-6-11-19-8-20-15-17-14-4-5-12-13-10-18-16

Passo 5: Gere uma lista aleatória de zeros e uns, L_1 e um novo código de mutação C'_3 de um indivíduo resultante da recombinação de C_1 e C_2.

C_1: 9-2-7-16-3-12-13-14-20-1-6-11-10-8-4-5-15-17-19-18-9-2-7-16-3-12-13-14-20-1-6-11-10-8-4-5-15-17-19-18

L_1: 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1

C_2: 3-9-2-7-1-6-11-19-8-20-15-17-14-4-5-12-13-10-18-16-3-9-2-7-1-6-11-19-8-20-15-17-14-4-5-12-13-10-18-16

C'_3= 9-2-7- 3 - 9 -12-1-11-20-8-6-20-10-8-4-5-15-17-19-18-15 -2-7-17-3-12-13-14-4-13-6-10-16-1-11-5-14-16-19-18

Passo 6: Escolha, aleatoriamente, da população, um indivíduo I_3 diferente de I_1 e I_2.

Indivíduo selecionado: 4

Máquina 1: 21-12-13-10-3-15-4-5-18-19 Custo = 541

Máquina 2: 22-20-1-6-11-7-2-8-9-16-17-14 Custo = 490

Passo 7: Realize a mutação em I_3 utilizando o código de mutação C'_3, resultando um novo indivíduo I'_3.

Máquina 1: 21-12-13-10-3-15-4-5-18-19 Custo = 541

Máquina 2: 22-20-1-6-11-7-2-8-9-16-17-14 Custo = 490

C'$_3$=9-2-7-3-9-12-1-11-20-8-6-20-10-8-4-5-15-17-19-18-15-2-7-17-3-12-13-14-4-13-6-10-16-1-11-5-14-16-19-18

remoção da tarefa 9 da Máquina 2: 22-20-1-6-11-7-2-8-9-16-17-14, após a remoção aplicação do Or-opt modificado.

Máquina' : 22-14-17-16-20-1-6-11-7-2-8

Depois da remoção da tarefa 9, é feita a remoção da tarefa 2, de acordo com o código de mutação:

Máquina: 22-14-17-16-20-1-6-11-7-2-8, após a remoção, é aplicado o Or-opt modificado:

Máquina' : 22-20-1-6-11-7-16-17-14-8

Este procedimento é repetido até que sejam feitas todas as retiradas das tarefas e, após, a inserção das tarefas de acordo com o código de mutação:

Remoção da tarefa 7 Máquina: 22-20-1-6-11-7-16-17-14-8

Máquina' : 22-8-14-16-17-20-1-6-11

Remoção da tarefa 3 Máquina: 21-12-13-10-3-15-4-5-18-19

Máquina' : 21-19-15-4-5-18-10-13-12

Inserção da tarefa 9 22-8-14-9-16-17-20-1-6-11

Remova a tarefa 12 Máquina 1: 21-19-15-4-5-18-10-13-12

Máquina' : 21-19-15-4-10-13-18-5

Remoção da tarefa 1 Máquina 2: 22-8-14-9-16-17-20-1-6-11

Máquina' : 22-8-14-9-16-17-20-6-11

Remoção da tarefa 11 Máquina 2: 22-8-14-9-16-17-20-6-11

Máquina' : 22-8-14-9-16-17-20-6

Remoção da tarefa 20 Máquina 2: 22-8-14-9-16-17-20-6

Máquina' : 22-14-9-16-17-8-6

Remoção da tarefa 8 Máquina 2: 22-14-9-16-17-8-6

Máquina' : 22-14-9-16-17-6

Remoção da tarefa 6 Máquina 2: 22-14-9-16-17-6

Máquina' : 22-14-9-16-17

Inserção da tarefa 20 Máquina 2: 22-14-9-16-17-20

Remoção da tarefa 10 Máquina 1: 21-19-15-4-10-13-18-5

Máquina' : 21-19-15-4-5-13-18

Inserção da tarefa 8 Máquina 2: 22-14-9-16-17-8-20

Remoção da tarefa 4 Máquina 1: 21-19-15-4-5-13-18

Máquina' : 21-19-15-5-13-18

Remoção da tarefa 5 Máquina 1: 21-19-15-5-13-18

Máquina' : 21-19-15-13-18

Remoção da tarefa 15 Máquina 1: 21-19-15-13-18

Máquina' : 21-19-13-18

Remoção da tarefa 17 Máquina 2: 22-14-9-16-17-8-20

Máquina' : 22-14-9-16-8-20

Remoção da tarefa 19 Máquina 1: 21-19-13-18

Máquina' : 21-13-18

Remoção da tarefa 18 Máquina 1: 21-13-18

Máquina' : 21-13

Inserção da tarefa 15 Máquina 1: 21-13-15

Inserção da tarefa 2 Máquina 1: 21-2-13-15

Inserção da tarefa 7 Máquina 1: 21-2-7-13-15

Inserção da tarefa 17 Máquina 1: 21-2-7-13-15-17

Inserção da tarefa 3 Máquina 1: 21-3-2-7-13-15-17

Inserção da tarefa 12 Máquina 2: 21-3-2-7-12-13-15-17

Remoção da tarefa 13 Máquina 1: 21-3-2-7-12-13-15-17

Máquina' : 21-3-12-7-2-15-17

Remoção da tarefa 14 Máquina 2: 22-14-9-16-8-20

Máquina' : 22-9-16-8-20

Inserção da tarefa 4 Máquina 2: 22-4-9-16-8-20

Inserção da tarefa 13 Máquina 2: 22-4-9-16-8-20-13

Inserção da tarefa 6 Máquina 1: 21-3-12-7-2-6-15-17

Inserção da tarefa 10 Máquina 1: 21-3-12-7-2-6-15-10-17

Remoção da tarefa 16 Máquina 2: 22-4-9-16-8-20-13

Máquina' : 22-8-20-13-4-9

Inserção da tarefa 1 Máquina 2: 22-8-20-1-13-4-9

Inserção da tarefa 11 Máquina 2: 22-8-20-1-11-13-4-9

Inserção da tarefa 5 Máquina 2: 22-5-8-20-1-11-13-4-9

Inserção da tarefa 14 Máquina 1: 21-3-12-7-2-14-6-15-10-17

Inserção da tarefa 16 Máquina 2: 22-5-8-20-1-16-11-13-4-9

Inserção da tarefa 19 Máquina 1: 21-3-12-7-2-14-19-6-15-10-17

Inserção da tarefa 18 na máquina 2: 22-18-5-8-20-1-16-11-13-4-9

Passo 8: Se *iter* ≤ *popmax,* então insira I'_3 na população. Caso contrário, se I'_3 é melhor que o pior indivíduo na população, então troque pelo I'_3.

Indivíduo 6:

Máquina 1: 21-3-12-7-2-14-19-6-15-10-17 Custo = 504

Máquina 2: 22-18-5-8-20-1-16-11-13-4-9 Custo = 523

Este procedimento é repetido por mais 14 vezes e resulta na seguinte população final:

Indivíduo1:

Máquina 1: 21-3-9-2-7-1-6-11-13-14-16 Custo = 514

Máquina 2: 22-8-10-4-5-12-20-15-17-19-18 Custo = 513

Indivíduo 2:

Máquina 1: 21-3-9-7-14-6-11-16-5-15-20-18 Custo = 519

*Máquina 2:*22-2-1-17-19-8-10-13-4-12 Custo = 503

Indivíduo 3:

Máquina 1: 21-17-14-13-6-15-18-16-2-7-9-12 Custo = 521

Máquina 2: 22-1-10-8-20-4-11-19-5-3 Custo = 506

Indivíduo 4:

Máquina 1: 21-8-10-9-7-2-12-13-14-6-15 Custo = 503

22-4-18-20-1-11-19-16-5-3-17 Custo = 506

Indivíduo 5:

Máquina 1: 21-6-5-3-9-2-12-13-14-17-20-15-18 Custo = 515

Máquina 2: 22-11-7-1-8-10-4-19-16 Custo = 510

Indivíduo 6:

Máquina 1: 21-19-16-5-12-13-18-9-7-20 Custo = 517

Máquina 2: 22-2-1-8-4-11-10-14-17-3-6-15 Custo = 507

Indivíduo 7:

Máquina 1: 21-8-3-9-2-14-7-19-12-20-15 Custo = 491

Máquina 2: 22-4-11-13-6-18-5-16-10-1-17 Custo = 521

Indivíduo 8:

Máquina 1: 21-19-5-4-11-13-7-2-8-20-3 Custo = 506

Máquina 2: 22-1-17-14-18-10-9-16-6-15-12 Custo = 523

Indivíduo 9:

Máquina 1: 21-1-17-3-9-7-2-11-13-14-5-20 Custo = 504

Máquina 2: 22-19-16-8-10-4-18-12-6-15 Custo = 517

Indivíduo 10:

Máquina 1: 21-18-5-8-20-15-3-17-1-7-11-10 Custo = 516

Máquina 2: 22-14-6-13-4-19-9-16-2-12 Custo = 507

E o *makespan* após as 15 iterações foi de **506**.

3.2.4 ESched3

A última estratégia evolutiva proposta aqui será denominada ***ESched3***, e utilizará a variável *NMoves*. Essa variável é um número aleatório que diz quantas iterações devem ser executadas da busca local e parará nesse número de iterações ou quando não existirem mais movimentos de melhora. ***ESched3*** difere de ***Esched1*** apenas no passo 5, que pode ser escrito como segue:

Passo 5: Escolha aleatoriamente, do conjunto *MoveSet*, um algoritmo de melhoramento e execute por *NMoves* iterações ou enquanto não houver mais movimentos de melhora.

Utilizando o exemplo acima e resolvendo com ***Esched3,*** temos:

pop = 5; *popmax* = 10; *imax* = 15;

Passo 1:

Indivíduo 1:

Máquina 1: 21-2-7-1-8-20-15-4-5-12-13-18 Custo = 506

Máquina 2: 22-19-6-11-10-3-9-16-17-14 Custo = 506

Indivíduo 2:

Máquina 1: 21-3-9-2-7-1-6-15-12-13-14-17-18 Custo = 514

Máquina 2: 22-5-4-11-10-8-20-19-16 Custo = 496

Indivíduo 3:

Máquina 1: 21-8-10-1-6-11-13-14-17-20-18 Custo = 510

Máquina 2: 22-5-3-9-2-7-16-15-4-19-12 Custo = 502

Indivíduo 4:

Máquina 1: 21-10-1-8-20-15-4-5-3-9-16 Custo = 489

Máquina 2: 22-6-11-7-2-12-13-14-17-19-18 Custo = 540

Indivíduo 5:

Máquina 1: 21-18-5-3-9-2-7-16-8-20-15-4-19 Custo = 539

Máquina 2: 22-10-1-6-11-13-14-17-12 Custo = 475

Passo 2: *iter* =1.

Passo 3: Se *iter* > *imax*, então PARE. Caso contrário, escolha aleatoriamente um indivíduo I_j da população.

Indivíduo 2 selecionado:

Indivíduo 2:

Máquina 1: 21-3-9-2-7-1-6-15-12-13-14-17-18 Custo = 514

Máquina 2: 22-5-4-11-10-8-20-19-16 Custo = 496

Passo 4: *M+* = 1 *M-* = 2

Passo 5: Escolha, aleatoriamente, do conjunto *MoveSet*, um algoritmo de melhoramento e execute por *NMoves* iterações ou enquanto não houver mais movimentos de melhora.

MoveSet = 2-opt* e *NMoves* = 3.

Passo 6: Execute o algoritmo selecionado no passo 5 por *Nmoves* ou enquanto houver movimentos de troca, para *M+* e *M-*, gerando um novo indivíduo I_n.

Máquina 1: 21-3-9-2-7-1-6-15-12-13-14-17-18

Máquina 2: 22-5-4-11-10-8-20-19-16

Máquina 1' 21-3-9-4-11-10-8-20-19-16

Máquina 2' 22-5-2-7-1-6-15-12-13-14-17-18

Passo 7: Se (*pop + iter)* <= *popmax*, então insira I_n na população. Caso contrário, se I_n é melhor que o pior indivíduo na população, então troque pelo I_n.

Indivíduo 6:

Máquina 1: 21-3-9-4-11-10-8-20-19-16 Custo = 510

Máquina 2: 22-5-2-7-1-6-15-12-13-14-17-18 Custo = 509

Passo 8: *iter = 1* +1 e vá para o **Passo 3**.

Este procedimento se repete por mais 14 vezes, e a população final, após as 15 iterações, fica a seguinte:

Indivíduo 1:

Máquina 1: 21-2-7-1-8-20-15-4-5-12-13-18 Custo = 506

Máquina 2: 22-19-6-11-10-3-9-16-17-14 Custo = 506

Indivíduo 2:

Máquina 1: 21-2-7-1-8-20-15-4-5-12-13-18 Custo = 506

Máquina 2: 22-19-6-11-10-3-9-16-17-14 Custo = 506

Indivíduo 3:

Máquina 1: 21-2-7-1-8-20-15-4-5-12-13-18 Custo = 506

Máquina 2: 22-19-6-11-10-3-9-16-17-14 Custo = 506

Indivíduo 4:

Máquina 1: 21-2-7-1-8-20-15-4-5-12-13-18 Custo = 506

Máquina 2: 22-19-6-11-10-3-9-16-17-14 Custo = 506

Indivíduo 5:

Máquina 1: 21-2-7-1-8-20-15-4-5-12-13-18 Custo = 506

Máquina 2: 22-19-6-11-10-3-9-16-17-14 Custo = 506

Indivíduo 6:

Máquina 1: 21-2-7-1-8-20-15-4-5-12-13-18 Custo = 506

Máquina 2: 22-19-6-11-10-3-9-16-17-14 Custo = 506

Indivíduo 7:

Máquina 1: 21-8-10-1-6-11-13-14-17-20-18 Custo = 510

Máquina 2: 22-5-3-9-2-7-16-15-4-19-12 Custo = 502

Indivíduo 8:

Máquina 1: 21-18-10-8-20-1-6-11-13-14-17 Custo = 505

Máquina 2: 22-4-5-3-9-2-7-19-16-15-12 Custo = 497

Indivíduo 9:

Máquina 1: 21-8-10-1-6-11-13-14-17-20-18 Custo = 510

Máquina 2: 22-5-3-9-2-7-16-15-4-19-12 Custo = 502

Indivíduo 10:

Máquina 1: 21-2-7-1-8-20-15-4-5-12-13-18 Custo = 506

Máquina 2: 22-19-6-11-10-3-9-16-17-14 Custo = 506

Após as 15 iterações do ***ESched3***, temos, como *makespan* do exemplo, **505**.

Os parâmetros utilizados para o *Tabu Long*, *Tabu Fast* e Memético foram os mesmo utilizados nos artigo originais. Para *ESched1*, *ESched2* e *ESched3*, foi usado *pop* = 20, *popmax* = 50, *imax* = 400 e *NMoves* foi aleatoriamente gerado de acordo com a distribuição discreta uniforme no intervalo [1,10].

Para finalizar o capítulo, apresentar-se-á a solução de todas as metaheurísticas que serão utilizadas no próximo capítulo para comparações, para a instância u_01_0010_20_02_07.

A solução encontrada por ***ESched1*** foi:

Máquina 1: 21-4-11-10-1-7-19-16-5 Custo = 501

Máquina 2: 22-17-3-9-2-8-20-15-12-13-14-6-18-22 Custo = 503

A solução encontrada por ***ESched2*** foi:

Máquina 1: 21-3-12-20-11-13-18-16-5 Custo = 505

Máquina 2: 22-2-14-17-7-1-6-15-4-19-8-10-9 Custo = 507

A solução encontrada por ***ESched3*** foi:

Máquina 1: 21-4-19-16-5-3-9-7-1-8-20-15 Custo = 504

Máquina 2: 22-18-12-13-6-11-10-2-14-17-22 Custo = 504

A solução encontrada pelo *Tabu Fast* foi:

Máquina 1: 21-6-11-4-19-18-17-3-9-2-14-13 Custo = 523

Máquina 2: 22-20-15-1-7-16-10-5-12-8 Custo = 521

A solução encontrada pelo *Tabu Long* foi:

Máquina 1: 21-18-5-4-7-11-13-14-2-8 Custo = 511

Máquina 2: 22-19-6-9-16-10-3-15-1-17-12-20 Custo = 512

A solução encontrada pelo Memético:

Máquina 1: 21-2-14-17-7-1-6-11-10-8-20-15-4 Custo = 500

Máquina 2: 22-19-16-5-3-9-12-13-18-22 Custo = 506

Solução ótima encontrada pelo Algoritmo Exato:

Máquina 1: 21-18-5-4-19-16-3-9-2-7-1-17 Custo = 501

Máquina 2: 22-8-20-15-12-13-14-6-11-10 Custo = 501

Após a apresentação dos resultados para a instância não estruturada que foi dada no exemplo u_01_0010_20_02_07, segue-se para o próximo capítulo, no qual apresentaremos todos os resultados e a discussão deles.

4 RESULTADOS

Os algoritmos desenvolvidos nesta dissertação foram implementados em JAVA e executadas em uma máquina Pentium III, 1GHz – 512MB, com um sistema operacional Linux (Mandrake 9.0).

4.1 Instâncias

As instâncias testadas têm o mesmo formato das utilizadas por Müller et al. (2002). Elas foram geradas com combinações de n = 20, 40, 60 ,80 com m = 2, 4, 6, 8. O tempo de processamento foi gerado utilizando a distribuição uniforme entre (1,100). Os tempos de preparação das máquinas (limpeza das máquinas) são divididos em duas categorias: instâncias pequenas com valores entre [0, 10] e instâncias grandes com valores entre [0,100], ambos com distribuição uniforme. Os tempos de preparação das máquinas também foram gerados com duas possibilidades: estruturadas e não estruturadas. As estruturadas têm o tempo de preparação respeitando a desigualdade triangular, que é $s_{ij} \leq s_{ik} + s_{kj}, \forall\ i, j, k \neq i, j$. As instâncias não estruturadas não seguem a desigualdade triangular, o que, geralmente, torna os problemas mais difíceis de serem resolvidos. Foram gerados dez problemas para cada uma das combinações de *m* e *n*.

A nomenclatura utilizada nas instâncias estruturadas foi feita da seguinte forma, de acordo com o exemplo s_01_0100_60_04_05:

onde:

s significa a que instância pertence à classe dos problemas estruturados;

_01_0100 significa que os tempos de preparação das tarefas estejam entre [0,100];

60 é o número de tarefas;

04 é o número de máquinas;

05 representa qual instância está sendo trabalhada dentre as dez geradas.

A nomenclatura das instâncias estruturadas é ***s*** (*structed*) e das não estruturadas é *u* (*unstructed*), como pode ser visto nas Figuras 1 e 2.

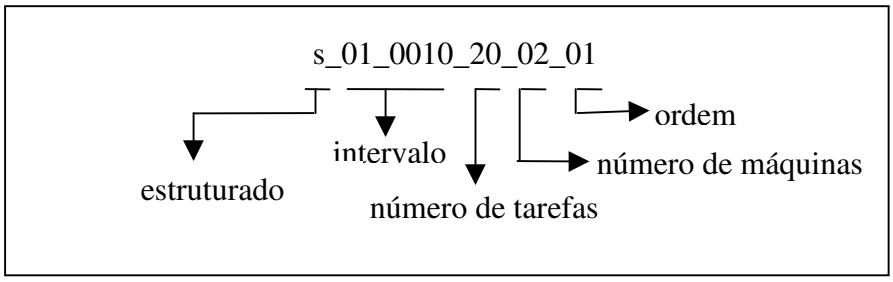

Figura 6 – Formato das instâncias estruturadas.

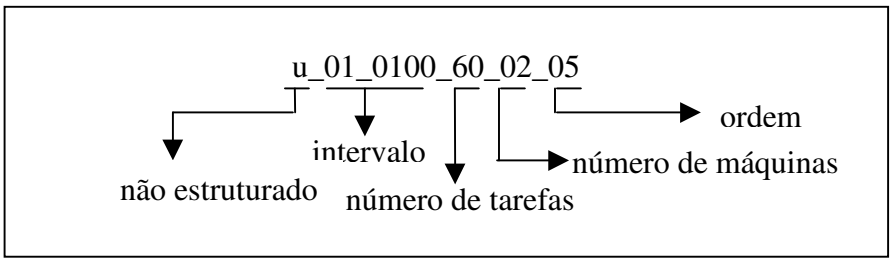

Figura 7 – Formato das instâncias não estruturadas.

4.2 **Resultados**

No presente trabalho, foram geradas um total de 640 instâncias. Desenvolveram-se três estratégias evolutivas para o P|tpds|Cmax. Os resultados obtidos foram comparados com as metaheurísticas desenvolvidas por (Müller, 1996) e (Mendes, 2002). Também foram comparados com o método exato proposto por Müller (1993). Para haver justiça nas comparações entre as metaheurísticas, testaram-se todas na mesma máquina e implementaram-se todas com a mesma linguagem de programação.

A escolha dos métodos para comparar com as EEs se deu devido a dois fatores importantes: a qualidade de solução obtida pelo Long Tabu e o esforço computacional despendido pelo Algoritmo Memético. Nesse sentido, foi mostrado que *ESched1*, *ESched2* e *ESched3* superam, na média, o desempenho dos outros dois métodos.

Os resultados apresentados nas Tabelas 2 a 5 mostram a média do desvio percentual em relação ao ótimo. Quando o número de instâncias resolvidas na otimalidade é inferior a 10, esse número é indicado entre parênteses. Nesse caso, a média do desvio percentual é calculada em relação ao número de soluções ótimas encontradas. Quando não foi encontrada nenhuma solução ótima, o desvio percentual foi desconsiderado. O tempo máximo de execução do método exato foi de 5000 segundos, por execução [como é realizada uma busca dicotômica, podem ser realizadas várias execuções do algoritmo exato para o DVRP (*Distance-Constrained Vehicle Routing Problem*)].

As variáveis utilizadas nas Tabelas 2 a 5 são definidas a seguir:

m = número de máquinas;

n = número de tarefas;

$$DP_i = \left(\frac{SH_i - SO}{SO} \right) * 100;$$

onde: SO = solução ótima do problema;

DP_i = desvio percentual para $i \in \Re \;\; 1 \leq i \leq 6$;

SH_1= solução encontrada pela estratégia evolutiva *ESched1*;

SH_2= solução encontrada pela estratégia evolutiva *ESched2*;

SH_3= solução encontrada pela estratégia evolutiva *ESched3*;

SH_4= solução encontrada pelo algoritmo *Busca Tabu Fast*;

SH_5= solução encontrada pelo algoritmo *Busca Tabu Long*;

SH_6= solução encontrada pelo algoritmo Memético;

Tabela 2 - Desvio Percentual em relação a Solução Ótima para as Instâncias Não Estruturadas, com $s_{ij} \in$ [0,10].

n	***m***	***DP₁***	***DP₂***	***DP₃***	***DP₄***	***DP₅***	***DP₆***
20	2	**0,49815**	1,04318	0,52537	3,58002	1,77687	0,74460
40	2	0,45899	1,12685	**0,39762**	4,96503	2,98822	0,91606
60	2(6)	**0,26902**	0,76773	0,28499	4,77494	3,05366	0,79029
80	2(2)	0,17432	0,63003	**0,14810**	5,47224	3,38698	0,69727
80	4(1)	0,70140	1,70341	**0,50100**	6,21242	3,40681	1,30260
Média		0,47857	1,08502	**0,46149**	4,27252	2,38254	0,83033

De acordo com a Tabela 2, observa-se que as soluções encontradas pelas heurísticas *ESched1* e *ESched3* apresentaram melhor desempenho em relação às outras, pois têm menor *DPs,* quando os tempos de preparação estão entre [0,10].

Tabela 3 - Desvio Percentual em relação a Solução Ótima para as Instâncias Não Estruturadas, com sij $\in$ [0,100].

N	***m***	***DP₁***	***DP₂***	***DP₃***	***DP₄***	***DP₅***	***DP₆***
20	2	3,20005	5,83669	**2,86589**	30,73044	15,09432	3,98870
20	4(2)	8,61599	9,59436	8,84586	17,54201	8,02781	**6,84180**
20	6(1)	21,68675	**0,60241**	13,85542	16,26506	4,81928	4,21687
20	8(4)	18,30900	10,44660	17,43520	7,26959	7,45499	**5,56941**
40	2(9)	**3,21181**	7,69185	3,43749	43,88907	27,23059	8,71014
60	2	**2,98110**	7,03257	3,28299	45,05205	31,11548	9,47882
80	2	3,20413	6,61094	**3,15159**	49,19504	33,53166	9,82810
Média		3,12843	6,49340	**3,00874**	41,65918	26,58049	7,76521

A Tabela 3 mostra que as soluções encontradas pelas heurísticas *ESched1*, *ESched2, ESched3* e Memético apresentam melhor desempenho em relação à *Busca Tabu Fast e à Busca Tabu Long*, pois apresenta-

ram menores *DPs,* quando o tempo de preparação das tarefas está entre [0,100].

Tabela 4 - Desvio Percentual em relação a Solução Ótima para as Instâncias Estruturadas, com sij ∈ [0,10].

n	*m*	DP_1	DP_2	DP_3	DP_4	DP_5	DP_6
20	2	**0,43877**	1,03771	0,45607	3,42725	1,85222	1,03078
40	2	0,48514	1,09712	**0,45292**	4,40787	2,78035	0,98799
60	$2_{(8)}$	**0,48268**	1,17168	0,51809	4,83009	3,16866	1,14257
80	$2_{(6)}$	0,53340	0,93771	**0,53245**	5,24089	3,53873	1,02567
Média		**0,48500**	1,06106	0,48988	4,47653	2,83499	1,04675

Observa-se que as soluções encontradas pelas heurísticas *ESched1* e *ESched3* apresentam melhor desempenho em relação à *ESched2, Busca Tabu Fast, Busca Tabu Long* e Memético*,* pois apresentaram menores *DPs,* quando o tempo de preparação da máquina é de [0,10].

Tabela 5 - Desvio Percentual em relação a Solução Ótima para as Instâncias Estruturadas, com $s_{ij} \in$ [0,100].

n	*m*	DP_1	DP_2	DP_3	DP_4	DP_5	DP_6
20	2	**1,46198**	4,04270	2,07766	22,42903	11,16399	2,22707
20	$4_{(2)}$	6,74157	7,72472	6,60112	18,11798	7,44382	**2,38764**
20	$8_{(3)}$	42,71330	42,14048	45,47570	36,09232	34,47919	**32,03424**
40	2	**3,06321**	8,00558	3,45993	35,65687	23,32451	8,06981
60	2	3,48017	7,08892	**3,09104**	38,21440	26,50852	8,13925
80	$2_{(8)}$	**3,01088**	7,18880	3,17876	40,34468	29,96709	8,02903
Média		**10,07852**	12,69853	10,64737	31,80921	22,14785	10,14784

Os resultados obtidos mostram que as soluções encontradas pelas heurísticas *ESched1*, *ESched3* e Memético apresentaram melhor desempenho em relação à *ESched2, Busca Tabu Fast, Busca Tabu Long,* pois apresentaram menores DPs, quando o tempo de preparação da máquina é de [0,100].

Pode-se notar que, em todos os exemplos, existe uma maior dificuldade quando $2{,}5 \leq n/m \leq 5$. Isso se deve ao fato de que, como a média de tarefas por máquina é pequena, existe um número reduzido de combinações que levam à solução ótima.

As Tabelas 6 a 9 mostram a relação da metaheurística *ESched1* com as demais metaheurísticas desenvolvidas neste trabalho, inclusive ela, bem como as propostas pela literatura.

Tabela 6 - Resultado da média da relação entre a solução do *ESched1* entre si e com as demais metaheurísticas para Instâncias Não Estruturadas, $s_{ij} \in [0,10]$.

n	*m*	SH_1/SH_1	SH_2/SH_1	SH_3/SH_1	SH_4/SH_1	SH_5/SH_1	SH_6/SH_1
20	2	**1,00000**	1,00545	1,00036	1,03051	1,01271	1,00254
20	4	**1,00000**	1,01296	1,00222	1,01481	1,00333	0,99778
20	6	1,00000	1,01789	1,00632	1,00053	0,98579	**0,98105**
20	8	1,00000	1,03614	0,99926	0,99779	0,99705	**0,98156**
40	2	1,00000	1,00657	**0,99942**	1,04481	1,02511	1,00444
40	4	**1,00000**	1,01304	1,00019	1,03798	1,01611	1,00211
40	6	1,00000	1,02181	0,99970	1,02535	1,00884	**0,99764**
40	8	1,00000	1,02491	0,99962	1,01019	0,99283	**0,98679**
60	2	**1,00000**	1,00537	1,00032	1,04422	1,02758	1,00524
60	4	1,00000	1,01305	**0,99986**	1,04835	1,02363	1,00536
60	6	**1,00000**	1,02049	1,00116	1,03751	1,01334	1,00329
60	8	1,00000	1,02566	0,99788	1,02831	1,00450	**0,99603**
80	2	1,00000	1,00401	**0,99985**	1,05128	1,03016	1,00472
80	4	**1,00000**	1,01168	1,00029	1,05034	1,02669	1,00657
80	6	**1,00000**	1,01904	1,00090	1,04648	1,02084	1,00630
80	8	**1,00000**	1,02301	1,00193	1,03597	1,01315	1,00193

Quando a análise considera a relação entre os métodos, observa-se que a relação do *ESched1/ESched1, ESched3/ESched1,* Meméti-co*/ESched1* apresenta resultados superiores aos *ESched2/ ESched1,Tabu Fast/ESched1 e Tabu Long/ESched1.* Observa-se também que, quando o número de tarefas executadas por máquina fica entre 3 e

8, o algoritmo Memético apresenta melhor resultado.

Tabela 7 - Resultado da média da relação entre a solução do *ESched1* entre si e com as demais metaheurísticas para Instâncias Não Estruturadas, $s_{ij} \in [0,100]$.

n	***m***	SH_1/SH_1	SH_2/SH_1	SH_3/SH_1	SH_4/SH_1	SH_5/SH_1	SH_6/SH_1
20	2	**1,00000**	1,02586	0,99711	1,26459	1,11451	1,00765
20	4	1,00000	1,00210	1,02190	1,08968	0,98350	**0,97180**
20	6	1,00000	0,96023	1,01180	0,97771	0,94231	**0,93881**
20	8	1,00000	0,95069	1,00106	0,94061	0,93001	**0,91145**
40	2	**1,00000**	1,04407	1,00363	1,38859	1,22873	1,05207
40	4	**1,00000**	1,06135	1,01758	1,30867	1,12656	1,03849
40	6	1,00000	1,04157	1,00408	1,18669	1,05935	**0,99928**
40	8	1,00000	1,00206	0,98588	1,06914	**0,94851**	**0,94851**
60	2	**1,00000**	1,03911	1,00239	1,40196	1,27404	1,06067
60	4	1,00000	1,04602	**0,99760**	1,35957	1,20345	1,04048
60	6	**1,00000**	1,05160	1,00500	1,29146	1,10975	1,02226
60	8	1,00000	1,04352	**0,99404**	1,22355	1,03733	0,99863
80	2	1,00000	1,03307	**0,99962**	1,44314	1,29267	1,06355
80	4	**1,00000**	1,04653	1,00336	1,41361	1,25738	1,06289
80	6	**1,00000**	1,05780	1,00437	1,35145	1,18719	1,04269
80	8	**1,00000**	1,06434	1,00478	1,30911	1,11822	1,02907

Observa-se que as relações do *ESched1/ESched1, ESched3/ESched1, Tabu Long/ESched1* e Memético*/ESched1* apresentam os melhores resultados em relação aos *ESched2/ ESched1* e *Tabu Fast/ESched1*.

Um dos fatores que leva a melhorar a qualidade de solução do algoritmo Memético é a busca local utilizada.

Tabela 8 - Resultado da média da relação entre a solução do ESched1 entre si e com as demais metaheurísticas para Instâncias Estruturadas, sij ∈ [0,10].

n	*m*	SH_1/SH_1	SH_2/SH_1	SH_3/SH_1	SH_4/SH_1	SH_5/SH_1	SH_6/SH_1
20	2	**1,00000**	1,00570	**1,00000**	1,02933	1,01365	1,00591
20	4	1,00000	1,01365	**0,99852**	1,01070	1,00111	1,00000
20	6	1,00000	1,02908	1,00415	1,00935	0,99533	**0,99013**
20	8	1,00000	1,03609	0,99716	1,00000	0,98938	**0,97523**
40	2	1,00000	1,00597	**0,99971**	1,03871	1,02282	1,00501
40	4	1,00000	1,01389	**0,99868**	1,02928	1,01483	1,00300
40	6	**1,00000**	1,02662	1,00198	1,02209	1,01076	1,00085
40	8	1,00000	1,03638	1,00148	1,01188	0,99703	**0,99146**
60	2	**1,00000**	1,00642	1,00026	1,04165	1,02521	1,00589
60	4	**1,00000**	1,01319	1,00061	1,03712	1,02210	1,00574
60	6	1,00000	1,02029	**0,99903**	1,02957	1,01449	1,00290
60	8	1,00000	1,02963	0,99895	1,02255	1,00760	**0,99764**
80	2	**1,00000**	1,00447	1,00019	1,04622	1,02804	1,00525
80	4	**1,00000**	1,01209	1,00066	1,04063	1,02504	1,00671
80	6	**1,00000**	1,01849	1,00115	1,03987	1,02152	1,00664
80	8	**1,00000**	1,02861	**1,00000**	1,03088	1,01383	1,00246

Tabela 9 - Resultado da média da relação entre a solução do *ESched1* entre si e com as demais metaheurísticas para Instâncias Estruturadas, $s_{ij} \in [0,100]$.

n	*m*	SH_1/SH_1	SH_2/SH_1	SH_3/SH_1	SH_4/SH_1	SH_5/SH_1	SH_6/SH_1
20	2	**1,00000**	1,02487	1,00559	1,20129	1,09226	1,00692
20	4	1,00000	1,01664	1,00581	1,07186	0,99392	**0,98336**
20	6	1,00000	0,98960	0,99703	0,96398	0,93947	**0,93465**
20	8	1,00000	0,97281	0,99032	0,94470	0,92719	**0,91889**
40	2	**1,00000**	1,04804	1,00363	1,31433	1,19564	1,04877
40	4	1,00000	1,05647	**0,99834**	1,21138	1,08999	1,00725
40	6	1,00000	1,03822	0,99056	1,12475	1,03602	**0,97562**
40	8	1,00000	1,02979	0,99887	1,05362	0,95319	**0,95206**
60	2	1,00000	1,03471	**0,99620**	1,33440	1,22190	1,04483
60	4	**1,00000**	1,05401	1,00582	1,27506	1,17544	1,03812
60	6	1,00000	1,04945	**0,98960**	1,20911	1,08925	1,00090
60	8	1,00000	1,03679	0,98265	1,13782	1,01830	**0,97674**
80	2	**1,00000**	1,03880	1,00300	1,36449	1,25987	1,04696
80	4	**1,00000**	1,04279	1,00284	1,32350	1,20470	1,03671
80	6	1,00000	1,04670	**0,99796**	1,26197	1,14392	1,03209
80	8	**1,00000**	1,05498	1,00856	1,20715	1,09599	1,01908

Tabela 10 - Tempo de CPU milisegundos, Instâncias Não Estruturadas, $s_{ij} \in [0,10]$.

n	*m*	*ESched1*	*ESched2*	*ESched3*	*Tabu Fast*	*Tabu Long*	*Memético*
20	2	199,3	302,0	**168,4**	590,7	10736,3	120041,8
20	4	170,5	319,4	**159,4**	431,2	8700,2	120042,8
20	6	168,5	302,3	**155,3**	357,3	7658,9	120051,3
20	8	162,3	311,5	**160,2**	320,1	7304,4	120054,4
40	2	242,4	666,2	**233,3**	1910,6	73774,9	120020,4
40	4	**192,5**	539,8	194,0	1313,0	23198,9	120022,4
40	6	184,3	520,4	**179,1**	968,7	13733,6	120022,0
40	8	**178,9**	564,9	179,5	730,3	11920,5	120025,2
60	2	354,1	1221,2	**346,3**	14450,3	554395,4	120032,7
60	4	228,8	939,8	**228,2**	2795,9	109968,2	120036,1
60	6	210,3	891,5	**221,5**	1888,9	45694,4	120028,4
60	8	**206,6**	855,3	208,0	1278,6	20238,2	120062,3
80	2	530,4	2245,2	**509,0**	27631,5	1958231,0	120054,7
80	4	309,1	1733,2	**293,2**	5920,9	398410,9	120069,1
80	6	253,1	1464,2	**248,3**	3383,4	147280,0	120052,1
80	8	**255,2**	1422,3	269,4	2819,9	69984,6	120058,8

Tabela 11 -Tempo de CPU milisegundos, Instâncias Não Estruturadas, $s_{ij} \in [0,100]$.

n	*M*	*ESched1*	*ESched2*	*ESched3*	*Tabu Fast*	*Tabu Long*	*Memético*
20	2	185,1	349,1	**168,3**	604,0	10826,4	120047,4
20	4	**157,1**	316,4	160,1	469,9	8182,8	120055,9
20	6	**154,6**	318,9	155,9	383,5	7445,8	120047,3
20	8	**159,0**	360,7	159,4	328,8	7226,1	120050,9
40	2	240,8	675,0	**237,6**	2130,0	74301,1	120031,2
40	4	195,6	588,5	**186,6**	1271,7	26994,5	120020,8
40	6	185,9	558,5	**180,8**	993,8	12504,5	120040,1
40	8	**180,0**	587,4	180,6	868,9	10769,9	120050,9
60	2	356,8	1220,9	**356,0**	12100,2	494738,6	120029,1
60	4	231,0	1033,2	**230,4**	3613,0	120048,4	120032,1
60	6	210,5	945,0	**209,6**	2161,2	47512,0	120025,3
60	8	**206,7**	910,5	239,3	1939,0	24798,4	120047,4
80	2	563,3	2038,0	**530,4**	27214,7	2059991,0	120057,5
80	4	293,8	1605,3	**291,5**	7867,6	384663,9	120092,2
80	6	261,2	1619,0	**250,5**	4799,4	167017,1	120069,7
80	8	268,3	1830,3	**244,8**	3788,7	74922,0	120071,8

Tabela 12 - Tempo de CPU milisegundos, Instâncias Estruturadas, $s_{ij} \in [0,10]$.

n	*M*	*ESched1*	*ESched2*	*ESched3*	*Tabu Fast*	*Tabu Long*	*Memético*
20	2	168,9	234,2	172,3	574,7	10355,3	120085,3
20	4	160,4	221,5	197,2	420,1	7741,0	120083,3
20	6	154,8	233,6	187,4	385,3	7327,3	120151,3
20	8	158,6	234,2	199,1	344,3	6946,9	120073,9
40	2	235,6	439,9	240,8	1843,3	77068,5	120024,5
40	4	190,3	354,1	189,2	1215,6	23844,0	120021,8
40	6	190,9	343,5	179,2	945,3	11376,8	120026,4
40	8	179,7	351,1	179,7	759,6	10222,6	120026,6
60	2	364,9	771,5	345,3	11727,0	536860,2	120034,2
60	4	230,5	574,7	236,7	2742,7	112310,5	120042,0
60	6	210,8	540,2	210,8	2241,2	50434,6	120033,4
60	8	220,1	534,2	209,8	1732,9	23515,9	120022,4
80	2	546,8	1278,3	518,5	24922,3	2080070,0	120059,1
80	4	316,7	894,9	294,9	7629,8	359401,7	120054,5
80	6	254,4	805,8	264,3	4722,5	170064,0	120041,1
80	8	253,8	759,5	261,7	3110,3	84724,4	120050,8

Tabela 13 - Tempo de CPU milisegundos, Instâncias Estruturadas, $s_{ij} \in [0,100]$.

n	*m*	*ESched1*	*ESched2*	*ESched3*	*TabuFast*	*TabuLong*	*Memético*
20	2	173,8	232,7	169,6	619,5	10709,2	120045,8
20	4	160,1	220,8	158,2	469,0	8348,8	120050,2
20	6	154,6	227,9	164,8	381,3	7457,3	120037,7
20	8	158,8	235,0	159,5	345,4	7296,6	120073,8
40	2	247,7	421,4	241,9	1872,6	74530,5	120024,4
40	4	200,4	348,3	189,5	1336,6	26596,7	120032,3
40	6	177,9	339,7	195,8	871,4	12258,3	120027,3
40	8	181,3	364,4	182,2	873,7	10909,2	120022,5
60	2	388,6	732,2	374,6	11827,6	529565,5	120023,5
60	4	235,0	581,1	265,7	3987,8	114075,5	120034,7
60	6	208,6	522,8	210,5	2066,0	50880,1	120033,6
60	8	205,9	521,1	208,4	1810,9	27421,8	120034,6
80	2	609,0	1160,2	592,1	28720,8	2027927,0	120035,3
80	4	301,6	897,5	296,2	6159,4	354919,1	120052,9
80	6	258,6	783,3	267,4	4099,6	184386,2	120042,8
80	8	257,4	769,4	270,5	3315,6	83985,8	120080,2

Em relação ao tempo computacional gasto pelos métodos, pode-se observar que as estratégias evolutivas consomem o menor tempo, quando comparadas com os métodos da literatura. Quanto ao algoritmo Memético, é importante lembrar que Mendes (1999) fixou o tempo em 2 minutos.

Outra avaliação que pode ser feita em relação ao tempo computacional é que as EEs utilizaram menos esforço computacional quando comparadas com os métodos da literatura, em torno de 50 a 1000 vezes menos.

Levando em conta a solução e o esforço computacional, as estratégias evolutivas apresentam um bom desempenho para o P|tpds|Cmax, mesmo utilizando buscas locais simples.

Da Figura 8 a 16, estão apresentados alguns gráficos, para visualizar o comportamento das soluções encontradas. Os gráficos apresenta-

dos referem-se aos casos em que se sabe a existência da solução ótima de todo o conjunto de dez instâncias do mesmo tipo.

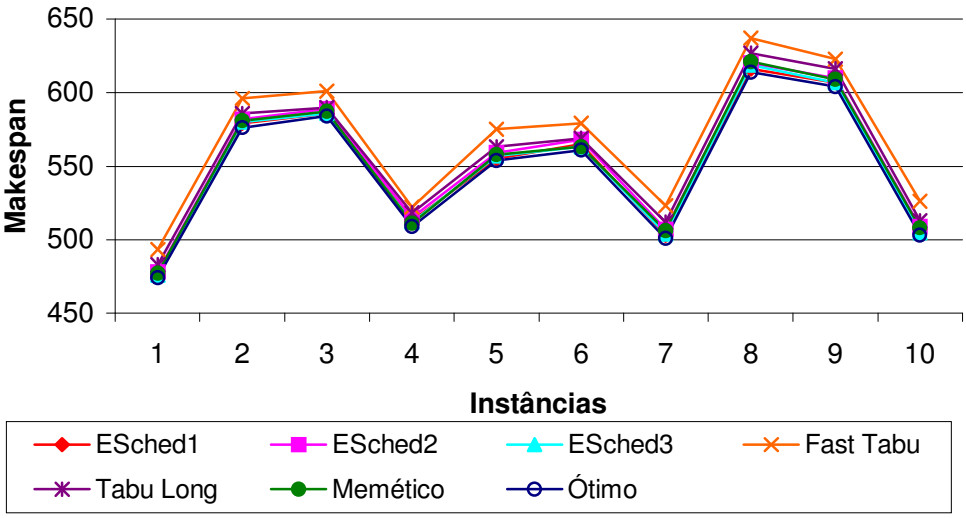

Figura 8 – Valor da solução para dez instâncias testadas do tipo não estruturado para n/m = 20/2, $s_{ij} \in [0,10]$.

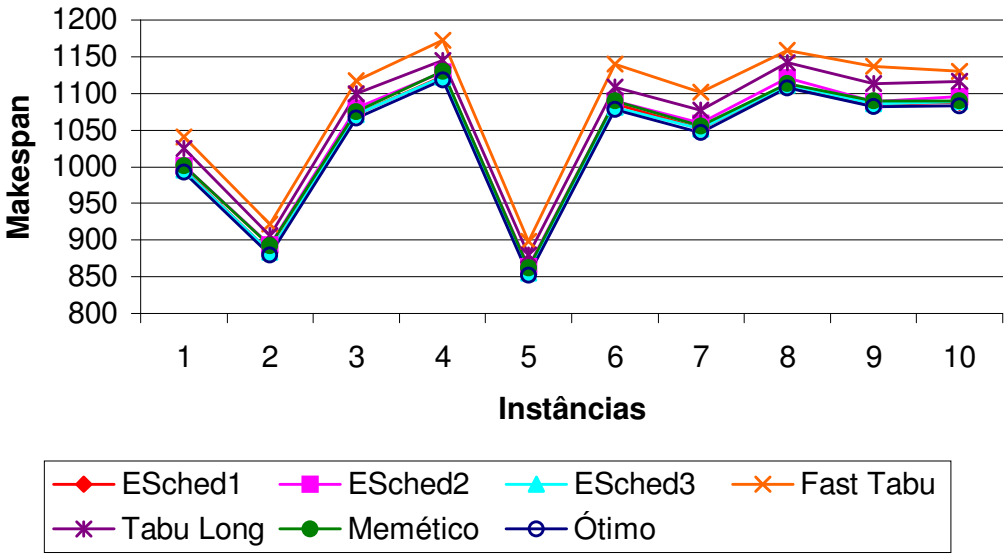

Figura 9 – Valor da solução para dez instâncias testadas do tipo não estruturado para n/m = 40/2, $s_{ij} \in [0,10]$.

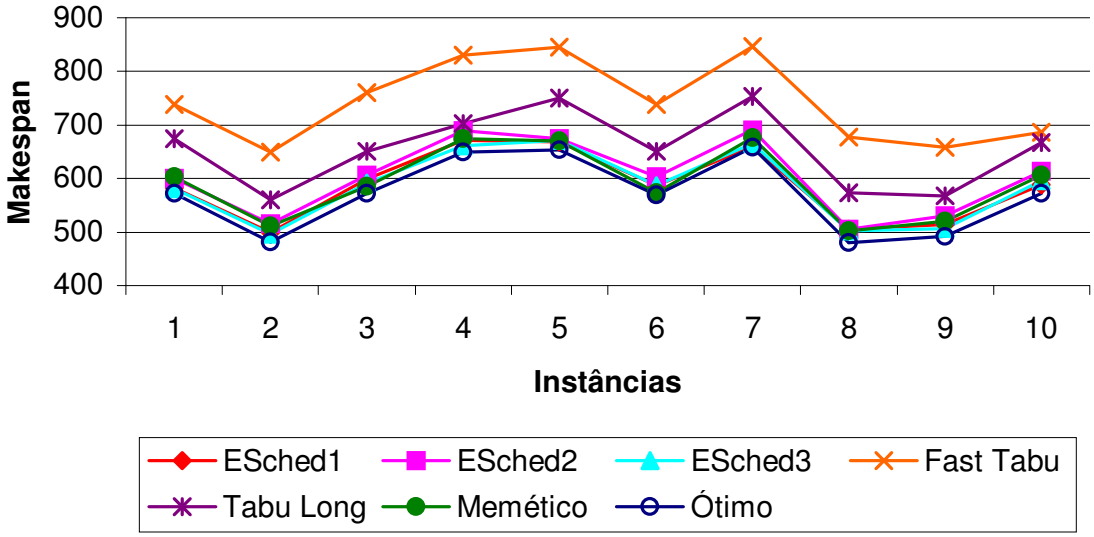

Figura 10 – Valor da solução para dez instâncias testadas do tipo não estruturado para n/m = 20/2, $s_{ij} \in [0,100]$.

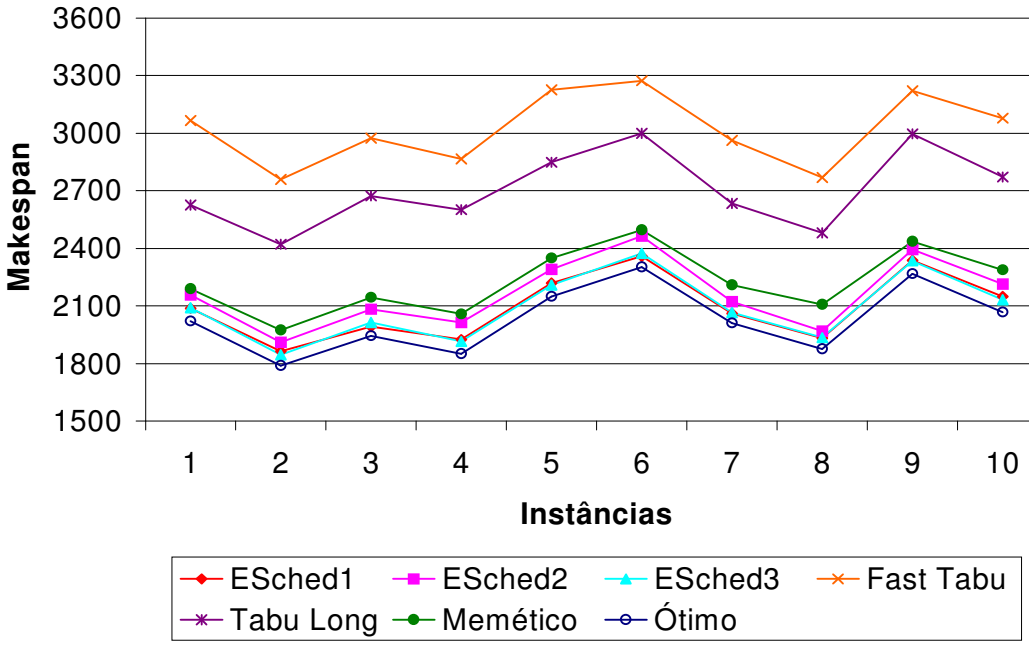

Figura 11 – Valor da solução para dez instâncias testadas do tipo não estruturado para n/m = 80/2, $s_{ij} \in [0,100]$.

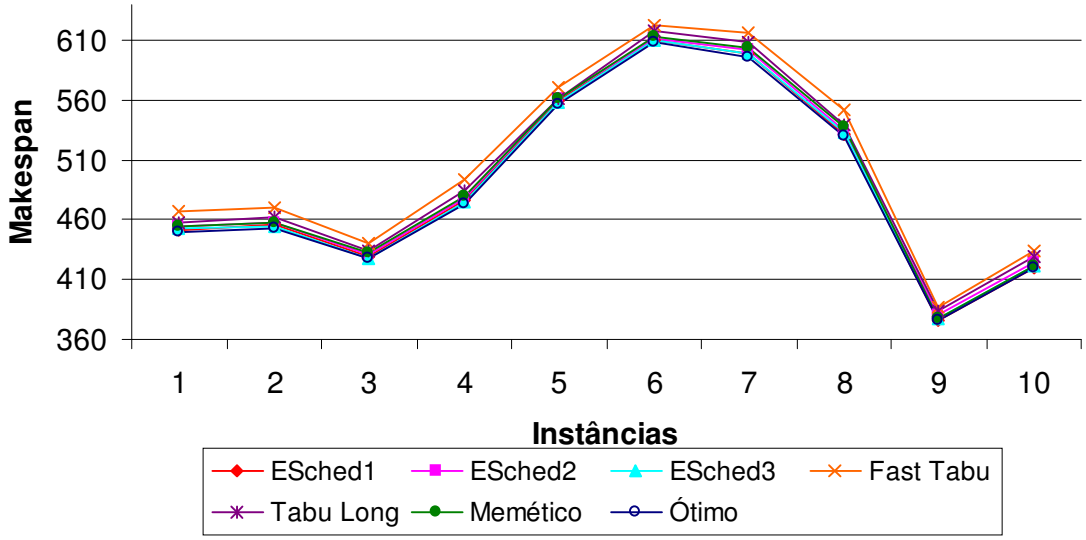

Figura 12 – Valor da solução para dez instâncias testadas do tipo estruturado para n/m = 20/2, $s_{ij} \in [0,10]$.

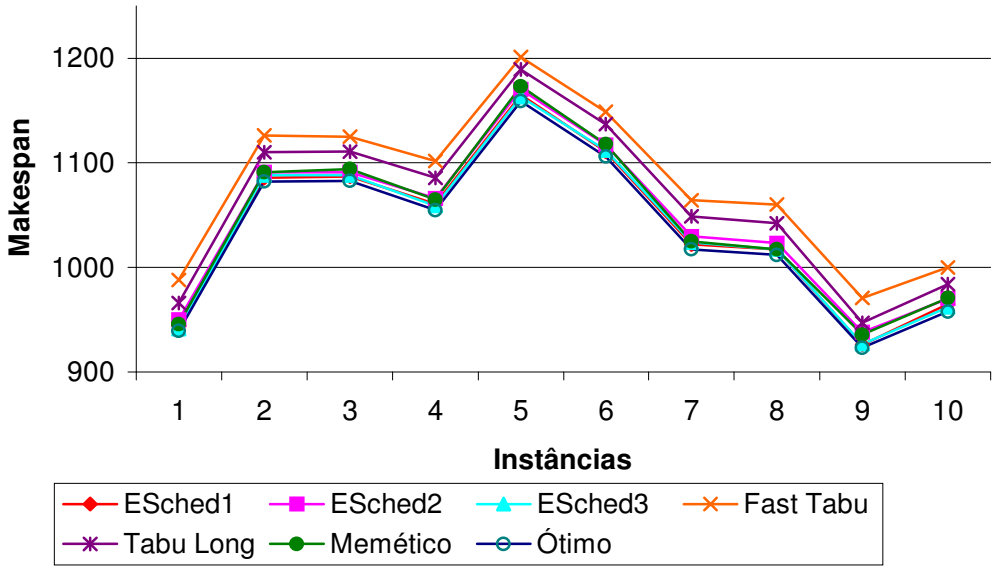

Figura 13 – Valor da solução para dez instâncias testadas do tipo estruturado para n/m = 40/2, $s_{ij} \in [0,10]$.

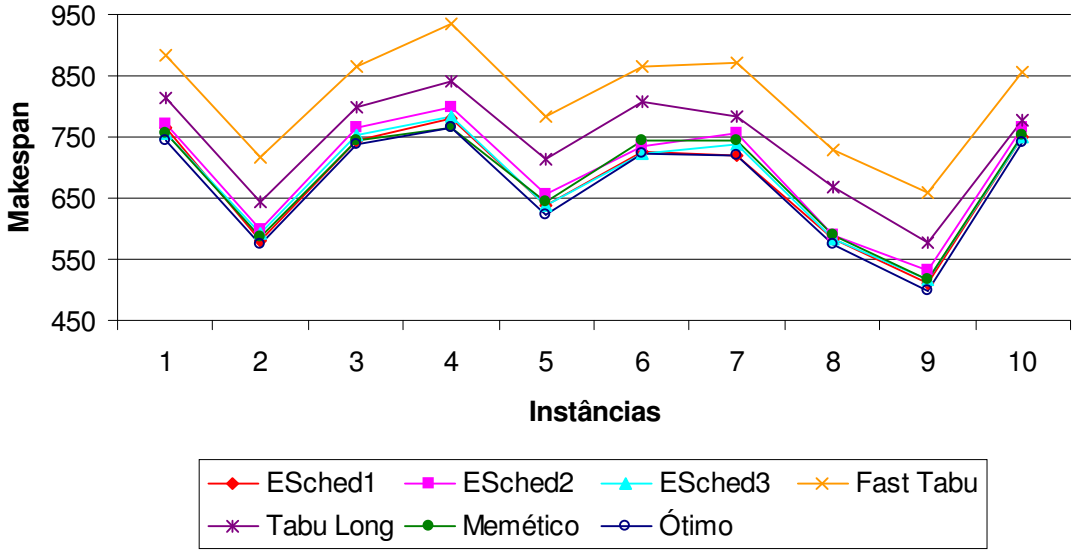

Figura 14 – Valor da solução para dez instâncias testadas do tipo estruturado para n/m = 20/2, $s_{ij} \in [0,100]$.

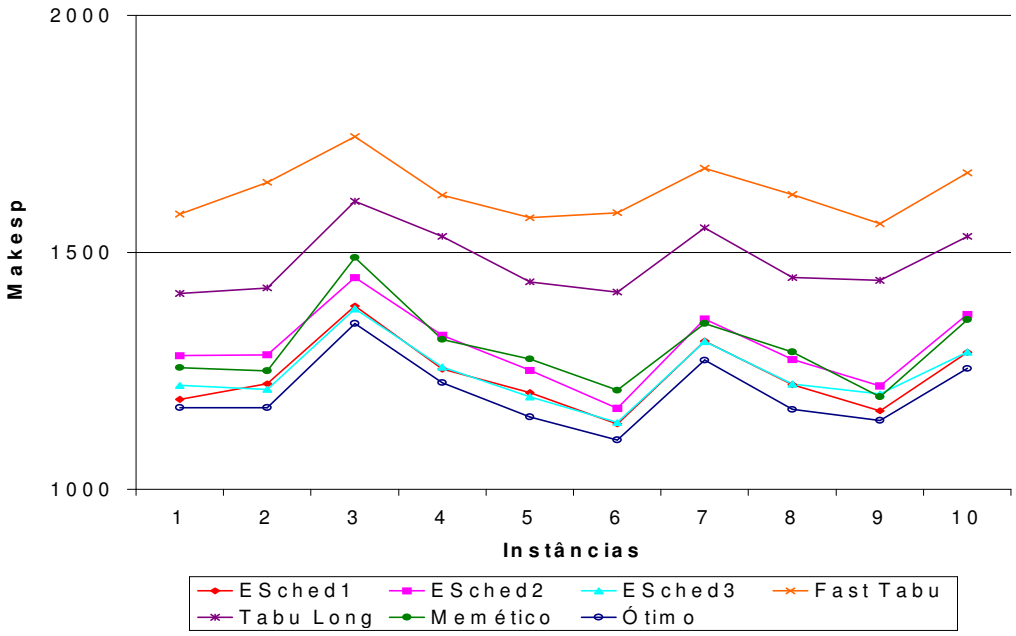

Figura 15 – Valor da solução para dez instâncias testadas do tipo estruturado para n/m = 40/2, $s_{ij} \in [0,100]$.

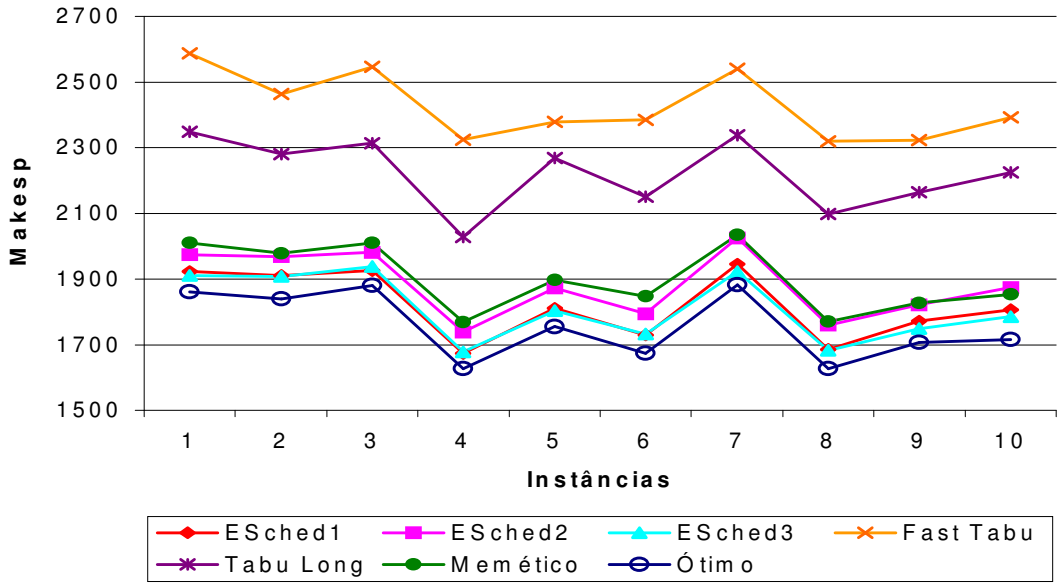

Figura 16 – Valor da solução para dez instâncias testadas do tipo estruturado para n/m = 80/2, $s_{ij} \in [0,100]$.

O comportamento das linhas do gráfico é bastante semelhante. Isso se deve à questão de que todos utilizam uma busca local baseada em caixeiro viajante. Caso existisse um método com uma busca local específica para P|tpds|Cmax, o comportamento poderia ser diferente.

Nas Figuras 5, 6, 8, 9 e 11, podemos verificar que as soluções das Estratégias Evolutivas estão bem próximas da solução ótima, seguido do Algoritmo Mémetico, e a Busca Tabu encontra-se mais afastada da solução ótima.

No próximo capítulo serão feitas as conclusões do trabalho e também a apresentação de sugestões para trabalhos futuros.

5 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, foram propostas três estratégias evolutivas para um problema NP-difícil, o P|tpds|Cmax. As estratégias evolutivas foram desenvolvidas adaptando as idéias de Homberger e Gehring (1999) para o problema de rotemento de veículos com janelas de tempo a um problema de seqüenciamento, buscando obter algoritmos simples, robustos e rápidos.

As soluções encontradas pelas estratégias evolutivas desenvolvidas neste trabalho trouxeram resultados satisfatórios em qualidade de solução, pois para as instâncias estruturadas e não estruturadas com tempo de preparação das tarefas entre [0,10] obteve-se melhores soluções para as estratégias evolutivas em 75% dos casos. Em instâncias estruturadas com tempo de preparação das tarefas entre [0,100] obteve-se melhores resultados em 63% das soluções. E para instâncias não estruturadas com tempos de preparação entre [0,100] os resultados foram melhores em 69% das soluções. Os poucos casos onde as soluções das Estratégias Evolutivas não apresentaram melhor solução que a Busca Tabu e Algoritmo Memético foram quando a relação entre o número de tarefas e o número de máquinas era pequena, ou seja, $2{,}5 \leq n/m \leq 5$. Isso se deve ao fato de que, existe um número reduzido de combinações que levam à solução ótima, tornando o problema mais difícil de resolver.

Fazendo uma análise de todas as metaheurísticas utilizadas neste trabalho, conclui-se que elas apresentam soluções boas para as instâncias estruturadas, isso se deve ao formato e as buscas locais utilizadas pelas metaheurísticas que foram originalmente desenvolvidas para problemas de roteamento, cujas características são presentes nas instâncias estruturadas. As estratégias evolutivas apresentam boas soluções independentes das instâncias serem estruturadas ou não, isso ocorre devido a

combinação de várias buscas locais e, também, devido à aleatoriedade dos algoritmos populacionais, que minimizam o impacto de buscas locais originalmente desenvolvidas para outros problemas.

Dentre as estratégias evolutivas propostas temos melhores soluções em *ESched1* e *ESched3*, tanto para o caso estruturado e não estruturado, com $s_{ij} \in$ [0,10] e [0,100].

O tempo computacional das Estratégias Evolutivas foi em torno de 50 a 1000 vezes, melhor quando comparadas com a busca tabu e o memético, pois se utilizam buscas locais simples, de fácil implementação e que são rápidas.

Para trabalhos futuros, sugere-se que sejam utilizadas, buscas locais mais elaboradas, por exemplo, GENIUS ou a busca local do Algoritmo Memético, pois se tem uma margem de tempo computacional disponível para trabalhar.

Outra sugestão seria desenvolver uma busca local específica para o P|tpds|Cmax, e não adaptações do problema de roteamento de veículos.

Também pode-se aplicar estratégias para manter a diversidade da população no decorrer das iterações.

6 REFERÊNCIAS BIBLIOGRÁFICAS

AMIN, S. ***Simulated Jumping***, Annals of Operations Research 86, p.23-38, 1999.

BAKER, K. R. ***Introduction to Sequencing and Scheduling***, John Wiley & Sons Inc., 1974.

BURIOL, L.; FRANÇA, P.M.; MOSCATO, P. ***Recursive Arc Insertion: a new Local Search Embedded in a Memetic Algorithm for the Asymmetric Traveling Salesman Problem***, Journal of Heuristics, 1999.

CAI, J.; THIERAUF, G. ***Evolution Strategies for Solving Discrete Optimization Problems***, Advances in Engineering Software v. 25, p.177-183, 1996.

CLARKE, G.; WRIGHT, J. ***Scheduling vehicles from a Central Depot to a Number of Delivery Points***, Operations Research, v.12, p.568-581, 1964.

DEARING, P.M.; HENDERSON, R.A. ***Assigning Looms in a Textile Weaving Operation with Changeover Limitations,*** Production and Inventory Management v. 25, p. 23-31, 1984.

FEO, T.; REZENDE, M. ***A Greed Randomized Adaptive Search Procedure For Maximum Independent Set***, Operations Research 42, p.860-879,1995.

FIECHTER, C. ***A Parallel Tabu Search Algorithm for Large Traveling Salesman Problems***, Discrete Applied Mathematics 51, p.243-267, 1994.

GAREY, M.R.; JONHSON, D.S. ***Computers and Intractability: A Guide to the Teory of NP-Completeness***, San Francisco: Freeman, 1979.

GENDREAU, M.; HERTZ, A.; LAPORTE, G., ***New Insertion e Post-Optimization Procedures for the Traveling Salesman Problem***, Operations Research, v. 40(6), p.2086-1094, 1992.

GLOVER, F.; ***Tabu Search: a Tutorial***, CAAI Report, University of Colorado, Boulder, 1990.

GLOVER, F.; LAGUNA, M. ***Tabu Search***, Colin Reeves (ed.), em *Modern Heuristic Techniques*. Blackwell Scientific Publications, Oxford, Blackwell, p.70-150, 1993.

GOLDBARG, M. C.; LUNA, H. P. ***Otimização Combinatória e Programação Linear: Modelos e Algoritmos***, Rio de Janeiro: Campus, 2000.

GOLDBERG, D. E. ***Genetic Algorithms in Search, Optimization, and Machine Learning***, Addison-Wesley, 1989.

GRAHAM, R.L.; LAWLER, E.L.; LENSTRA, J.K.; RYNNOY KAN, A.H.G ***Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey***, Annals of Discrete Mathematics, v. 5, p.287-326, 1979.

GUINET, A. ***Textile Production Systems: a Succession of Non-Identical parallel processor shops***, Journal of the Operational Reasearch Society, v. 42(8), 655-671, 1991.

GUPTA, A.K.; GREENWOOD, G.W. ***Static Task Allocation Using (μ, λ) Evolutionary Strategies***, Information Sciences v. 94, p. 141-150, 1996.

HOMBERGER, J.; GEHRING, H. ***Two evolutionary Metaheuristics for the Vehicle Routing Problem with Time Windows***, Information Systems and Operational Research – Special issue: Metaheuristics for location and routing problems, p.297-318, 1999.

KIRKPATRICK, S.; GELLAT, C.D.; VECCHI, M.P. ***Optimization by Simulated Annealing***, Science 220 (4598), p.671 - 679, 1983.

MENDES, A.; MÜLLER, F.; FRANÇA, P.; MOSCATO, P. ***Comparing Meta-heuristic Approaches for Parallel Machine Scheduling Problems***, Production Planning & Control, v. 13, n. 2, p.143-54, 2002.

MOSCATO, P. ***On Evolution, Search, Optimization, Genetic Algorithms, and Matial Arts: Towards Memetic Algorithms***, Technical Report, Caltech Concurrent Computation Program, C3P Report 826, 1989.

_____, P. ***Memetic Algorithms: a Short Introduction***. In D.Corne, M. Dorigo e F. Glover (eds), New ideias in optimization, Washington, McGraw-Hill, 1999.

MÜLLER, F.M. ***Algoritmos Heurísticos e Exatos para Resolução do Problema de Seqüenciamento em Processadores Paralelos***, Tese (Doutorado em Engenharia Elétrica) – Universidade Estadual de Campinas, Campinas, 1993.

OR, I. ***Traveling Salesman-type combinatorial problems and their relation to the logistics of blood banking***, Phd. Thesis, Departament of Industrial Engineering and Management Science, Northwestern University, Evanston, IL, 1976.

OSMAN, I. ***Heuristics for Combinatorial Optimization Problems: Developments and New Directions***, In proceedings of the first seminar on information technology and applications, Marfield Conference Centre, September, 1991.

PARKER, R.G.; DEANE, R.H.; HOLMES, R.A. ***On the Use of a Vehicle Routing Algorithm for the Parallel Processors Problems with Sequence Dependent Changeover Costs***, AIIE Transactions, v. 9(2), p.155-160, 1977.

POTVIN, J. Y. ***The Traveling Salesman Libraly***, ORSA Journal on Computing v.3, p.376-384, 1993.

POTVIN, J.Y.; ROUSSEAU, J.M. ***An Exchange Heuristic for Routing Problems with Time Windows***, Journal of the Operational Research Society, 46, 1433-1446, 1995.

RAVETTI, M. G. ***Problemas de Seqüenciamento com Máquinas Paralelas e Tempos de Preparação Dependentes da Seqüência***, Dissertação (Mestrado em Ciência de Computação) – Universidade Federal de Minas Gerais, Minas Gerais, 2003.

RECHENBERG, I***. Evolutionsstrategie: Optimirerung Technisher Systeme nach Prinzippien der Biologischen Evolution***, Frommann-Holzboog, Stuttgart, 1973.

SEDEWICK, R. ***Algorithms in C***, Addison-Wesley, USA, 1990.

STÜTZLE, T.; DORIGO, M. ***ACO Algorithms for the Traveling Salesman Problem***, To appear in K. Miettinen, M. Mäkelä, P. Neittaanmäki e J. Periaux, (eds), *Evolutionary Algoritms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications*, John Wiley & Sons, 1999.

SUMICHRAST, R.T.; BACKER, J.R. ***Scheduling Parallel Processors: an Integer Linear Programming based Heuristic for Minimizing Setup Time***, International Journal of Production Research, v. 25(5), p.761-771, 1987.