Algoritmo Genético aplicado ao Problema de *Job Shop* Clássico: Estudos Comparativos sobre Operadores de *Crossover* e Geração da População Inicial

Universidade Federal do Piauí
Centro de Ciências da Natureza
Departamento de Informática e Estatística

Algoritmo Genético aplicado ao Problema de *Job Shop* Clássico: Estudos Comparativos sobre Operadores de *Crossover* e Geração da População Inicial

Orientador: Prof. Dr. Antonio Costa de Oliveira
Orientando: Filipe de Oliveira Saraiva

Monografia apresentada para aprovação na disciplina Estágio Supervisionado II no curso de Ciência da Computação pela Universidade Federal do Piauí

Junho de 2009
Teresina, Piauí

Monografia apresentada para aprovação na disciplina de Estágio Supervisionado II sob o título "*Algoritmo Genético aplicado ao Problema de Job Shop Clássico: Estudos Comparativos sobre Operadores de Crossover e Geração da População Inicial*", defendida por Filipe de Oliveira Saraiva em Teresina, Estado do Piauí, em banca examinadora constituída pelos seguintes avaliadores:

_____

Prof. Dr. Antonio Costa de Oliveira

Orientador

_____

Prof. Dr. Francisco Vieira de Souza

Membro da Banca

_____

Msc. Aldir Silva Sousa

Membro da Banca

A Saraiva e Dêca, meus pais,
Por tudo que sou.

A Telma, minha madrinha,
Pelo hábito da leitura.

"Se vi tão longe, é porque estava sobre o ombro de gigantes"
Isaac Newton

Agradecimentos

Em primeiro lugar, a todos os membros do corpo discente, docente e funcionários do Departamento de Informática e Estatística da Universidade Federal do Piauí. Este trabalho é fruto dos já mais de quatro anos de convivência neste ambiente, de forma que me é impossível não refletir toda essa troca construtiva de experiências na presente monografia;

Ao Prof. Dr. Antonio Costa de Oliveira, por toda dedicação, paciência e orientação dadas nestes anos todos. Espero que este trabalho lhe traga tanto orgulho e satisfação quanto o que sinto ao vê-lo realizado. E obrigado, acima de tudo, por ter acreditado;

Aos companheiros de curso Francisco “Chicão” Borges e Herdeson Sousa, por todos os momentos que compartilhamos, tantos os alegres como os revoltantes, nesta vida de “aspirantes” a Cientistas da Computação;

Aos “comparsas” do Movimento Estudantil, por nossas reflexões e sonhos de uma sociedade mais justa para todos. Levarei nossas discussões sempre comigo. São muitos os amigos e amigas, mas gostaria de citar, em especial, Allyson Jullian, André “Café”, Salathyel Pereira, Flávio de Castro, Miguel Ernesto, André Igor, Phelipe Cunha, Natasha Karenina, Emanuel Alcântara, Emanuel “Choco” Henrique, Bianca, Cleiton, Victor e Wisllan;

A Aracele Torres, por todo carinho, dedicação e afeto que partilhamos nos últimos anos, bem como nossos debates, sempre muito enriquecedores;

A minha família, por todo carinho, aprendizado e apoio que nunca faltaram nos últimos 23 anos e ao qual sempre recorrerei;

A sociedade brasileira, que mantem este importante patrimônio que é a Universidade Pública. Mesmo esta última, as vezes e em alguns casos, sendo um tanto ingrata.

“A economia da Terra é estável, e assim há de continuar,
por se basear em decisões de máquinas calculadoras que,
pela influência irresistível da Primeira Lei da Robótica,
só se preocupam com o bem da humanidade.”

Isaac Asimov – *O Conflito Evitável*

Resumo

Problemas de Otimização Combinatória estão entre os temas mais pesquisados e discutidos no ramo de ciências como a matemática, computação, engenharias e pesquisa operacional. Nestes problemas as variáveis de decisão são discretas – ou seja, a solução é um conjunto, ou sequência, de inteiros ou outros objetos discretos. O início das pesquisas nesse tema mostrou que encontrar a solução exata para estes problemas é muito difícil, por conta do grande número de combinações existentes para as variáveis de decisão.

Por volta de meados dos anos 70, diferentes grupos de pesquisadores, confrontados essa dificuldade, começaram a partir para uma nova abordagem. A idéia era conseguir uma boa solução para o problema, não necessariamente ótima, porém suficiente para ser aceita pelo modelo, a um custo computacional viável. Começa assim o desenvolvimento e aplicação de métodos computacionais conhecidos como Heurísticas e Metaheurísticas.

Este trabalho faz um levantamento bibliográfico acerca dos problemas de Otimização Combinatória e de métodos Metaheurísticos utilizados para solucioná-los, além de implementar um Algoritmo Genético para o Problema de *Job Shop*. Na implementação, foram feitos dois estudos comparativos: um sobre operadores de *crossover* encontrados na literatura aplicados ao problema e, no segundo estudo, a análise do resultado do algoritmo onde, num primeiro cenário, a população inicial do mesmo foi gerada através da hibridização com alguns métodos heurísticos de Regras de Despacho e, no segundo cenário, a população inicial é construída totalmente de forma aleatória.

Palavras-chave: Otimização Combinatória, Problemas NP - Completo, Metaheurísticas, Algoritmo Genético, Problema de *Job Shop*.

Sumário

Figuras

Tabelas

Parte I:

Desenvolvimento do Eixo Teórico

1.0 - Problemas de Otimização Combinatória

Nos últimos cinquenta anos, os problemas de Otimização Combinatória ficaram entre os temas mais pesquisados e discutidos no ramo de ciências como a matemática, computação e engenharias (Goldbarg e Luna, 2005). Tratam-se de problemas nos quais, dado um conjunto de restrições e uma função objetivo, encontrar uma solução que satisfaça estas restrições e otimize o melhor possível o valor dessa função. Problemas desse tipo caracterizam-se pelo número muito grande de soluções viáveis, demandando um grande esforço computacional para solucioná-los de maneira exata através de métodos convencionais, como a enumeração completa, particionar e limitar, programação dinâmica e outros.

Um problema de otimização pode ser formulado da seguinte forma, de acordo com a Figura 1.1 (Reeves, 1996):

$$\begin{aligned} &\textit{Minimizar} \quad f(X) \\ &\quad \textit{sujeito a} \quad g_i(X) \geq b \quad ; i = 1, 2, \ldots, m; \\ &\qquad\qquad\quad\; h_j(X) = c_{ij} \quad ; j = 1, 2, \ldots, n \end{aligned}$$

Figura 1.1: Modelo Geral de Problemas de Otimização.

onde X é um vetor de variáveis de decisão, $f(.)$ é a função objetivo que pretende-se otimizar, e $g_i(.)$ e $h(.)$ são funções que representam as restrições. A formulação apresentada é utilizada para problemas de minimização, mas basta aplicar algumas modificações para servir de modelo também para problemas de maximização.

Existem muitos tipos específicos de problemas de otimização, classificados a partir de suas restrições, dos tipos de funções em consideração e dos valores que as variáveis de decisão poderão assumir (Reeves, 1996). Entre eles, existe aquele que caracteriza-se quando as variáveis de decisão e as restrições somente poderão assumir valores contínuos. Problemas assim são conhecidos como Problemas de Programação Linear. A Figura 1.2 apresenta o espaço de busca em um Problema de Programação Linear com duas variáveis de decisão (X_1 e X_2); no caso, estas variáveis poderão assumir qualquer valor pertencente a área hachurada de cinza no diagrama. Este tipo de problema já é bastante estudado, e a literatura traz alguns algoritmos exatos de resolução com bom desempenho computacional, como o algoritmo *simplex* (Arenales, Armentano, Morabito e Yanasse, 2007).

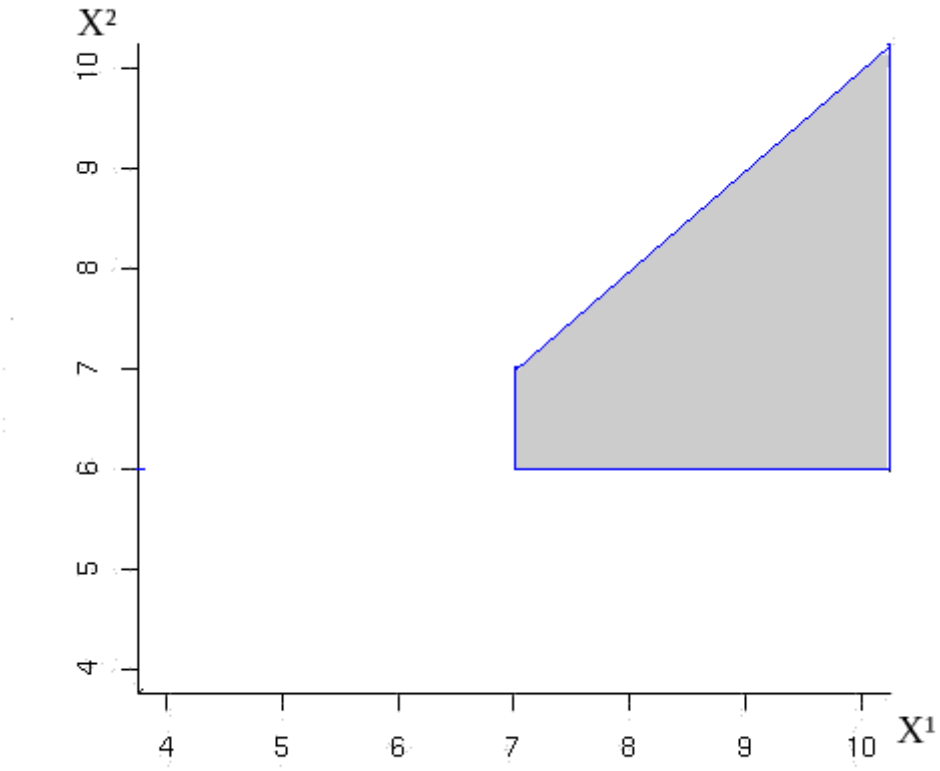


Figura 1.2: O Espaço de Busca em um Problema de Programação Linear.

Outro exemplo de tipos de problemas são aqueles nos quais uma ou mais

variáveis de decisão só podem assumir valores inteiros. Isso significa que o espaço de busca da solução não é contínuo. Problemas desse tipo pertencem ao conjunto dos Problemas de Programação Inteira. A Figura 1.3 apresenta o espaço de busca em um Problema de Programação Inteira com duas variáveis de decisão (X_1 e X_2); no caso, estas variáveis só poderão assumir valores correspondentes aos círculos incolores no espaço hachurado de cinza no diagrama. Existe uma subclasse de problemas desse tipo conhecida como Problemas de Programação Inteira 0-1, onde as variáveis de decisão só poderão assumir valores 0 ou 1 (Goldbarg e Luna, 2005).

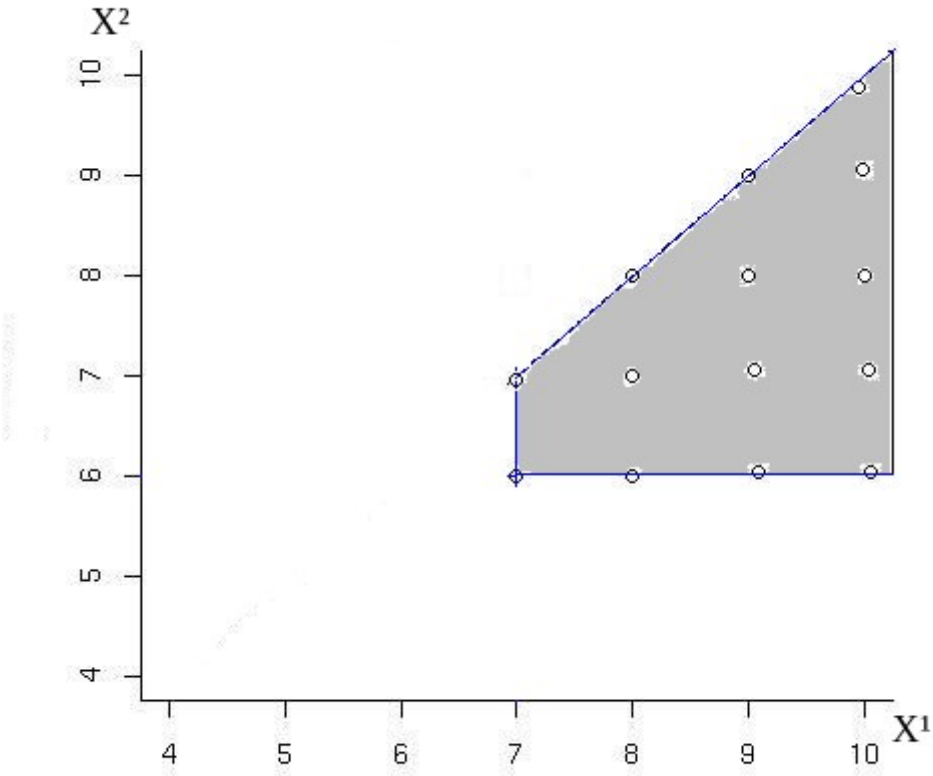


Figura 1.3: O Espaço de Busca em um Problema de Programação Inteira.

Como Problemas de Programação Linear, Problemas de Programação Inteira e Programação Inteira 0-1 são pesquisados a muitos anos, e já foram desenvolvidos alguns algoritmos exatos para eles, como o *branch-and-bound*, programação dinâmica, plano de corte e *branch-and-cut* (Arenales, Armentano, Morabito e

Yanasse, 2007). Infelizmente, estes algoritmos só se mostram eficientes para resolução de instâncias de problemas de dimensão baixa ou mediana, não sendo aplicáveis satisfatoriamente a instâncias maiores. Normalmente, estas instâncias maiores refletem o modelo de problemas encontrados em cenários reais.

Além dos problemas citados anteriormente, existe uma outra classe de problemas conhecidos como Problemas de Otimização Combinatória. Nestes problemas as variáveis de decisão são discretas – ou seja, a solução é um conjunto, ou sequência, de inteiros ou outros objetos discretos (Reeves, 1996).

No início das pesquisas em problemas de Otimização Combinatória, os pesquisadores depararam-se com dificuldades em encontrar métodos de resolução exata para alguns destes problemas. A principal causa observada foi a "explosão combinatória" proporcionada pelo grande número de combinações de valores diferentes entre as diversas variáveis de decisão que compõem o modelo do problema. O número dessas combinações é notadamente muito grande, impossibilitando métodos de resoluções exatos baseados em enumeração completa ou enumeração implícita, como os algoritmos particionar e limitar ou programação dinâmica (Reeves, 1996).

Para ilustrar, tomemos como exemplo um dos problemas de otimização combinatória mais tratados na literatura especializada: o Problema do Caixeiro Viajante. Este problema em sua forma clássica consiste em, dado um conjunto de N cidades ligadas entre si, um determinado caixeiro-viajante dejesa sair de uma dessas cidades e visitar todas as demais cidades do conjunto, uma única vez, e voltar a cidade de origem, ao custo menor possível.

Como o ponto de início é arbitrário, existem $(N - 1)!$ possíveis soluções. Supondo um computador capaz de listar todas as possíveis soluções do Problema do Caixeiro Viajante para 20 cidades em 1 hora. Este mesmo computador, seguindo a mesma fórmula utilizada para listar todas as soluções possíveis para o problema com

20 cidades, levará 20 horas para listar as soluções para o problema com 21 cidades; 17.5 dias para o problema com 22 cidades; e aproximadamente 6 séculos para o problema com 25 cidades (Reeves, 1996).

Acreditou-se, entretanto, que a evolução do *hardware* e do *software* das máquinas possibilitaria que algoritmos tradicionais como a enumeração completa, particionar e limitar, *branch-and-bound*, programação dinâmica e outros, resolvessem estes problemas sem dificuldades (Reeves, 1996). Porém, isso não aconteceu.

Em verdade, o desenvolvimento do conceito de complexidade computacional (Goodrich e Tamassia, 2002) acabou por demonstrar que existem problemas para os quais encontrar um algoritmo capaz de resolvê-los, de maneira exata, é uma tarefa bastante difícil (Goldbarg e Luna, 2005).

A complexidade de um algoritmo mede o esforço computacional dispendido para a execução da tarefa descrita pelo algoritmo analisado (Goodrich e Tamassia, 2002). Para problemas em que este esforço pode ser verificado através de uma função polinomial de baixo grau, a resolução do mesmo não demanda muitos ciclos de processamento por parte da máquina, podendo ser obtida em pouco tempo. Estes problemas são categorizados como do tipo P (Campello e Maculan, 1994).

Porém, existem problemas em que a complexidade do algoritmo é dada por uma função exponencial. Nestes casos, a medida em que a instância do problema cresce linearmente, o custo computacional cresce exponencialmente. No exemplo do Problema do Caixeiro Viajante referenciado anteriormente, um aumento pequeno no tamanho do problema (exemplificado pelo aumento no número de cidades) faz o tempo de execução de um algoritmo de enumeração completa aumentar significativamente.

Para um exemplo do impacto do aumento no tempo de execução dado um aumento pequeno na instância do problema, supõe-se um problema cujo número de

soluções viáveis é *n!*, onde *n* é o tamanho da instância do problema. Admite-se uma máquina que consegue listar uma solução viável em 10^{-9} segundos (1 nanossegundo). Logo, uma instância de tamanho 20 terá suas soluções listadas em 80 anos; caso a instância seja 21, o tempo aumentará para 1680 anos (Campello e Maculan, 1994).

O tempo computacional para resolução de um problema desse tipo é muito grande, de acordo com o tamanho do problema; a grande quantidade de combinações entre as variáveis de decisão ocasionam um espaço de busca bastante irregular, e a busca nesse espaço é de alto custo computacional, tanto em memória quanto em processamento. Problemas desse tipo são conhecidos como Problemas NP – Completo (Goodrich e Tamassia, 2002).

No próximo tópico, iremos nos aprofundar em alguns importantes problemas de Otimização Combinatória que pertencem a classe de problemas NP - Completo, citando seus conceitos e aplicações em cenários reais.

2.0 – Exemplos de Problemas de Otimização Combinatória do tipo NP - Completo

Visto a teoria de problemas de Otimização Combinatória e um pouco da teoria de complexidade de algoritmos, lista-se agora quatro exemplos de problemas NP – Completo.

Os quatro problemas apresentados são: a) Problema do Caixeiro-Viajante; b) Problema da Mochila; c) Problema da Árvore de Steiner; d) Problema de *Job Shop*. Eles representam problemas clássicos da literatura sobre problemas de Otimização Combinatória NP, e a modelagem de algumas situações encontradas em organizações levam a problemas desse tipo (Goldbarg e Luna, 2005). Em cada subseção, descreve-se o conceito do problema e algumas dessas aplicações práticas.

2.1 - Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante é um dos mais tradicionais e conhecidos problemas de programação matemática (Goldbarg e Luna, 2005). Trata-se de, dado um conjunto de *cidades* e um conjunto de *estradas* entre as mesmas, encontrar um percurso mínimo no qual um caixeiro viajante possa, a partir de uma cidade, passar por todas as outras apenas uma vez e retornar a cidade de origem.

Utilizando conceitos de teoria dos grafos, pode-se modelar o Problema do Caixeiro Viajante através da representação dos elementos do problema como componentes de um grafo: as cidades seriam representadas pelos nós e as estradas pelas arestas. A Figura 2.1.1 apresenta este tipo de formulação para o problema. A solução então seria encontrar um ciclo hamiltoniano de custo mínimo para o grafo dado (Arenales, Armentano, Morabito e Yanasse, 2007).

A importância do Problema do Caixeiro Viajante é devida a, pelo menos, três características (Goldbarg e Luna, 2005): a) grande aplicação prática; b) grande relação com outros modelos e c) grande dificuldade de resolução exata. Já fora comentado, neste trabalho, a publicação de Reeves que trata do assunto quanto a demanda de tempo para a aplicação de um algoritmo de enumeração completa a este tipo de problema, com diferentes números de cidades (Reeves, 1996).

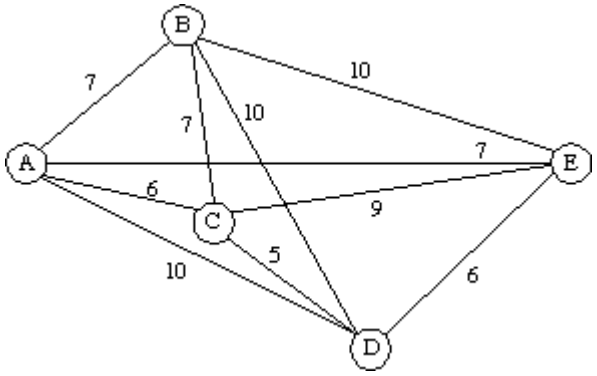


Figura 2.1.1: Uma representação em grafo do Problema do Caixeiro Viajante.

Entre algumas aplicações para o problema (Goldbarg e Luna, 2005), pode-se citar:

- Programação de operações de máquinas em manufaturas;
- Programação de transporte entre células de manufatura;
- Otimização do movimento de ferramentas de corte;
- Problemas de roteamentos de veículos;
- Otimização de perfurações de furos em placas de circuito impresso.

2.2 - Problema da Mochila

Para o Problema da Mochila, primeiro tem-se um conjunto de elementos com diferentes pesos e valores associados. Dada uma certa capacidade máxima que a mochila consegue suportar, alocar elementos a ela de forma que a somatória do valor dos elementos alocados seja o maior possível (Arenales, Armentano, Morabito e Yanasse, 2007).

Esse talvez seja um dos problemas de otimização em que mais facilmente pode-se perceber a diferença de dificuldade de solução entre suas várias versões (Goldbarg e Luna, 2005). No caso do Problema da Mochila em que um item pode ser "quebrado" - seu valor e seu peso podem ser fracionados, ou seja, podem assumir um valor contínuo -, o problema pode ser solucionado de modo eficiente em tempo polinomial, utilizando o método *simplex*.

Já no caso do Problema da Mochila 0-1, como conceituado no início desse tópico, métodos de resolução exata conseguem resolvê-lo apenas para problemas de magnitude baixa ou mediana. Este problema é NP - Completo (Goldbarg e Luna, 2005).

Algumas aplicações típicas para o Problema da Mochila são (Goldbarg e Luna, 2005):

- Investimento de capital;
- Problemas de corte e empacotamento;
- Carregamento de veículos;
- Orçamento.

2.3 - Problema da Árvore de Steiner

O Problema da Árvore de Steiner consiste em, dado um grafo não-direcionado $G = (V, E)$, onde o custo das arestas é sempre positivo ou zero ($c: E \rightarrow R+$) para qualquer aresta pertencente a E, e um subconjunto de vértices do grafo G, que chamaremos de S (ou seja, $S \subseteq V$), devemos encontrar um subgrafo $G' = (V', E')$ tal que (Kapsalis, Rayward-Smith e Smith, 1993):

- V' contém todos os vértices de S;
- G' é conexo; e
- $\sum c(e) \ \{ \ e \in E' \ \}$ é mínimo.

Ou seja, o subgrafo G' conterá todos os nós de S, será conexo e a somatória das arestas que o formam deverá ser a menor possível. O subgrafo G' poderá conter vértices que não façam parte de S. A inclusão desse tipo de vértice, denominados Vértices de Steiner, normalmente são necessárias para a conexidade da solução e, dependendo daqueles que forem colocados, poderão reduzir ou não o custo total das arestas de G' (Kapsalis, Rayward-Smith e Smith, 1993).

Percebe-se então que a restrição do Problema da Árvore de Steiner não é a necessidade de que os vértices de G' sejam formados apenas pelo conjunto S, nem que o número de arestas de G' seja o mínimo possível. O problema requer a conexidade da solução e a necessidade de que o custo dessa conexidade seja o mínimo possível, podendo para este fim serem utilizados vértices que não pertençam a S.

Exemplificando, considere o grafo da Figura 2.3.1. (a) indica o grafo original, onde pode-se identificar os vértices do conjunto S (também chamados de Vértices Especiais) marcados com um círculo ao redor. São eles v1, v3, v7 e v9. (b) mostra visualmente como seria a resolução de um Problema da Árvore de Steiner sobre este grafo, onde os Vértices de Steiner estão indicados por retângulos; são eles v6 e v4. Esta solução determina o menor custo possível para se “ligar” os Vértices Especiais,

que no caso usará os dois já citados Vértices de Steiner (Costa e Saraiva, 2008).

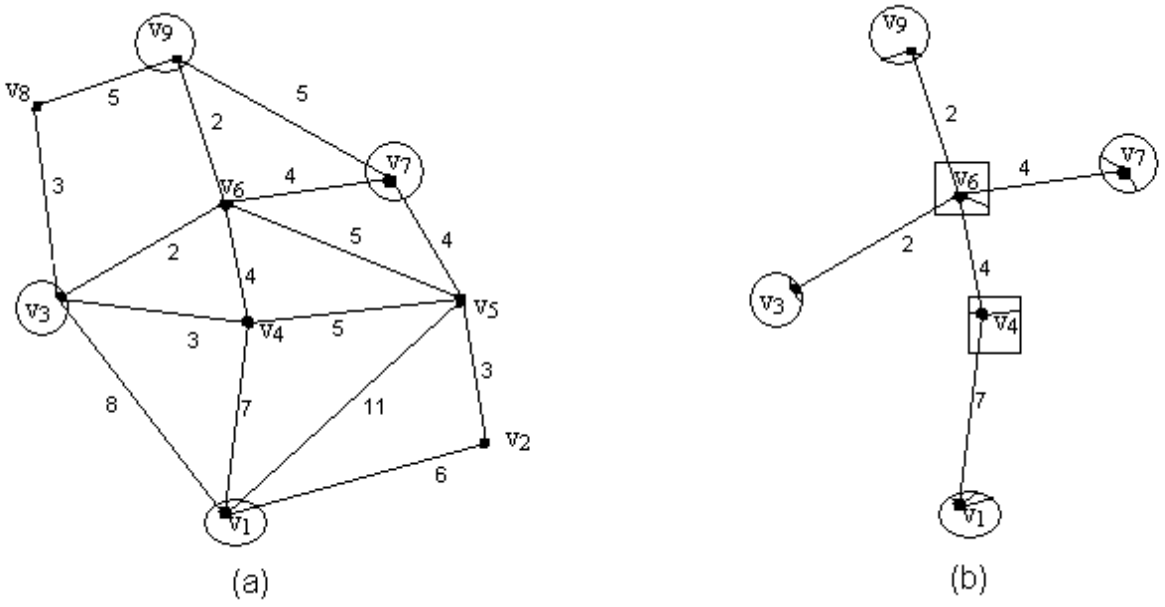


Figura 2.3.1: Um grafo apresentando um problema da Árvore de Steiner.

Algumas das aplicações para o Problema da Árvore de Steiner são (Goldbarg e Luna, 2005):

- Projeto de circuitos eletrônicos;
- Redes de comunicação;
- Planejamento de redes externas de comunicação;
- Tubulação de gás e óleo;
- Projeto de instalações elétricas e mecânicas.

2.4 - Problema de *Job Shop*

O Problema de Programação de Tarefas em ambiente *Job Shop*, pode ser definido por um conjunto de *M* máquinas e *N* tarefas, onde cada tarefa consiste de uma sequência ordenada de *M* operações. Cada operação deve ser processada em uma única máquina por um determinado tempo, sem interrupção, e uma máquina pode processar somente uma operação de cada vez. As operações de uma mesma tarefa devem ser executadas em máquinas diferentes, e cada tarefa possui uma ordem particular de processamento.

O problema consiste em programar as operações das tarefas nas máquinas, de forma a otimizar alguma medida de desempenho (Baker, 1974):

- *Makespan* – completar todas as tarefas no menor tempo. A medida de tempo entre o início da primeira e a conclusão da última tarefa é chamada de *makespan*;

- Atraso total – em alguns cenários, existe uma data de entrega das tarefas. Porém, nem sempre há a possibilidade de todas as tarefas serem completadas sem incorrer em atraso para algumas destas. Nesse caso, o problema tenta minimizar o atraso total das tarefas;

- Número de tarefas atrasadas – como em Atraso Total, algumas vezes não dá para evitar que algumas tarefas não sofram atraso. Problemas de *Job Shop* na categoria número de tarefas atrasadas, tentam minimizar a quantidade de tarefas que não serão cumpridas na data de entrega.

O Problema de *Job Shop* tem como suas principais aplicações aquelas votadas para a programação de diferentes tarefas a serem realizadas em uma fábrica ou organização, onde normalmente os recursos necessários para a finalização de uma operação devem ser compartilhados para a execução das demais operações. Podemos então citar alguns exemplos, tais como:

- Programação de entrega de materiais gráficos em uma gráfica;
- Programação da produção de diferentes tipos de embalagens;
- Programação do empacotamento de produtos em uma empresa.

Agora que foram apresentados alguns exemplos de problemas de Otimização Combinatória do tipo NP – Completo e suas aplicações, e lembrando da dificuldade em se desenvolver um método de resolução exata para problemas desse tipo, podemos nos perguntar como os pesquisadores se debruçam sobre essa questão, visto a importância prática que a resolução destes problemas tem para diversos setores de empresas em geral. Veremos no próximo tópico o conceito de Heurística e Metaheurística, e como a linha de pesquisa baseada nestes ramos de resolução tem contribuído para a obtenção de boas soluções para problemas NP – Completo.

3.0 - Heurísticas e Metaheurísticas

Como já colocado, para problemas de Otimização Combinatória do tipo NP – Completo, métodos de resolução exatos baseados, por exemplo, em enumeração completa, particionar e limitar, programação dinâmica e outros, não conseguem solucionar o problema proposto em tempo razoável, devido à "explosão combinatória" e sua respectiva consequência no aumento do custo computacional em gasto de memória e ciclos de processamento.

Nos primeiros anos da Pesquisa Operacional, a ênfase dos pesquisadores foi em encontrar as soluções exatas para os problemas propostos (Reeves, 1996). Por conta disso, vários algoritmos exatos com boa performance para alguns grupos de problemas foram desenvolvidos, como o *simplex* e o *branch-and-bound*, a programação dinâmica. Infelizmente, mesmo esses métodos sendo mais "elegantes" que a enumeração completa, não foram eficientes para resolver uma instância do Problema do Caixeiro Viajante com magnitude mediana – alta.

Por conta disso, muitos pesquisadores nos anos 60 se viram as voltas com o questionamento: existe um algoritmo de otimização "polinomial" para solucionar o Problema do Caixeiro Viajante? Karp, em 1972, demonstrou que, se a resposta for "sim", então existirá um algoritmo polinomial para todos os outros problemas NP – Completo. Como, desde então, nenhum algoritmo foi desenvolvido, é fortemente sugerido que a resposta para essa questão seja "não". Porém, a verdadeira resposta ainda é desconhecida (Reeves, 1996).

Utilizando-se de outra abordagem, grupos de pesquisadores desenvolveram o conceito de heurísticas para aplicarem à resolução de problemas NP - Completo. Pela definição, heurísticas são métodos de resolução de problemas que não garantem que encontrarão a melhor solução para o problema dado, porém se for uma heurística bem

planejada, encontrará uma boa solução suficiente para ser aceita como resposta ao modelo do problema, a um custo computacional aceitável e em tempo hábil (Norvig e Russel, 2004).

Temos como exemplo de métodos heurísticos alguns algoritmos como Descida/Subida de Encosta, Método Guloso, A*, entre outros (Norvig e Russel, 2004). O algoritmo Descida/Subida de Encosta é uma das mais tradicionais e antigas heurísticas desenvolvidas. Consiste em, dado um conjunto de valores assumidos pelas variáveis de decisão e valor correspondente para a função objetivo, pesquisar a vizinhança dessa solução e comparar os valores para a função objetivo em cada vizinho. Se existe um vizinho que melhore o valor para a função objetivo dada, descartar a atual solução e assumir essa nova, repetindo então o procedimento de avaliação da vizinhança. Caso não exista um vizinho que melhore a função objetivo, assumir a atual configuração das variáveis de decisão como solução para o problema (Norvig e Russel, 2004).

Apesar da busca pela solução exata não ser o objetivo primordial de uma heurística, as abordagens heurísticas modernas encontram boas soluções, senão as próprias soluções ótimas, para as instâncias do problema dado.

Porém, nem sempre o uso de heurísticas garante, apenas por si, que boas soluções serão encontradas. Como o espaço de busca de um problema NP - Completo é particularmente irregular, é provável que métodos heurísticos se satisfaçam com ótimos locais, que apesar de serem boas soluções para sua vizinhança específica, podem estar muito distantes, em termos qualitativos, das soluções ótimas globais. O algoritmo Subida/Descida de Encosta, explicado anteriormente, tem muitas possibilidades de encontrar um ótimo local de baixo desempenho e assumi-lo como solução para o problema, não realizando uma busca mais diversificada no espaço de busca de soluções. A Figura 3.1 apresenta um espaço de busca irregular e um círculo negro, representando a busca de uma heurística. Nela, o círculo está em um ótimo

local, porém, qualitativamente, ainda está distante do ótimo global.

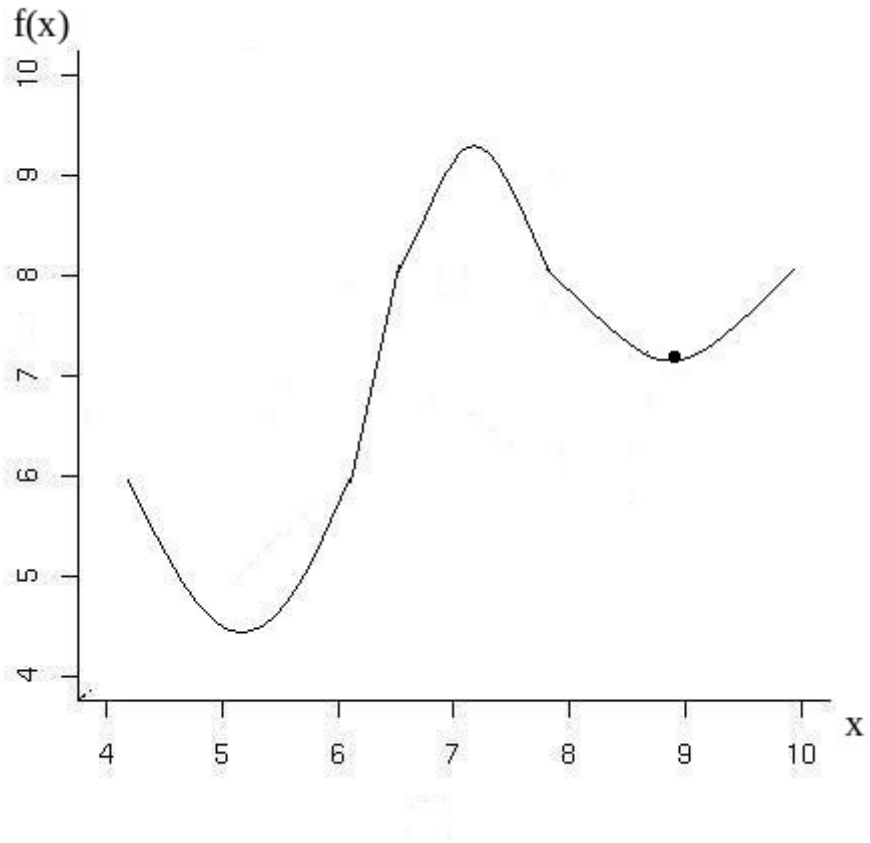


Figura 3.1: Um espaço de busca irregular.

Nesse caso, o uso de metaheurísticas, conceituadas como definições de regras e políticas que guiarão a busca heurística, tem obtido melhores resultados em diversas pesquisas (Cordenonsi, Muller e Bastos, 2005). O uso de metaheurísticas garante que, mesmo que a busca heurística chegue a um ótimo local, a busca optará por uma ação e continuará seguindo as regras adotadas, garantindo assim uma maior exploração do espaço de busca e aumentando a probabilidade do método encontrar uma boa solução (Glover e Kochenberger, 2003).

A literatura traz vários tipos de metaheurísticas, muitas delas baseadas em observações de processos naturais. Por exemplo, temos os Algoritmos Genéticos, baseados na dinâmica populacional das espécies (Coley, 1999); Colônia de Formigas,

baseado no processo de busca de comida pelas formigas de um formigueiro; Busca Tabu, que faz referência ao processo de memorização humana (Reeves, 1996); e o Recozimento Simulado, inspirado no resfriamento e aquecimento de materiais físicos (Reeves, 1996).

Como parte do escopo deste trabalho, conceituaremos quatro metaheurísticas, suas principais características e em quais problemas foram utilizadas, segundo relatos na bibliografia consultada. As metaheurísticas expostas a seguir são Algoritmos Genéticos, Busca Tabu , Colônia de Formigas e GRASP.

4.0 - Conceitos das Metaheurísticas

4.1 - Algoritmos Genéticos

Algoritmos Genéticos estão entre as metaheurísticas mais estudadas e desenvolvidas nas últimas décadas (Coley, 1999). Criada por John Holland na década de 70, apresentada em sua tese *Adaptation in Natural and Artificial Systems*, baseia-se na dinâmica natural das espécies para fazer evoluir um conjunto de soluções para um determinado problema, afim de, a cada geração, as soluções encontradas vão sendo melhor adaptadas – tem um desempenho melhor na função objetivo – para o ambiente em que se encontra – as restrições do problema (Costa e Saraiva, 2008). Sua concepção lembra bastante a teoria darwinista de evolução (Coley, 1999).

AG's são implementados como uma simulação de computador em que uma população de representações abstratas de soluções para um determinado problema passam por um processo evolutivo, como o que ocorre na natureza, onde melhores soluções são buscadas a cada geração da população.

A representação de soluções em um AG é feita através de uma *string* de *bits*, chamada de cromossomo, onde cada *bit* é comumente chamado de gene. Existem vários tipos de codificação, desde cromossomos de inteiros, números reais e variáveis binárias (Linden, 2006). Esse tipo de modelagem facilita os processos de cruzamento (*crossover)* e mutação, principais operadores no processo de evolução desenvolvido pelo método (Coley, 1999).

Podemos citar, de forma a exemplificar a codificação em algoritmos genéticos, duas modelagens para o Problema da Mochila. No exemplo, o problema será com 5 itens. Em uma delas, teremos a codificação binária, onde o valor “0” identifica que um item não está na mochila; já o valor “1” significa que aquele item está presente na mochila. A outra codificação indicará a ordem de preferência dos itens na mochila:

avaliando a *string* que representa o cromossomo, da esquerda para a direita, teremos a indicação dos itens que entrarão na mochila, respeitando a restrição de capacidade, como mostra a Figura 4.1.1.

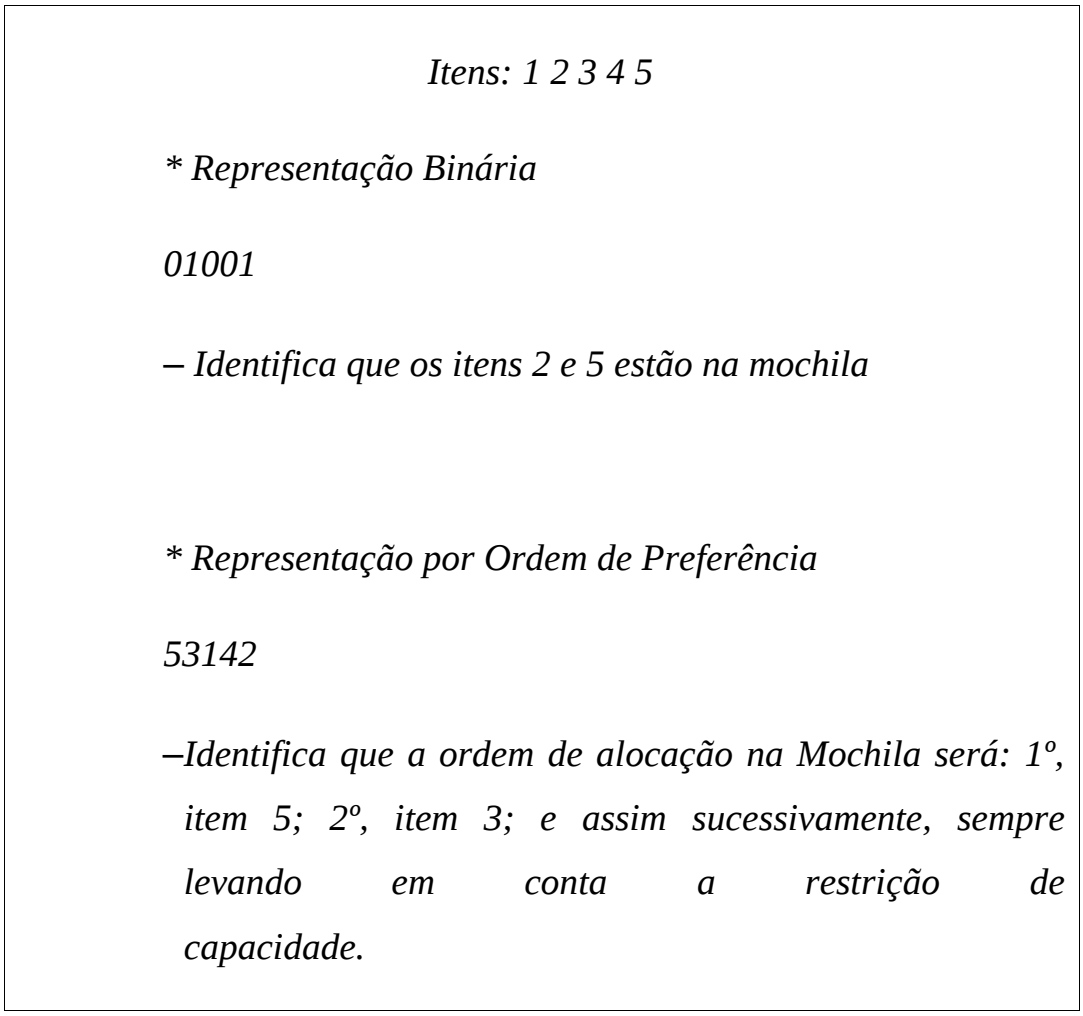


Figura 4.1.1: Dois exemplos de Codificação para o Problema da Mochila

A forma como se dará a codificação é de fundamental importância para o desempenho do algoritmo genético. A codificação indica de que maneira uma solução será representada, e isso permite executar a busca no espaço codificado de soluções, que nem sempre corresponde ao espaço de busca original, impedindo ou não que soluções inviáveis sejam representadas pela população. O projetista de algoritmo genético deve ponderar se a representação de soluções inviáveis é

interessante para uma busca mais diversificada, ou se isso acabará por causar um gasto excessivo e não justificado em ciclo de processamento e memória (Linden, 2006).

Dada uma população de soluções em um AG, faz-se um processo de seleção entre suas representações para escolher quais delas passarão pelo processo de cruzamento, onde seus *bits* serão permutados entre si, a partir de uma determinada regra.

A etapa da seleção tenta mimetizar o processo de Seleção Natural, como descrito por Darwin em sua obra *A Origem das Espécies*. Para o naturalista inglês, indivíduos melhor adaptados ao seu ambiente teríam maiores chances de procriar – ou seja, de passar seus genes para gerações futuras. Isso possibilita a evolução das espécies, onde a cada geração espera-se que os indivíduos estejam melhor adaptados as condições do ambiente.

Em algoritmos genéticos, um dos métodos mais utilizados de seleção chama-se Método da Roleta. Os indivíduos de uma geração são representados em uma "roleta", com uma fatia proporcional a seu valor na função objetivo. Ou seja, aqueles melhor adaptados terão uma representação maior no mecanismo de seleção. Gera-se então um número aleatório entre [0, $\sum Z(x_i)$], onde Z é o valor da função objetivo de todos os indivíduos da população. Esse número indicará qual indivíduo foi selecionado nessa rodada; faz-se então o número de rodadas necessárias para selecionar os indivíduos que irão cruzar naquela iteração (Rezende, 2003).

A próxima etapa será o cruzamento, ou *crossover*, entre os indivíduos selecionados na etapa anterior. A idéia é que os bons genes da população possam se combinar, formando indivíduos melhores adaptados ao ambiente – ou seja, que tenham um melhor desempenho na função objetivo. Existem três tipos mais conhecidos de *crossover:* de um ponto, multiponto e máscara (Rezende, 2003).

O *crossover* de um ponto determina um gene a partir do qual as informações genéticas dos pais serão trocadas. As informações anteriores a este ponto em um dos pais serão ligadas às informações posteriores a este ponto no outro pai (Carvalho, Braga e Ludermir, 2003), como mostra a Figura 4.1.2.

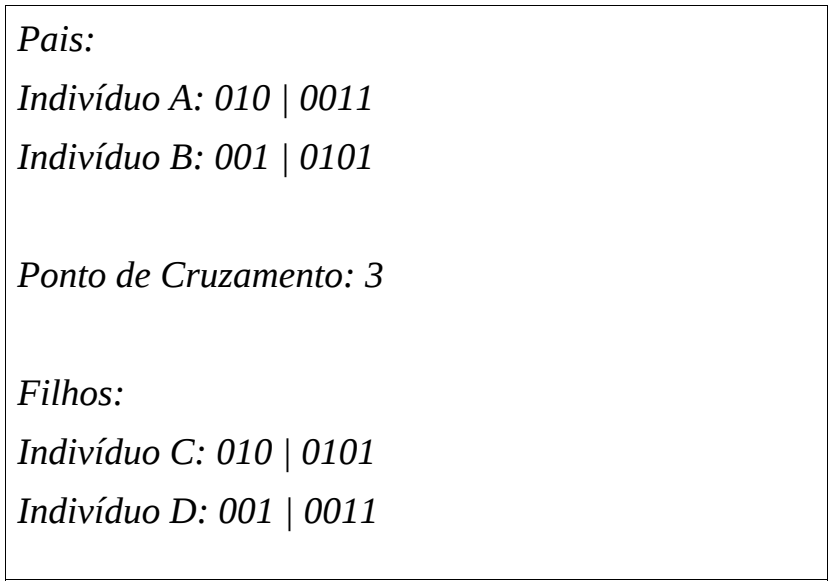


Figura 4.1.2: Exemplo de crossover de 1 ponto.

O cruzamento multiponto é uma generalização da idéia de troca de material genético através de pontos, em que vários pontos de cruzamento podem ser utilizados. Na Figura 4.1.3 podemos ver o funcionamento deste método para o *crossover* de 2 pontos (Carvalho, Braga e Ludermir, 2003).

Pais:

Indivíduo A: 010 | 00 | 11

Indivíduo B: 001 | 01 | 01

Pontos de Cruzamento: 3 e 5

Filhos:

Indivíduo C: 010 | 01 | 11

Indivíduo D: 001 | 00 | 01

Figura 4.1.3: Exemplo de crossover multiponto – 2 pontos

O cruzamento máscara determina, por meio de uma representação a parte, quais os genes que serão trocados entre os pais, de forma a não utilizar o conceito de intervalos, existentes nos outros dois métodos (Carvalho, Braga e Ludermir, 2003) (Figura 4.1.4).

Pais:

Indivíduo A: 0 1 0 0 0 1 1

Indivíduo B: 0 0 1 0 1 0 1

Máscara (valores 0 virão do Indivíduo A; valores 1 virão do indivíduo B. Inverte-se para o próximo indivíduo): 0 1 0 1 0 0 0

Filhos:

Indivíduo C: 0 1 1 0 1 0 1

Indivíduo D: 0 0 0 0 0 1 1

Figura 4.1.4: Exemplo de crossover máscara.

Em seguida o processo de mutação inicia-se. Uma das formas mais utilizadas, existe uma probabilidade de determinado gene do cromossomo ser permutado para outro valor aceitável para aquela codificação. O operador de mutação serve para colocar na população aqueles genes que, porventura, se perderam durante o processo de cruzamento e formação de gerações, tornando a busca mais diversificada. Normalmente, utilizam-se baixos valores para mutação (entre 0.01 e 0.05) (Coley, 1999).

A partir dessas fases têm-se novas soluções, que farão parte da próxima geração do problema, repetindo o ciclo até um número de gerações pré-estabelecido ou outro critério de parada. Na Figura 4.1.5 pode-se ver o fluxograma de funcionamento de um Algoritmo Genético.

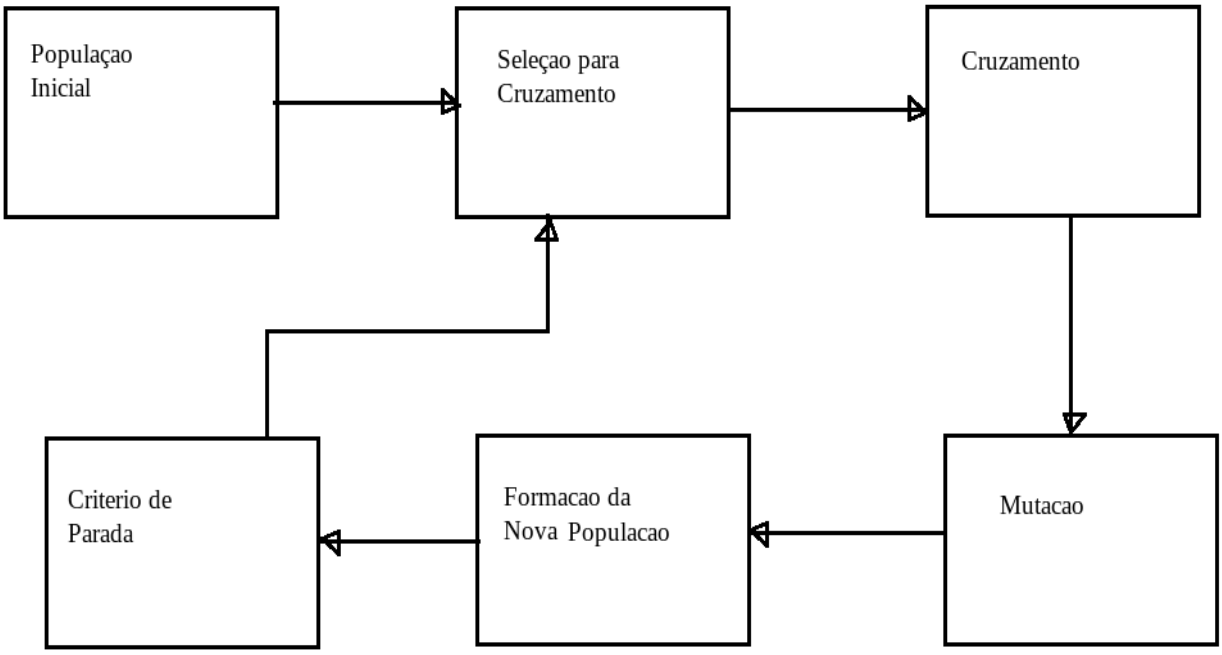


Figura 4.1.5: O Fluxograma das etapas de um Algoritmo Genético

É imprescindível que os valores do operador de cruzamento e mutação sejam

bem avaliados e determinados. Também, o número de gerações e o tamanho da população inicial devem ser bem selecionados, pois a eficiência do Algoritmo Genético dependerá dos valores determinados. Normalmente, cada problema tem seus valores que otimizam o desempenho do algoritmo. Para tanto, aqueles que forem implementar esta heurística, devem conhecer razoavelmente o problema que se propõem a resolver, fazendo diferentes simulações, alterando os valores dos parâmetros e procurar aqueles que melhor atenderem ao problema (Kapsalis, Rayward-Smith e Smith, 1993).

4.2 - Busca Tabu

A heurística denominada Busca Tabu se baseia no conceito de Busca Local (Norvig e Russel, 2004), e também possui recursos que visam impedir a busca de ficar presa em uma solução ótimo local. Desenvolvida por Glover, em 1986, ela tem como analogia o modelo de memorização humana (Reeves, 1996).

A implementação da "memória" é dada pelo armazenamento de soluções previamente encontradas usando uma simples estrutura de dados. Na criação dessa lista – chamada Lista Tabu - de exploração em vizinhanças visitadas recentemente, evita-se que a busca entre em "ciclos", onde a Busca Local ficaria restrita a um subespaço do espaço de busca. Evitando a ciclagem, haverá uma maior diversidade no espaço percorrido pela busca, e a probabilidade de se encontrar uma boa solução será maior (Reeves, 1996).

Dado uma solução x, a heurística analisa a vizinhança de x (dada por $N(x)$) e procura por uma possível ocorrência de alguma dessas soluções na sua lista tabu (sua "memória"). Caso encontre alguma, essa solução não estará no conjunto das possíveis soluções que o método avaliará na atual iteração.

As diversas etapas características ao método são descritas em termos de quatro

parâmetros na Busca Tabu: recenticidade, frequência, qualidade e influencia (Reeves, 1996).

- Recenticidade: como o objetivo da Busca Tabu é encorajar a exploração de partes do espaço de solução ainda não explorados, pode-se acabar limitando demais o espaço de busca pela proibição de se voltar a soluções já encontradas. Dessa forma, a proibição de se percorrer uma solução já encontrada é feita apenas para as mais recentes (ou seja, aquelas que estejam dentro de um certo limite de interações anteriores). Recenticidade identifica o número de iterações em que cada solução ficará armazenada na lista tabu.

- Frequência: recenticidade simula a memória a curto prazo; a “memória de longo prazo” pode ser implementada pelo uso de Frequências. O objetivo é identificar o número de vezes que uma determinada característica foi encontrada em uma solução, como a presença de um determinado atributo, por exemplo. Frequências são utilizadas para gerar penalidades na função objetivo, encorajando a geração de soluções diferentes daquelas que foram encontradas significativamente ao longo da busca, tornando a exploração mais diversa.

- Influência: a idéia é permitir que um movimento caracterizado como tabu possa ser realizado para permitir uma modificação para um ótimo local, ou seja, que o mesmo leve a uma solução melhor que a atual. Quando um movimento é considerado influente, significa que uma vez feito irá modificar substancialmente a solução – isto é, leva a um nível além dos oferecidos pelos movimentos não-tabu.

- Qualidade: é uma medida da maneira como um movimento particular contribui na busca de um ótimo global. Refere-se a habilidade de diferenciar o mérito de soluções visitadas durante a busca. Pode ser usada para identificar elementos que são comuns a boas soluções ou a caminhos que levam a tais soluções.

Deve-se colocar incentivos para reforçar ações que levam a boas soluções e penalidades para desencorajar ações que levem a soluções ruins.

A Busca Tabu também trabalha com o conceito de memória de longo prazo. Essa etapa consiste em salvar as melhores soluções encontradas durante a busca – chamadas Soluções de Elite (Reeves, 1996). Quando a Busca Tabu completa uma iteração, pode-se recomeçar a busca a partir de um desses ótimos locais encontrados.

Assim, impedindo que a busca siga por um caminho já percorrido, pode-se fazer uma melhor exploração do espaço de busca, diversificando-a e aumentando as chances de se conseguir uma boa solução para o problema dado.

4.3 – Colônia de Formigas

O algoritmo de otimização por Colônia de Formigas implementa os conceitos de Inteligência de Enxame. Esta abordagem estabelece um ambiente onde vários agentes interagem entre si através de uma série de regras pré-estabelecidas. A partir desta interação, os agentes podem adquirir experiência e passar a desempenhar uma determinada função de forma cooperada e adaptativa, otimizando o desempenho da realização da mesma (Von Zubben e Attux, s/d).

O algoritmo Colônia de Formigas foi proposto por Dorigo, como uma nova metaheurística para ser aplicada a problemas de otimização combinatória. O algoritmo foi baseado na dinâmica de uma colônia de formigas e a maneira como elas procedem para buscar comida (Dorigo, Maniezzo e Corloni, 1996).

A primeira implementação da metaheurística foi aplicada ao Problema do Caixeiro Viajante. Em seguida, vários pesquisadores utilizaram este algoritmo para outros problemas de otimização combinatória, como Roteamento de Veículos, Minimax, Coloração de Grafos, Mochila Múltipla e outros. Estas implementações

evoluíram o método, tornando-o mais robusto e de propósito geral (Dorigo, Maniezzo e Corloni, 1996).

Como uma metaheurística, a Colônia de Formigas tenta otimizar uma função objetivo através de uma busca no espaço de soluções a um custo computacional razoável. Este algoritmo implementa um conjunto de agentes, chamados “formigas”, para coletivamente atuarem sobre o problema formulado para uma representação em grafo.

A informação obtida pelas formigas através da busca é guardada em uma “trilha de feromônios” associada aos nós do grafo. Esta trilha representa a memória de “longo prazo” da busca (Dorigo, Maniezzo e Corloni, 1996).

A formiga possui memória para guardar o caminho até o momento. Isso significa que a formiga vai construindo a solução aos poucos, a medida que vai visitando os nós do grafo. Ela poderá utilizar essa informação para procurar soluções viáveis, melhorar soluções previamente encontradas e retroceder no caminho.

A formiga também deve respeitar a vizinhança dos nós do grafo que foi modelado para o problema. Cada nó só poderá ser alcançado a partir do nó em que a formiga se encontra em dada iteração.

Uma formiga “decide” ir de um nó a outro nó vizinho a partir de uma função probabilística relacionada com a trilha de feromônios, a memória da formiga e a as restrições do problema (Dorigo e Stutzle, 2003).

A função probabilística tem por objetivo aumentar as chances de uma formiga passar por um arco que esteja otimizando a função objetivo, em detrimento de outros. Isso visa garantir que as formigas vão convergindo para uma solução ótimo local ao problema.

Porém, ainda existe a probabilidade de uma formiga escolher caminhar por outro arco do grafo, que não aquele onde a trilha de feromônio seja maior. Isso possibilita uma maior diversidade na busca, além de fazer com que o método consiga escapar de ótimos locais e reduzir a convergência apressada do algoritmo, possibilitando uma busca mais robusta pelo espaço de soluções.

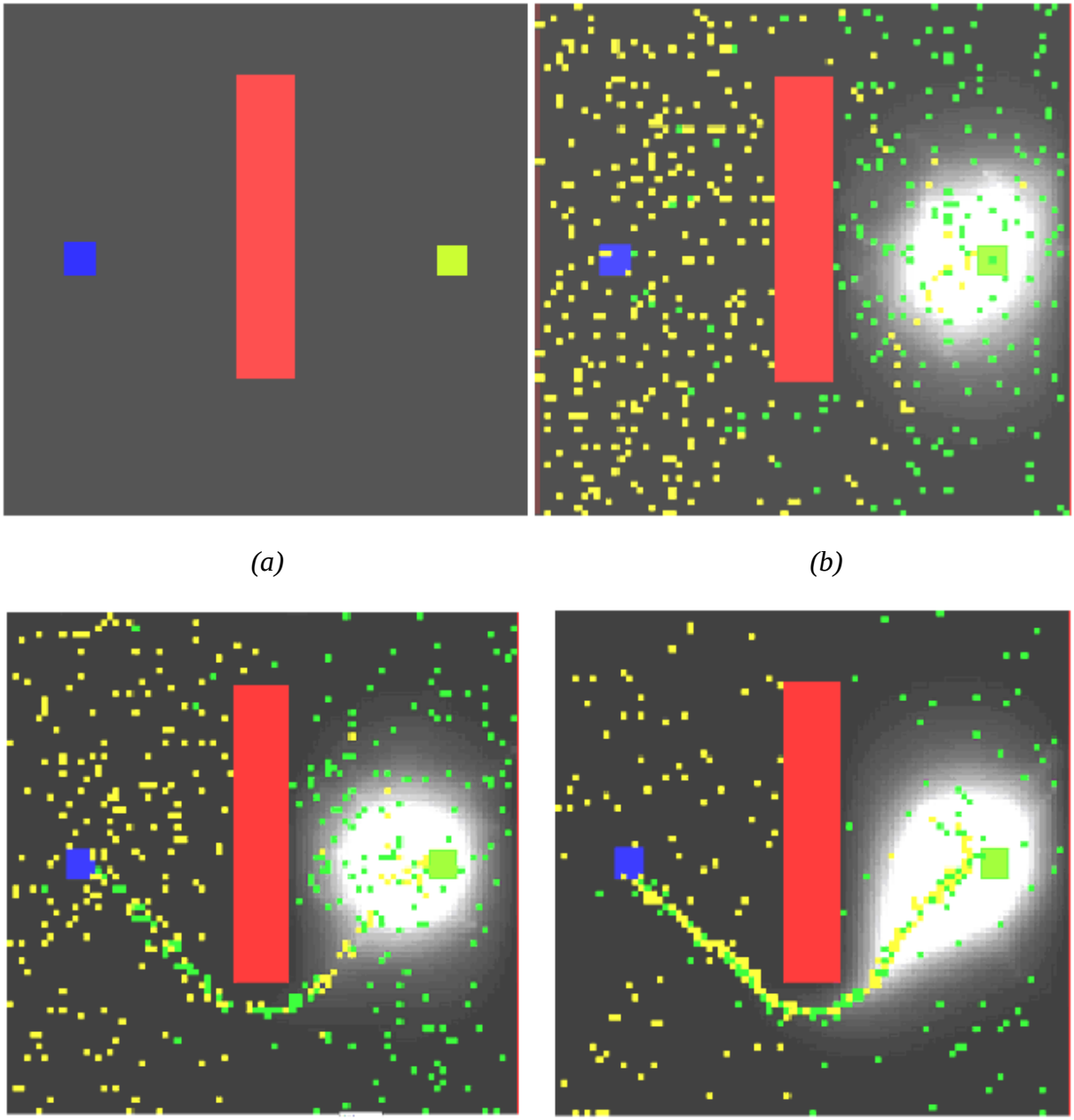
Após a formiga ter construído uma solução, ela retorna no grafo e deposita feromônios em algum arco, aumentando a trilha de feromônios para aquele caminho. Assim, a formiga cumpre sua função e é descartada pelo método, que criará um outro agente em substituição.

A partir destas interações, as soluções vão surgindo e vão sendo otimizadas pelo algoritmo. As formigas trocam informação entre si através da trilha de feromônio alocada na estrutura do grafo, que é adaptativamente modificado a partir dos caminhos que as formigas traçam por ele (Dorigo e Stutzle, 2003).

Uma maneira de impedir a convergência muito rápida do algoritmo é declarar uma função específica que “evapora” a trilha de feromônios de acordo com o tempo.

Na Figura 4.3.1 (Von Zubben e Attux, s/d) pode-se perceber a convergência da Colônia de Formigas. As figuras apresentam um ambiente em duas dimensões onde: (a) tem-se um ponto azul que representa a entrada da colônia, um obstáculo em vermelho e um ponto amarelo indicando a fonte de alimento. As formigas devem encontrar o caminho mínimo que as levem da colônia até a fonte de alimento e de volta para o ponto azul. Em (b), as primeiras iterações do método onde a busca encontra-se dispersa pelo espaço.

Após algumas iterações, a Figura 4.3.1 (c) apresenta o começo da convergência do algoritmo. O brilho branco significa a trilha de feromônio deixada no caminho onde as formigas estão convergindo. Já em (d), temos uma convergência bem perceptível e algumas poucas formigas dispersas procurando caminhos alternativos.



(a) *(b)*

(c) *(d)*

Figura 4.3.1 – Iterações da Colônia de Formigas em tempos diferentes.

As principais aplicações de Colônia de Formigas se dão em problemas de otimização combinatória, problemas onde as rotas do grafo são dinâmicas ou problemas onde a arquitetura computacional é paralela (Dorigo, Maniezzo e Corloni, 1996).

4.4 – GRASP

Greedy Randomized Adaptative Search Procedure – Procedimento de Busca Gulosa, Adaptativa e Randômica – é um método de busca metaheurística primeiramente desenvolvido por Feo e Resende. GRASP é usualmente implementado como um procedimento de que constrói uma solução para, em seguida, fazer uma busca local sobre a mesma afim de otimizá-la. Em seguida, o algoritmo faz outro início e repete o procedimento, até um dado número de iterações (Resende e Ribeiro, 2003).

Cada solução encontrada é salva em uma lista de soluções onde, ao final do número de iterações, aquela que melhor otimiza a função objetivo é selecionada como resposta pelo método.

Assim, pode-se perceber duas etapas diferentes para cada iteração. A primeira trata da construção da solução inicial. A segunda, da busca local empregada afim de otimizar a solução (Resende e Ribeiro, 2003).

Na etapa de construção, a solução é construída iterativamente, elemento por elemento. Esse método é normalmente heurístico e adaptativo , onde os benefícios associados a cada elemento é atualizado a cada iteração dessa fase, refletindo mudanças ocorridas pela seleção de elementos anteriores (Resende e Ribeiro, 2003).

Todos os elementos candidatos a compor a solução são alocados em uma lista, chamada de lista de candidatos. Esta lista é ordenada por uma função gulosa que mede o benefício que o elemento, caso escolhido, concederá a solução em

construção. Um subconjunto denominado lista restrita de candidatos é formada então pelos melhores elementos que compõem a lista de candidatos.

Em seguida, escolhe-se aleatoriamente um elemento da lista restrita de candidatos que será incluído na solução. No próximo passo, segue-se para a adaptação ou recálculo da função gulosa para os elementos ainda não pertencentes à solução.

A componente probabilística da metaheurística se deve a escolha aleatória de um elemento da lista restrita de candidatos para a construção da solução inicial, em cada fase (Resende e Ribeiro, 2003). Este procedimento permite que diferentes soluções com diferentes qualidades sejam geradas.

Desta forma, o principal parâmetro a ser configurado no GRASP é a cardinalidade da lista restrita de candidatos. Este parâmetro é um dos mais importante para o procedimento GRASP: se a cardinalidade for igual a 1, então a solução inicial é totalmente gulosa, selecionando sempre o melhor elemento da lista de candidatos; ao contrário, se a cardinalidade for igual ao número total de candidatos, então a solução é totalmente aleatória (Resende e Ribeiro, 2003).

Para a aplicação eficaz do método é necessário, portanto, a definição de um intervalo de valores para a cardinalidade da lista restrita de candidatos de forma a balancear a relação entre qualidade das soluções, quantidade de iterações necessárias e vizinhança explorada.

A fase de busca local de GRASP aproveita a solução inicial da fase de construção e explora a vizinhança desta solução. Se um componente que melhore o desempenho da solução é encontrado, então a solução corrente é atualizada e novamente a vizinhança dessa nova solução é analisada. O processo se repete até que nenhum componente que melhore o valor da função possa ser encontrado. Alguns dos cuidados a se ter nesta etapa refere-se a:

- Escolher uma vizinhança apropriada;
- Usar estruturas de dados eficientes para acelerar a busca local;
- Ter uma boa solução inicial.

Após esta etapa, a solução encontrada é salva e uma nova solução inicial é construída, repetindo-se o processo. O algoritmo termina quando o número máximo de iterações é alcançado. Assim, a solução que tem o valor que melhor otimize a função objetivo será dada como a resposta do GRASP ao problema.

Parte II:

Desenvolvimento Prático

5.0 – Algoritmos Genéticos aplicados ao Problema de *Job Shop*

Nos capítulos anteriores foi possível ter uma visão geral do tema Problemas de Otimização Combinatória do tipo NP – Completo e do uso de Metaheurísticas para a resolução desses problemas. Com os conceitos fundamentais de alguns problemas de Otimização Combinatória do tipo NP – Completos e de Metaheurísticas, o próximo passo desse trabalho será a implementação de um dos métodos estudados a um dos problemas propostos.

Acreditamos que a implementação computacional, a avaliação dos parâmetros, a aplicação do programa a diferentes instâncias de um problema – como os listados na OR-Library -, observação dos resultados encontrados e a comparação com outros métodos encontrados na literatura, contribuirão para um maior entendimento do assunto, além de servir como experiência à pesquisa científica.

A partir dos estudos e discussões realizadas, resolvemos implementar Algoritmos Genéticos para o problema de *Job Shop*, afim de otimizar a função *makespan*.

A implementação compreende um estudo comparativo entre vários operadores de *crossover* do Algoritmo Genético, próprios para a representação cromossômica inteira, e qual é o impacto na otimização da função objetivo a partir da geração da população inicial. Para este último caso, hibridizamos o Algoritmo Genético com métodos heurísticos de obtenção de soluções para o problema de *Job Shop* como uma forma de gerar alguns dos indivíduos da população inicial.

Antes de expor os estudos e os resultados obtidos, faz-se necessária algumas explicações adicionais sobre o Problema de *Job Shop*, as características do Algoritmo Genético implementado e os tipos de *crossover* para representações inteiras de um

Algoritmo Genético.

5.1 – Regras de Despacho para o Problema de *Job Shop*

A seção 2.4 já conceituou o Problema de *Job Shop*. Nesta seção, será definido o conceito de regras de despacho e como ele foi utilizado para gerar uma parte da população inicial dos Algoritmos Genéticos implementados.

Como o Problema de *Job Shop* é um problema NP-Completo, sabe-se que o esforço computacional dispendido para enumerar as possíveis soluções de uma instância do problema é bastante alto. Durante anos, vários pesquisadores empreenderam estudos que possibilitaram o desenvolvimento de heurísticas específicas para este problema, afim de gerar boas soluções a um custo computacional razoável.

Dentre estas heurísticas, existem as chamadas Regras de Despacho. Estas regras servem para, a partir de determinado critério, resolver o conflito de tarefas que estão prontas para serem processadas mas que requerem a mesma máquina (Baker, 1974). Quando estes métodos são aplicados, eles conseguem gerar soluções para instâncias dos problemas em pouco tempo.

Existe um algoritmo geral para aplicação de Regras de Despacho (Baker, 1974). Segue abaixo, na Figura 5.1.1, a descrição dos passos:

Algoritmo de Geração de Soluções por Regras de Despacho

Passo 1 – Dado *t*=0 indicando o tempo de início, PS_t vazio, representando a solução parcial. S_t Indica as operações preparadas para serem processadas. Em t=0, S_t contém todas as operações sem predecessores;

Passo 2 – Determinar $\phi^o = min_{j \in S_t} \phi_j$ que representa o menor tempo de término das operações de e a máquina m^0 na qual ϕ^0 seria realizada;

Passo 3 – Para cada operação $j \in S_t$ que requeira a máquina m^0 e para o qual o tempo de início σ das operações sejam tais que $\sigma_j < \phi^0$, calcular a operação *j* que será alocada em PS_t a partir do critério da Regra de Despacho e criar uma nova solução parcial PS_{t+1} para a próxima etapa;

Passo 4 – Para a nova solução parcial gerada, atualizar os seguintes dados:

a) Remover a operação *j* de ;

b) Formar pela adição da operação sucessora de *j;*

c) Incrementar *t* por 1.

Passo 5 – Retornar ao **Passo 2** até completar a programação de todas as operações.

Figura 5.1.1 – Algoritmo de Geração de Solução por Regra de Despacho.

No **Passo 3** descrito no algoritmo, deve-se implementar a Regra de Despacho que escolherá a operação a ser alocada caso exista mais de uma pronta para processamento e que requeiram a mesma máquina. Cada Regra de Despacho define um critério para solucionar estes conflitos. Abaixo tem-se alguns exemplos (Baker, 1974):

- SPT (*Shortest Processing Time*) – seleciona a operação com o menor tempo de processamento;
- FCFS (*First Come First Served*) – Seleciona a operação que está em S_t a mais tempo;
- MWKR (*Most Work Remaing*) – Seleciona a operação associada a tarefa com maior tempo de operações pendentes para ser processado;
- MOPNR (*Most Operations Remaing*) – Seleciona a operação cuja a tarefa associada tem o maior número de operações pendentes.

Estas Regras de Despacho foram utilizadas para gerar uma parte da população inicial de alguns Algoritmos Genéticos implementados. Este método, onde diferentes formas de busca são codificadas para trabalharem juntas, é chamado de hibridização.

Nos Algoritmos Genéticos implementados, alguns foram hibridizados com regras de despacho e outros não. Os testes feitos nas instâncias do problema propostas visam comparar o desempenho entre os algoritmos, e de que forma a geração da população inicial interfere no resultado encontrado.

5.2 – Características do Algoritmo Genético e operadores de *crossover*

Na seção 4.1 foram apresentados os conceitos e definições de Algoritmos Genéticos. Na implementação do Algoritmo Genético para o Problema de *Job Shop*, optou-se pela codificação inteira (Coley, 1999) em lista de preferência (Cheng, Gen e Tsujimura, 1996). Essa codificação necessita de operadores de *crossover* e mutação diferentes daqueles apresentados anteriormente.

Na codificação por lista de preferência, dado um problema com *n* tarefas e *m* máquinas, um cromossomo será composto por *m* subcromossomos, um para cada máquina. Cada subcromossomo é uma *string* de símbolos com tamanho *n*, onde cada símbolo representará uma operação de uma dada tarefa a ser processada naquela máquina. Estes subcromossomos descrevem a lista de preferência de execução em cada máquina.

A solução é deduzida a partir da análise de cada subcromossomo. Se a operação que estiver com mais alto nível de preferência em uma máquina estiver com as operações anteriores da tarefa a qual pertence processadas, ela estará pronta para ser processada. Caso contrário, avalia-se as operações com menor nível de preferência afim de encontrar aquela pronta para ser processada na máquina.

O operador de mutação serve para colocar na população aqueles genes que, porventura, se perderam durante o processo de cruzamento e formação de gerações, tornando a busca mais diversificada (Linden, 2006).

O operador de mutação utilizado trata-se do operador de inversão (Croce, Tadei e Volta, 1995). Neste operador, selecionam-se dois genes do cromossomo e inverte-se a subcadeia de genes compreendidos entre estes dois pontos. A Figura 5.2.1 exemplifica este processo.

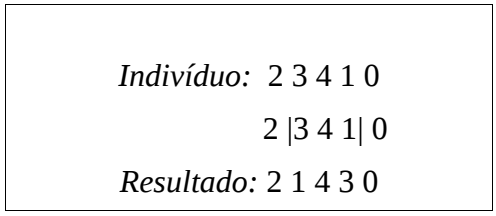


Figura 5.2.1: Exemplo do operador de inversão

Os operadores de *crossover* tem por finalidade possibilitar que bons genes da população possam se combinar, formando indivíduos melhores adaptados ao ambiente – ou seja, que tenham um melhor desempenho na função objetivo (Rezende, 2003). Nestes Algoritmos Genéticos foram implementados os *crossovers PMX, OX, CX, LOX,* e *PPX*.

Estes operadores de *crossover* foram retirados da literatura onde os mesmos foram aplicados com resultados satisfatórios a vários problemas de otimização combinatória em que a representação do Algoritmo Genético era inteira.

5.2.1 - PMX – *Partially Mapped Crossover*

O operador *PMX* foi proposto por Goldberg e Lingle (Potvin, 1996) para o Problema do Caixeiro Viajante. Dados dois cromossomos pais *p1* e *p2*, realizam-se

dois pontos de corte aleatórios sobre os mesmos. As subcadeias de genes encontradas entre estes cortes serão herdadas integralmente pelos indivíduos filhos *f1* e *f2*.

Estas subcadeias também determinam um mapeamento de relacionamento entre os genes dos pais afim de tratar a inviabilidade ocasionada pelo processo (Costa e Freitas, 2006). Tomemos a Figura 5.2.1.1 como exemplo:

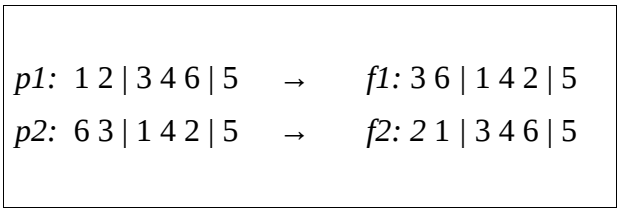


Figura 5.2.1.1: Exemplo de crossover PMX

No caso acima, os cortes em p1 (que cobrem os genes 3, 4 e 6) e os cortes em p2 (nos genes 1, 4 e 2) criam o relacionamento 3 ↔ 1, 4 ↔ 4 e 6 ↔ 2. Esse mapeamento significa que, após a troca de genes pelo PMX, os genes fora da subcadeia deverão ser mudados seguindo estas regras afim de tratar a inviabilidade. Por exemplo, o gene 1 em f1 herdado de p1 terá que ser mudado para 3, de acordo com o mapeamento.

5.2.2 - OX – *Order Crossover*

O operador *OX* foi proposto por Davis (Michalewicz, 1996) também para o Problema do Caixeiro Viajante. Assim como no *PMX*, dados dois cromossomos pais *p1* e *p2*, realizar-se-ão dois pontos de corte aleatórios sobre os mesmos, onde as subcadeias de genes encontradas entre estes cortes serão herdadas integralmente pelos indivíduos filhos *f1* e *f2*.

A partir do último corte em cada cromossomo, o método faz uma busca no

cromossomo do outro indivíduo pai pelos genes que não estão na subcadeia herdada, preenchendo assim o cromossomo (Costa e Freitas, 2006). A Figura 5.2.2.1 traz um exemplo:

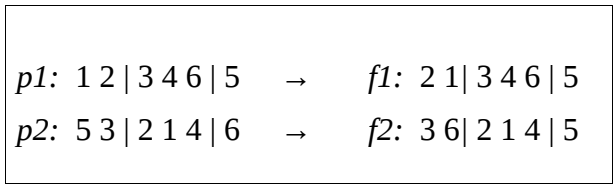


Figura 5.2.2.1: Exemplo de crossover OX

5.2.3 - CX – *Cycle Crossover*

O operador *CX* foi proposto por Oliver (Potvin, 1996) e aplicado ao Problema do Caixeiro Viajante. Ele trabalha sobre um subconjunto de genes que ocupam um mesmo conjunto de posições em ambos os pais.

Estes genes são então copiados para um determinado filho, enquanto os outros genes são copiados do outro pai (Costa e Freitas, 2006). A Figura 5.2.3.1 apresenta um exemplo:

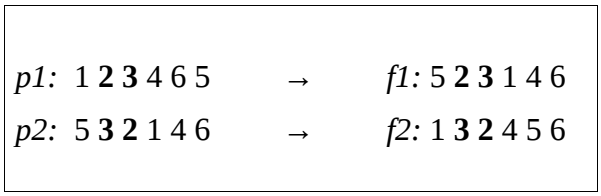


Figura 5.2.3.1: Exemplo de crossover CX

5.2.4 - LOX – *Linear Order Crossover*

O operador *LOX* é uma variação do operador *OX* e foi desenvolvido por Falkenauer e Bouffouix (Croce, Tadei e Volta, 1995). Neste operador, dois pontos de

corte aleatórios são realizados nos pais. Os genes pertencentes a subcadeia de um pai são retirados do outro pai, e os espaços vazios resultantes são reunidos entre os cortes.

Após esta etapa, troca-se a subcadeia anterior entre os pais, gerando os filhos. Veja o exemplo da aplicação deste operador na Figura 5.2.4.1 abaixo:

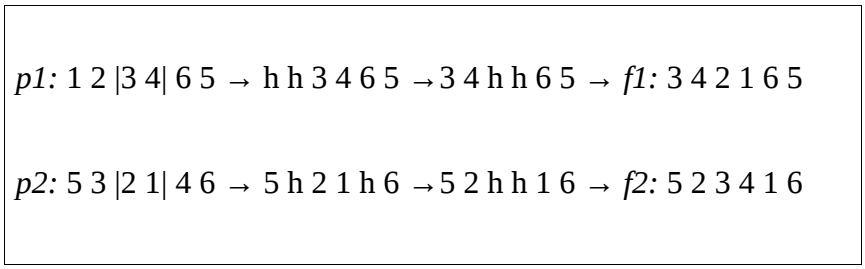


Figura 5.2.4.1: Exemplo de crossover LOX.

5.2.5 - PPX – *Precedence Preservative Crossover*

O operador *PPX* conserva a ordem de precedência dos genes dos pais (Mattfeld e Bierwirth, 2004). Através da geração de uma máscara similar a gerada para o *crossover* uniforme ou máscara (Carvalho, Braga e Ludermir, 2003), o método vai selecionando genes de um dos pais. Caso a seleção daquele gene torne o filho inviável, o operador busca o próximo gene do mesmo pai, desde que a viabilidade do filho seja mantida.

Abaixo, na Figura 5.2.5.1, temos o exemplo de aplicação do *PPX*. Na máscara, para geração de *f1*, os valores *0* indicam a busca de gene do *p1;* os valores *1* indicam busca de gene no cromossomo *p2*. Para geração de *f2* os valores de *0* e *1* são invertidos.

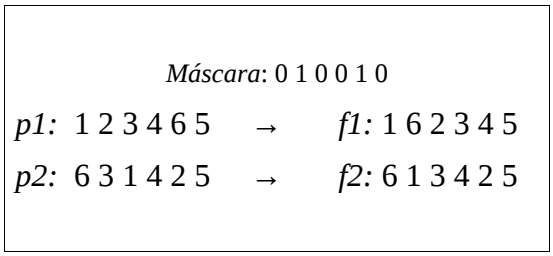


Figura 5.2.5.1: Exemplo de crossover PPX

Com os operadores de *crossover* apresentados, a próxima seção discorrerá sobre o conjunto de testes e os resultados obtidos. Cada *crossover* agirá sobre cada subcromossomo da representação por lista de preferência, que tem um formato semelhante aos apresentados nos exemplos anteriores.

6.0 – Testes e Resultados Obtidos

As demais características do Algoritmo Genético implementado são: valor da taxa de mutação em 0.01; taxa de *crossover* em 0.9. A população inicial foi gerada de duas formas: para um conjunto de Algoritmos Genéticos, ela foi gerada de forma aleatória; na segunda forma, 40% da população inicial foi gerada utilizando as regras de despacho apresentadas na seção 5.1. Neste último caso, cada solução gerada por uma regra de despacho é copiada 10 vezes para a população inicial. Os demais 60% da população são gerados aleatoriamente.

Para todos os métodos implementados, os 20 melhores indivíduos de cada geração foram mantidos na próxima, em um processo conhecido como elitismo (Coley, 1999). O tamanho da população é fixado no valor de 100 indivíduos. O critério de parada utilizado no presente trabalho foi o número máximo de gerações no valor de 1000 interações.

As instâncias do problema proposto foram retiradas da literatura, e compreendem alguns dos problemas encontrados na OR-Library. A Tabela 6.1 abaixo apresenta os tipos de problemas com seus respectivos valores no número de máquinas e tarefas:

Problemas		
Instâncias	Nº de Tarefas	Nº de Máquinas
ft06	6	6
abz5	10	10
abz8	20	15
yn1	20	20
swv20	50	10

Tabela 6.1: Instâncias para Testes Problema de Job Shop com Data de Entrega

Cada operador de *crossover* foi implementado ao Algoritmo Genético nas duas formas de geração da população inicial. Eles foram executados 5 vezes para cada instância, de forma que a média e o melhor resultado para cada método pudessem ser obtidos. A Tabela 6.2 expõe a média e os melhores resultados de cada Algoritmo abordando o problema ft06; a Tabela 6.3 apresenta estes valores para o problema abz5; Tabela 6.4 para abz8, Tabela 6.5 para yn1 e Tabela 6.6 para o problema swv20. Nestas tabelas, o primeiro resultado relaciona-se com os Algoritmos Genéticos com população inicial totalmente aleatória; o segundo resultado são para os algoritmos hibridizados com Regras de Despacho:

	ft06			
	Sem Regras de Despacho		**Com Regras de Despacho**	
	Média	**Melhor Resultado**	**Média**	**Melhor Resultado**
PMX	60,4	57	55	55
OX	63,6	58	57	55
CX	63,4	60	58,8	55
LOX	59,6	57	55	55
PPX	60	57	55	55

Tabela 6.2: Resultados para a instância ft06

	abz5			
	Sem Regras de Despacho		**Com Regras de Despacho**	
	Média	**Melhor Resultado**	**Média**	**Melhor Resultado**
PMX	1674,6	1555	1361,2	1332
OX	1903,8	1844	1390,2	1361
CX	1932	1695	1417	1399
LOX	1595,8	1417	1339,6	1301
PPX	1840,4	1703	1363,6	1337

Tabela 6.3: Resultados para a instância abz5

	abz8			
	Sem Regras de Despacho		**Com Regras de Despacho**	
	Média	**Melhor Resultado**	**Média**	**Melhor Resultado**
PMX	1541,8	1327	792,2	792
OX	1950	1748	806,8	805
CX	1763,2	1551	808	808
LOX	1854,6	1533	795	787
PPX	1929,8	1730	808	808

Tabela 6.4: Resultados para a instância abz8

	yn1			
	Sem Regras de Despacho		**Com Regras de Despacho**	
	Média	**Melhor Resultado**	**Média**	**Melhor Resultado**
PMX	2377	1932	1081,6	1069
OX	2996,8	2776	1102,6	1094
CX	3015	2739	1124,6	1123
LOX	2697,8	2167	1072,4	1064
PPX	3106,8	2678	1118,4	1097

Tabela 6.5: Resultados para a instância yn1

	swv20			
	Sem Regras de Despacho		**Com Regras de Despacho**	
	Média	**Melhor Resultado**	**Média**	**Melhor Resultado**
PMX	4857,4	4624	2879,8	2843
OX	7497,2	6883	2928	2928
CX	7146	6820	2928	2928
LOX	5304	5069	2864,8	2838
PPX	7675,4	7279	2928	2928

Tabela 6.6: Resultados para a instância swv20

A partir destes números, traçou-se um gráfico comparativo entre as médias dos Algoritmos Genéticos para cada instância. A Figura 6.1 expõe o gráfico para o problema ft06; a Figura 6.2 apresenta o gráfico para o problema abz5; Figura 6.3 para abz8, Figura 6.4 para yn1 e Figura 6.5 para o problema swv20.

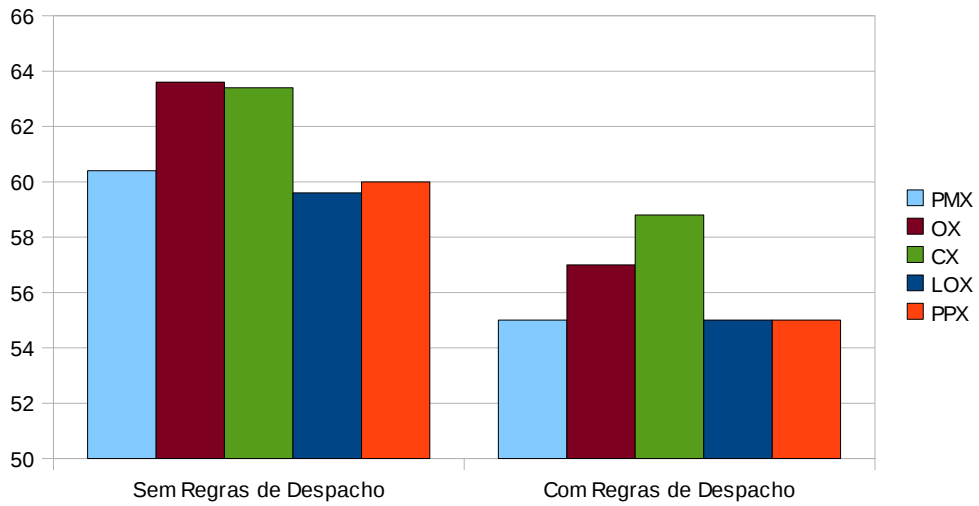


Figura 6.1: Gráfico para a instância ft06

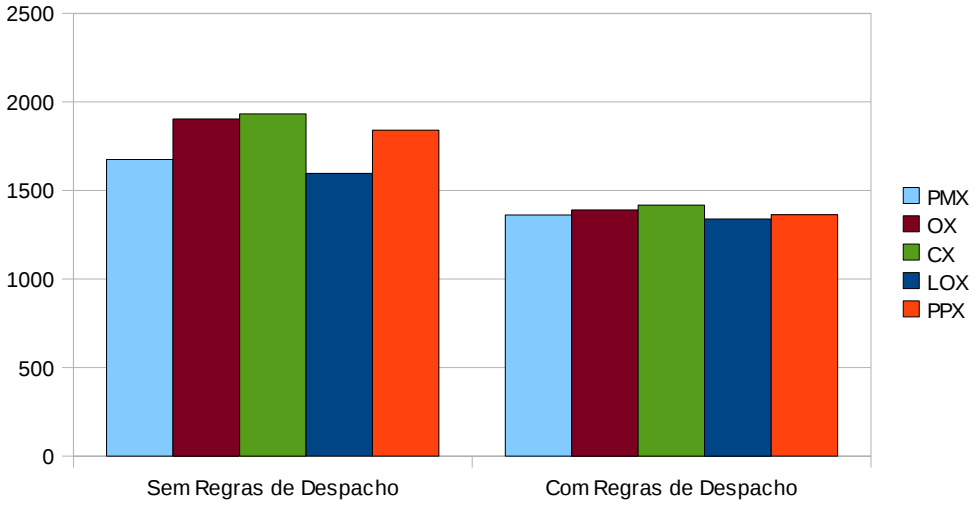


Figura 6.2: Gráfico para a instância abz5

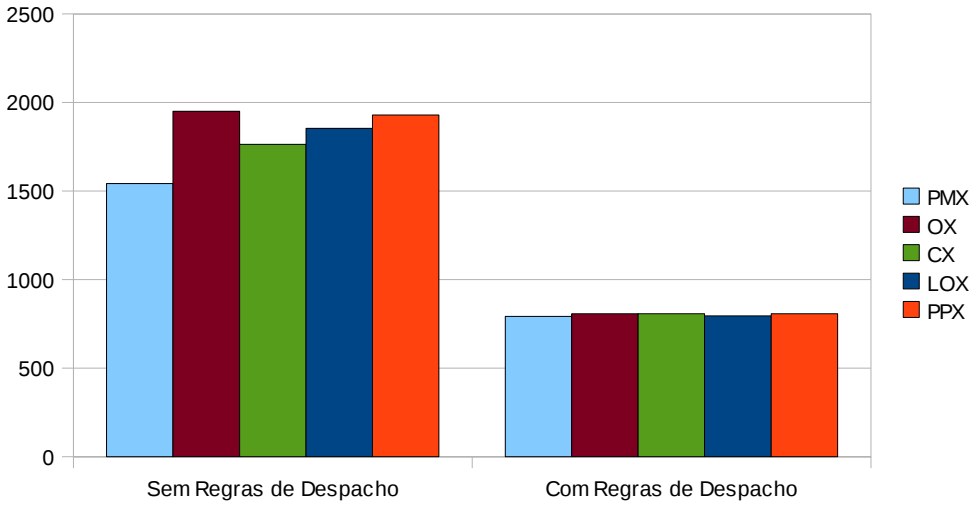


Figura 6.3: Gráfico para a instância abz8

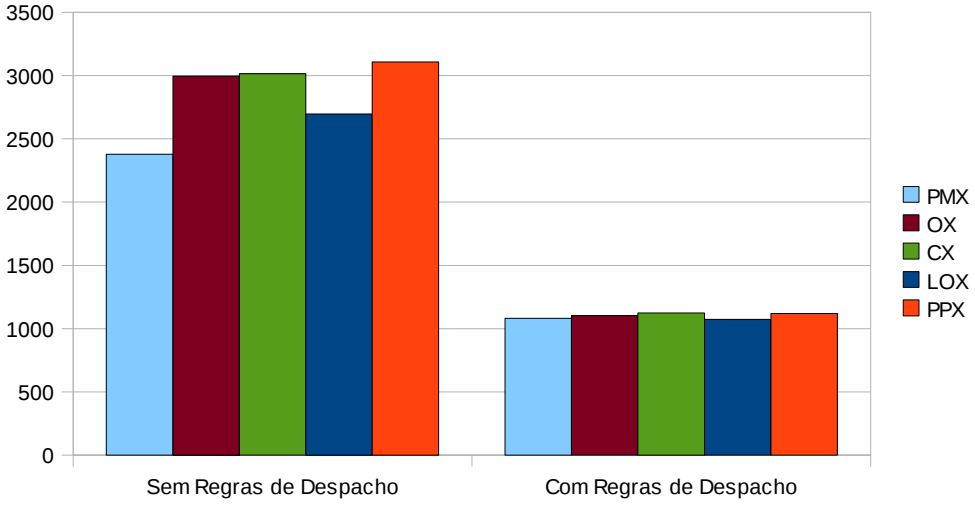


Figura 6.4: Gráfico para a instância yn1

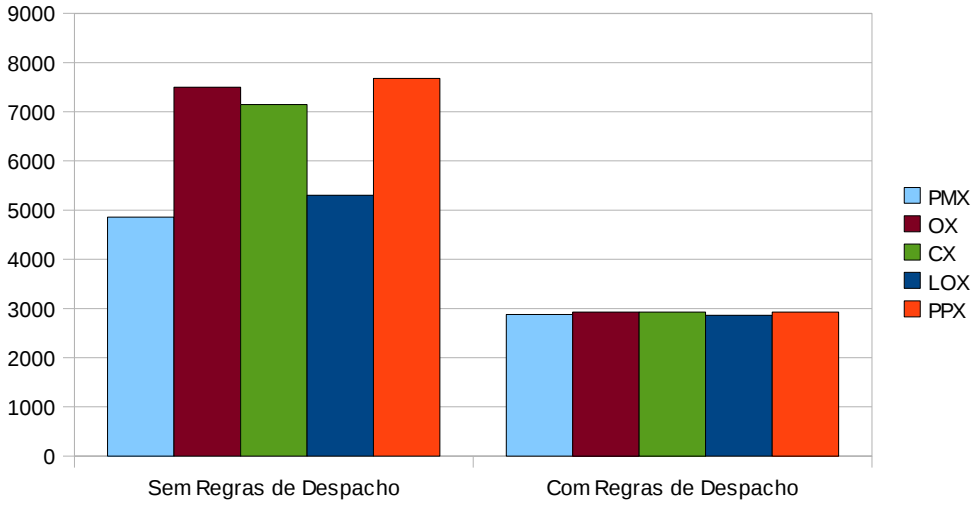


Figura 6.5: Gráfico para a instância swv20

Percebe-se que a hibridização com Regras de Despacho possibilitou a obtenção de indivíduos com valores na função objetivo bem melhores que o método totalmente aleatório. O *crossover PMX* foi o que obteve melhores resultados, em média, na abordagem não hibridizada; no entanto, nos testes para as instâncias *ft06* e *abz5*, o operador *LOX* obteve melhor desempenho. Para o método com Regras de Despacho, o *crossover LOX* alcançou melhores valores em todas as instâncias, exceto em *abz8*. A Figura 6.6 mostra um gráfico do comportamento geral dos operadores para o Algoritmo Genético sem Regras de Despacho:

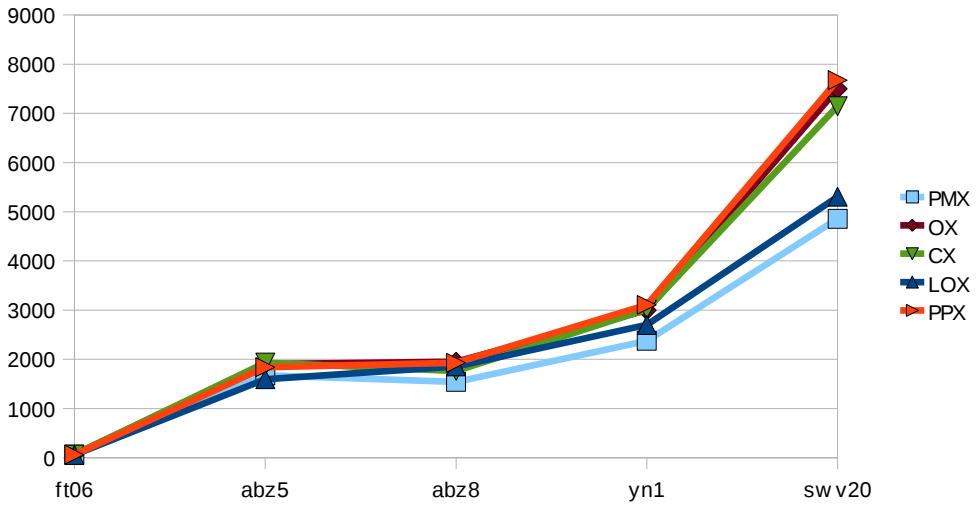


Figura 6.6: Gráfico do Comportamento Geral dos Operadores sem Despacho

Já para os algoritmos hibridizados, apesar do *crossover LOX* ter alcançado um melhor desempenho, todos os operadores acabaram por ter um desempenho muito parecido, como mostra a Figura 6.7 com o comportamento geral dos operadores para o Algoritmo Genético com Regras de Despacho:

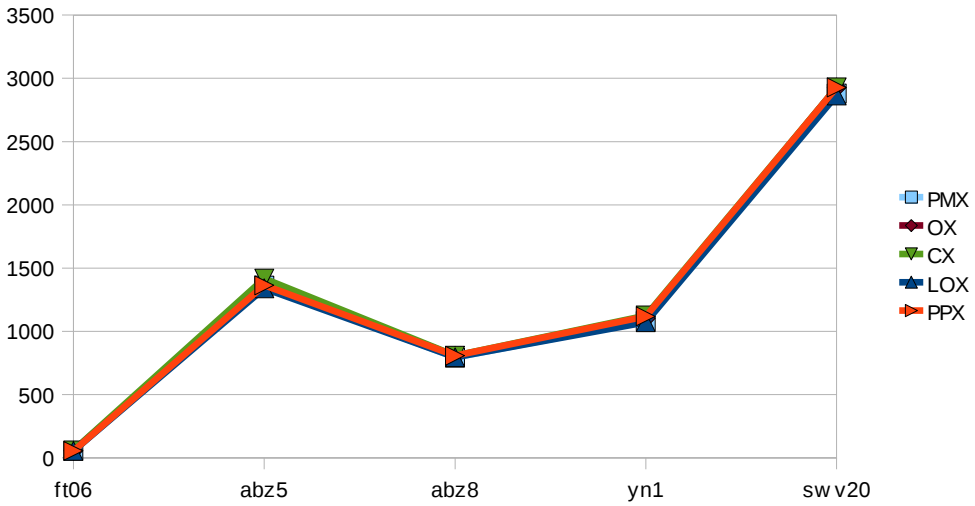


Figura 6.7: Gráfico do Comportamento Geral dos Operadores com Despacho

7.0 - Conclusões e Trabalhos Futuros

Os problemas de otimização combinatória estão entre os mais difíceis de serem resolvidos em tempo razoável, por conta do alto custo computacional dispendido por algoritmos de resolução convencionais baseados em enumeração explícita ou implícita (como *branch-and-bound*, programação dinâmica e outros).

Entretanto, muitos pesquisadores se utilizam de uma abordagem heurística para buscar as melhores soluções, a custo computacional admissível, afim de resolver tais problemas. Este tipo de problema é comumente enfrentado por organizações como empresas, indústrias e governos.

Esta monografia tratou de fazer um levantamento bibliográfico dos problemas de Otimização Combinatória do tipo NP-Completo e de métodos metaheurísticos de resolução. Tratou-se aqui do Problema do Caixeiro Viajante, Problema da Mochila, Problema da Árvore de Steiner e Problema de *Job Shop.* Dentre as metaheurísticas, discorremos sobre Algoritmos Genéticos, Busca Tabu, Colônia de Formigas e GRASP.

Para se ter uma experiência empírica de programação desse tipo de algoritmo, foi escolhida a implementação de Algoritmos Genéticos para a função *makespan* no Problema de *Job Shop*. Afim de contribuir com o desenvolvimento na área, foi feito um estudo comparativo entre operadores de *crossover* que obtiveram resultados satisfatórios em vários outros problemas encontrados na literatura, além de comparar o desempenho da função objetivo ocasionada pela geração da população inicial através de heurísticas hibridizadas aos Algoritmos Genéticos e, na segunda maneira, de forma totalmente aleatória.

Os resultados obtidos mostraram um desempenho melhor de algoritmos com geração da população hibridizada pelas regras de despacho, e o *crossover* com

melhor desempenho foi o *LOX*, apesar do *PMX* ter tido um desempenho melhor para os algoritmos com geração da população inicial totalmente aleatória.

O desenvolvimento da área de metaheurísticas e métodos heurísticos em geral tem grande importância para a resolução de problemas de otimização combinatória. Enquanto um algoritmo exato com custo computacional razoável não é desenvolvido, se for possível de ser desenvolvido, métodos heurísticos e metaheurísticos vão se tornando cada vez mais versáteis e robustos.

Dada a dimensão e importância que os problemas de otimização combinatória tem para instituições e organizações contemporâneas, o uso de métodos heurísticos e metaheuríticos recebem grande importância por parte destes grupos.

Como trabalho futuro, espera-se hibridizar o Algoritmo Genético com alguma metaheurística de busca local, como a Busca Tabu. A premissa é que, enquanto o Algoritmo Genético faz uma otimização mais global, um método de busca local poderá se utilizar de uma ou várias boas soluções iniciais (alcançadas após a execução do Algoritmo Genético) e, a partir delas, analisar suas vizinhanças em busca de uma solução que otimize a função objetivo.

Assim, espera-se que os estudos aqui realizados e documentados possam servir de introdução e contribuição ao desenvolvimento de métodos cada vez mais eficientes.

Bibliografia

ARENALES, Marcos; ARMENTANO, Vinícius; MORABITO, Reinaldo; YANASSE, Horacio. **Pesquisa Operacional para Cursos de Engenharia**, Editora Elsevier, São Paulo, 2007;

BAKER, Kenneth R. **Introduction to Sequencing and Scheduling**, John Wiley & Sons Inc, New York, 1974;

CAMPELLO, R. E. e MACULAN, N. **Algoritmos e Heurísticas – Desenvolvimento e Avaliação de Performance**, Niterói, Rio de Janeiro, 1994;

CARVALHO, André Carlos P. de L. F.; BRAGA, Antônio de P.; LUDERMIR, Teresa B. *Computação Evolutiva* In: REZENDE, Solange O. **Sistemas Inteligentes – Fundamentos e Aplicações**, Editora Manole, São Paulo, pág. 225 – 248, 2003,;

CHENG, R.; GEN, M; TSUJIMURA, Y. *A tutorial survey of job shop scheduling problems using genetic algorithms: representation*, **Comput Ind. Eng.,** vol. 30, nº 4, pág. 983 – 997, 1996;

COLEY, David A. **An Introduction to Genetic Algorithms for Scientists and Engineers**, World Scientific Publishing, 1999;

CORDENONSI, Andre Zanki; MULLER, Felipe Martins; BASTOS, Fábio da Purificação de. *O Ensino de Heurísticas e Metaheurísticas na área de Pesquisa Operacional sob a ótica da Educação Dialógica Problematizadora*, **Novas Tecnologias na Educação**, v.3 nº01, Universidade Federal do Rio Grande do Sul, 2005;

COSTA, A. de O. e FREITAS, A. S. (2006). *Algoritmos Genéticos: alguns experimentos com os operadores de cruzamento ("crossover") para o Problema do Caixeiro Viajante Assimétrico*, **Anais do XXVI Encontro Nacional de Engenharia de** Produção, 2006*;*

COSTA, Antônio de Oliveira; SARAIVA, Filipe de Oliveira. *Algoritmo Genético: Conceitos e Implementação para o Problema da Árvore de Steiner*, **Anais do XXVI Congresso Nacional dos Estudantes de Computação**, 2008;

CROCE, F. della, TADEI, R. e VOLTA, G. *A Genetic Algorithm for the Job Shop Problem*, **Computers Ops Res**. Vol. 22, nº 1, 15 – 24, 1995;

DORIGO, M.; MANIEZZO, V.; COLORNI, A. *The Ant System: Optimization by a Colony of Cooperating Agents*, **IEEE Transactions on System, Man, and Cybernetics – Part B**, vol. 26 nº

1, pág 1 – 13, 1996;

DORIGO, M; STUTZLE, T. *The Ant Colony Optimization Metaheuristic: Algorithms, Applications and Advances* In: GLOVER, F.; KOCHENBERGER, G. **Handbook of Metaheuristics**, Kluwer Academic Publishers, New York, pág. 251 – 287, 2003;

GLOVER, F.; KOCHENBERGER, G. **Handbook of Metaheuristics**, Kluwer Academic Publishers, New York, pág. 251 – 287, 2003;

GOLDBARG, M. C.; LUNA, H. P. L. **Otimização Combinatória e Programação Linear**, Editora Campus, Rio de Janeiro 2005;

GOODRICH, Michael T.; TAMASSIA, Roberto. **Projeto de Algoritmos – Fundamentos, Análise e exemplos da Internet**, Editora Artmed, São Paulo, 2002;

KAPSALIS, A.; RAYWARD-SMITH, V.J.; SMITH, G.D. *Solving the Graphical Steiner Tree Problem using Genetic Algorithm*, **Journal of the Operational Research Society**, vol. 44, nº. 4, pág. 397 – 406, 1993;

LINDEN, Ricardo. **Algoritmos Genéticos – Uma Importante Ferramenta da Inteligência Computacional**, Editora Brasport, Rio de Janeiro, 2006;

MATTFELD, D. C.; BIERWIRTH, C. *An Efficient Genetic Algorithm for Job Shop Scheduling with Tardiness Objectives*, **European Journal of Operational Research**, vol. 155, pág. 616 – 630, 2004;

MICHALEWICZ, Z. **Genetic Algorithms + Data Structures = Evolution Programs**, Alemanha, 1996;

NORVIG, Peter; RUSSEL, Stuart. **Inteligência Artificial**, Editora Campus, São Paulo, 2004;

POTVIN, J. Y. *Genetic Algorithms for the Traveling Salesman Problem*, **Annals of Operations Research**, vol 6, 339 – 370, 1996;

REEVES, Colin R. (org.). **Modern Heuristic Search Methods**, Orient Longman, Great Britain, 1996.

RESENDE, M. G. C.; RIBEIRO, C. C. *Greedy Randomized Adaptative Search Procedures* In: GLOVER, F.; KOCHENBERGER, G. **Handbook of Metaheuristics**, Kluwer Academic Publishers, New York, pág. 251 – 287, 2003;

VON ZUBEN, F. J.; ATTUX, R. R. F. Introdução à Computação Natural – Inteligência de Enxame, Notas de Aula na UNICAMP, sem data.

Apêndice A

Tabela de Resultado das Execuções dos Algoritmos Genéticos Implementados

Algoritmos Genéticos sem Regras de Despacho

Problemas	Número de Jobs	Número de Máquinas	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Média	Melhor Resultado
			PMX						
ft06	6	6	63	64	57	57	61	60,4	57
abz5	10	10	1668	1607	1714	1829	1555	1674,6	1555
abz8	20	15	1445	1327	2081	1517	1339	1541,8	1327
yn1	20	20	2270	2371	1932	3371	1941	2377	1932
swv20	50	10	4863	5397	4633	4770	4624	4857,4	4624
			OX						
Problemas	Número de Jobs	Número de Máquinas	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Média	Melhor Resultado
ft06	6	6	67	63	63	58	67	63,6	58
abz5	10	10	1909	1855	1844	1991	1920	1903,8	1844
abz8	20	15	1820	1883	2019	1748	2280	1950	1748
yn1	20	20	2972	3081	2951	2776	3204	2996,8	2776
swv20	50	10	8226	7452	7445	7480	6883	7497,2	6883
			LOX						
Problemas	Número de Jobs	Número de Máquinas	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Média	Melhor Resultado
ft06	6	6	57	59	60	60	62	59,6	57
abz5	10	10	1417	1575	1656	1731	1600	1595,8	1417
abz8	20	15	1974	1533	1608	2102	2056	1854,6	1533
yn1	20	20	3463	2167	2219	3091	2549	2697,8	2167
swv20	50	10	5195	5069	5604	5299	5353	5304	5069
			CX						
Problemas	Número de Jobs	Número de Máquinas	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Média	Melhor Resultado
ft06	6	6	69	60	63	64	61	63,4	60
abz5	10	10	1695	2060	2004	2029	1872	1932	1695
abz8	20	15	1892	1780	1765	1828	1551	1763,2	1551
yn1	20	20	3000	3014	2908	2739	3414	3015	2739
swv20	50	10	6820	7294	7185	6949	7482	7146	6820
			PPX						
Problemas	Número de Jobs	Número de Máquinas	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Média	Melhor Resultado
ft06	6	6	57	64	59	59	61	60	57
abz5	10	10	1748	1974	1884	1703	1893	1840,4	1703
abz8	20	15	1760	2168	1656	2335	1730	1929,8	1730
yn1	20	20	3149	3039	3133	2678	3535	3106,8	2678
swv20	50	10	7561	7279	7811	7939	7787	7675,4	7279

Algoritmos Genéticos com Regras de Despacho

Problemas	Número de Jobs	Número de Máquinas	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Média	Melhor Resultado
PMX com Regras de Despacho									
ft06	6	6	55	55	55	55	55	55	55
abz5	10	10	1332	1369	1361	1388	1356	1361,2	1332
abz8	20	15	792	792	792	792	793	792,2	792
yn1	20	20	1069	1086	1097	1072	1084	1081,6	1069
swv20	50	10	2891	2910	2898	2843	2857	2879,8	2843
OX com Regras de Despacho									
Problemas	Número de Jobs	Número de Máquinas	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Média	Melhor Resultado
ft06	6	6	57	55	55	58	60	57	55
abz5	10	10	1428	1361	1361	1382	1419	1390,2	1361
abz8	20	15	805	808	808	808	805	806,8	805
yn1	20	20	1105	1099	1099	1116	1094	1102,6	1094
swv20	50	10	2928	2928	2928	2928	2928	2928	2928
LOX com Regras de Despacho									
Problemas	Número de Jobs	Número de Máquinas	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Média	Melhor Resultado
ft06	6	6	55	55	55	55	55	55	55
abz5	10	10	1301	1338	1337	1361	1361	1339,6	1301
abz8	20	15	798	798	787	798	794	795	787
yn1	20	20	1082	1064	1072	1080	1064	1072,4	1064
swv20	50	10	2868	2840	2868	2910	2838	2864,8	2838
CX com Regras de Despacho									
Problemas	Número de Jobs	Número de Máquinas	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Média	Melhor Resultado
ft06	6	6	60	60	55	59	60	58,8	55
abz5	10	10	1429	1399	1399	1429	1429	1417	1399
abz8	20	15	808	808	808	808	808	808	808
yn1	20	20	1125	1125	1125	1123	1125	1124,6	1123
swv20	50	10	2928	2928	2928	2928	2928	2928	2928
PPX com Regras de Despacho									
Problemas	Número de Jobs	Número de Máquinas	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Média	Melhor Resultado
ft06	6	6	55	55	55	55	55	55	55
abz5	10	10	1361	1338	1394	1337	1388	1363,6	1337
abz8	20	15	808	808	808	808	808	808	808
yn1	20	20	1097	1125	1125	1120	1125	1118,4	1097
swv20	50	10	2928	2928	2928	2928	2928	2928	2928

Apêndice B

Projeto da Monografia

Justificativa

Nos últimos cinquenta anos, os problemas de Otimização Combinatória estão entre os temas mais pesquisados e discutidos no ramo de ciências como a matemática, computação e engenharias (Goldbarg, Luna, 2005). Tratam-se de problemas nos quais, dada uma série de restrições e uma Função Objetivo, encontrar uma solução que cumpra as restrições dadas e otimize o melhor possível o valor da Função Objetivo. Problemas desse tipo caracterizam-se pelo número muito grande de soluções viáveis, demandando um grande esforço computacional para solucioná-los de maneira exata.

Apesar de, normalmente, serem de fácil entendimento, problemas de Otimização Combinatória são de difícil solução através de métodos de resolução exatos. A maior dificuldade é a "explosão combinatória", proporcionada pela combinação de valores diferentes para as diversas variáveis de decisão que compõem o modelo do problema. O número dessas combinações é notadamente muito grande, impossibilitando métodos de resoluções exatos baseados em enumeração implícita (Reeves, 96).

Para ilustrar, tomemos como exemplo um dos problemas de otimização combinatória mais tratados na literatura especializada: o Problema do Caixeiro Viajante. Este problema em sua forma clássica consiste em, dado um conjunto de N cidades ligadas entre si, um determinado caixeiro-viajante sairá de uma dessas cidades e deverá visitar todas as demais cidades do conjunto, uma única vez, e voltar a cidade de origem, ao custo menor possível.

Como o ponto de início é arbitrário, existem $(N - 1)! + 1$ possíveis soluções (*idem*). Supondo um computador capaz de listar todas as possíveis soluções do Problema do Caixeiro Viajante para 20 cidades em 1 hora. Este mesmo computador, seguindo a mesma fórmula utilizada para listar todas as soluções possíveis para o

problema com 20 cidades, levará 20 horas para listar as soluções para o problema com 21 cidades; 17.5 dias para o problema com 22 cidades; e aproximadamente 6 séculos para o problema com 25 cidades.

Para além da curiosidade matemática, é importante ressaltar que diferentes modelos de problemas de Otimização Combinatória são encontrados em problemas corriqueiros de empresas, governos e organizações em geral, seja na prestação de algum serviço ou execução de algum projeto.

Tomemos como exemplo, novamente, o Problema do Caixeiro Viajante exposto anteriormente. Entre as diversas aplicações a qual a resolução desse problema otimiza o desempenho de um serviço, podemos elencar a própria atuação de um caixeiro viajante que deve fazer cobranças em um conjunto de cidades, a alocação de carteiros para visitar as casas de determinados bairros, construção de redes de esgoto ou abastecimento, alocação de entregas de produtos em diferentes estabelecimentos, etc.

Portanto, a resolução de problemas desse tipo tem aplicações práticas de grande importância para diversas organizações. Porém, como colocado anteriormente, problemas de Otimização Combinatória são de difícil solução através de métodos exatos. Como então superar essa situação aparentemente intransponível?

Por volta de meados dos anos 70, diferentes grupos de pesquisadores confrontados com a dificuldade da resolução exata de problemas desse tipo, começaram a partir para uma nova abordagem. A ideia era conseguir uma boa solução para o problema, não necessariamente ótima, porém suficiente para ser aceita pelo modelo, a um custo computacional viável. Começa assim o desenvolvimento e aplicação de métodos computacionais conhecidos como Heurísticas e Metaheurísticas (*ibidem*).

Com o avanço dos estudos nesse campo, diversos tipos de heurísticas foram

propostas e aplicadas a variados tipos de problemas, principalmente os de Otimização Combinatória. Busca Tabu, Arrefecimento Simulado, Algoritmo Genético e Colônia de Formigas são apenas alguns exemplos de métodos heurísticos, e já foram aplicados com sucesso a problemas como o do Caixeiro Viajante, Árvore de Steiner (Costa, Saraiva, 07), Cobertura de Conjuntos, Mochila, *Job Shop*, e muitos outros.

Dado a importância do desenvolvimento dessa área para as organizações em geral, o estudo de métodos heurísticos e metaheurísticos aplicados a resolução de problemas de otimização combinatória caracteriza-se como um importante campo de estudos, com contribuições significativas a dar tanto para a sociedade quanto para a ciência em geral, tornando a pesquisa nessa área interessante, como espera-se de um trabalho de estágio curricular.

Por tanto, visto a importância do desenvolvimento da área e sua gama de aplicações, o presente projeto objetiva fazer um levantamento bibliográfico acerca do tema, buscar os principais problemas pesquisados e expor os principais métodos metaheurísticos, analisando suas características.

Em seguida, será selecionado um problema de otimização combinatória, ao qual serão aplicados alguns dos métodos metaheurísticos estudados. A ideia é fazer um estudo comparativo entre os sistemas, avaliar seu desempenho e entender como a característica de cada método contribuiu para alcançar aquele resultado.

A conclusão destes estudos possibilitará um maior entendimento do tema, além da produção de uma monografia e artigos científicos divulgando os resultados encontrados.

Objetivos

- **Gerais**
  - Fazer um levantamento bibliográfico acerca da literatura especializada sobre Problemas de Otimização Combinatória, Complexidade Computacional e Metaheurísticas, afim de perceber a importância do estudo dessa área tanto para a resolução de problemas logísticos de organizações em geral, quanto para o desenvolvimento teórico da Ciência da Computação como um todo.
- **Específicos**
  - Fazer um levantamento bibliográfico acerca da literatura especializada sobre Problemas de Otimização Combinatória;
  - Fazer um levantamento bibliográfico acerca da literatura especializada sobre métodos Heurísticos e Metaheurísticos já utilizados para solucionar problemas de Otimização Combinatória e seus respectivos desempenhos;
  - Compreender a importância do desenvolvimento de métodos metaheurísticos e da solução de problemas de otimização combinatória para as organizações em geral;
  - Compreender a aplicação de alguns problemas de otimização combinatória presentes na literatura especializada;
  - Compreender o funcionamento de algumas metaheurísticas presentes na literatura especializada;
  - Cumprir carga horária e os requisitos necessários para a conclusão da

disciplina de Estágio Supervisionado II;

- Selecionar algumas metaheurísticas e implementá-las para algum problema de otimização combinatória também a ser selecionado;

- Produzir relatórios afim de registrar os resultados, as conclusões e a experiência em si de se trabalhar metaheurísticas aplicadas a problemas de otimização combinatória;

- Produzir trabalhos científicos afim de divulgar a produção acadêmica do curso de computação da UFPI em eventos e congressos pelo país.

Metodologia

- Leitura de artigos em periódicos e anais de congresso que tratam sobre aplicações de metaheurísticas a problemas de otimização combinatória;
- Leitura de livros sobre o tema a ser tratado;
- Discussões entre orientador e orientando acerca do entendimento do assunto e dos passos a serem tomados pela pesquisa.

Viabilidade

O ambiente disponibilizado pelo curso de computação da UFPI, em suas atuais condições, permite a execução da pesquisa da maneira como está explícita em seu cronograma, seguindo as metodologias expostas, afim de atingir os objetivos supracitados.

A implementação e testes dos sistemas serão realizadas no computador particular do orientando. Apesar de ser uma máquina de porte mediano, ela é suficiente para a realização prática do presente projeto.

O acervo das bibliotecas da instituição contém o material básico para um primeiro embasamento sobre os temas. Demais livros interessantes à pesquisa, como os de Inteligência Artificial e Pesquisa Operacional, estão em posse do orientando, além de material bibliográfico mais específico e avançado encontrados em artigos no site de Periódicos da CAPES e outros pertencentes ao orientador, sujeito a disponibilização para o avançar do estudo.

A conexão a Internet da instituição, devido a sua alta velocidade e disponibilidade, também servirá como meio de acesso a fonte de pesquisa fundamental para busca de conceitos, artigos, anais de congresso e demais material científico sobre os temas.

Cronograma

	2008				2009						
	Set.	Out.	Nov.	Dez.	Jan.	Fev.	Mar.	Abr.	Mai.	Jun.	Jul.
Leitura de artigos e livros sobre os assuntos	X	X	X	X	X	X	X	X	X	X	X
Discussão entre orientador e orientando	X	X	X	X	X	X	X	X	X	X	X
Implementação das metaheurísticas selecionada				X	X	X	X	X	X	X	
Produção de relatórios							X	X	X	X	
Produção de artigo científico							X	X	X	X	
Produção da monografia		X	X	X			X	X	X	X	
Apresentação da monografia											X