4.
Modelo Co-evolucionário para Otimização da Programação da Produção.

4.1.
Introdução.

O modelo proposto para solucionar este problema é um modelo co-evolucionário cooperativo formado por duas espécies. A primeira espécie irá decidir qual tarefa planejar, enquanto que a segunda irá decidir com quais recursos esta tarefa será executada.

As representações utilizadas nos indivíduos de cada uma destas espécies podem ser vistas na próxima seção.

Mais adiante serão discutidas as demais características do modelo, tais como: decodificação da solução (em uma programação válida), avaliação de um indivíduo, e operadores utilizados.

Outra discussão fundamental para a definição completa do modelo é a forma de representação do tempo. Este tópico é abordado em mais detalhes ao final do capítulo.

4.2.
Representação.

A especificação de uma representação apropriada para um *scheduling* é de fundamental importância para o desempenho de um algoritmo evolucionário. As representações usadas até o momento variam de simples *strings* a complexas estruturas de dados. Conseqüentemente, os operadores empregados variam também dos mais simples até algoritmos complexos baseados em conhecimento específico do problema.

As representações independentes de domínio parecem ser especialmente apropriadas para problemas de *scheduling* com poucas restrições e uma estrutura do problema razoavelmente simples. Em particular, problemas de *flow shop* têm

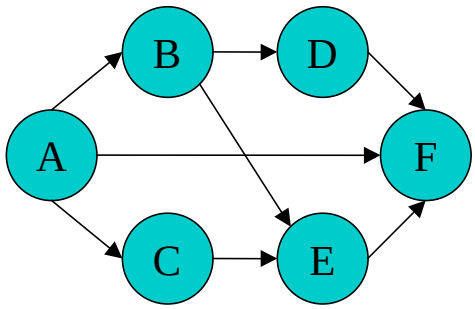
sido abordados com sucesso com a representação por seqüência de tarefas (descrita anteriormente) (Bäck et al, 1997).

A representação adotada nesta dissertação conta com duas espécies, que evoluem paralelamente aspectos diversos do problema e colaboram para formar a solução do problema. A arquitetura utilizada foi descrita na seção 3.3.1. A seguir são apresentadas as espécies utilizadas neste processo cooperativo.

A primeira espécie, chamada de "Alocação de Tempo" conta com duas estruturas: a lista de prioridades e o grafo de precedências. A lista de prioridades contém as prioridades de cada tarefa. O grafo contém as informações de precedências (ou dependências) entre tarefas, ou seja, quais tarefas devem ser programadas antes de outras.

Lista de Prioridades

Tarefa	**A**	**B**	**C**	**D**	**E**	**F**
Prioridade	**5**	**1**	**2**	**6**	**4**	**3**



Grafo de Precedências

Figura 6 – Representação da espécie "Alocação de Tempo".

Na figura 6, são mostradas as duas estruturas que compõem a representação da espécie "Alocação de Tempo". A lista de prioridades é o cromossomo onde os operadores genéticos irão atuar. A seguir, na figura 7, é apresentado um esboço do algoritmo para selecionar as tarefas a programar a partir do cromossomo da espécie 1.

```
procedure Programar tarefas
grafo ← grafo de precedências
while (tamanho(grafo)>0)
     if (todo vértice tem um precedente)
     then
        grafo inválido
        exit
     else
        v ← vértice com a maior prioridade entre os sem
     precedentes

        programar v
        remover v do grafo e todos as ligações que partem dele
     end if
end while
```

Figura 7 – Algoritmo para programar tarefas a partir da espécie 1.

É importante ressaltar que o algoritmo apresentado ilustra como o sistema utiliza a lista de prioridades, juntamente com o grafo de precedências, para selecionar as tarefas a programar respeitando as restrições de precedência. Para que fique ainda mais claro, a seqüência de figuras abaixo (da 8 a 12), ilustra passo a passo o funcionamento deste algoritmo, para um determinado cromossomo.

Tarefa	**A**	**B**	**C**	**D**	**E**	**F**
Prioridade	**5**	**1**	**2**	**6**	**4**	**3**

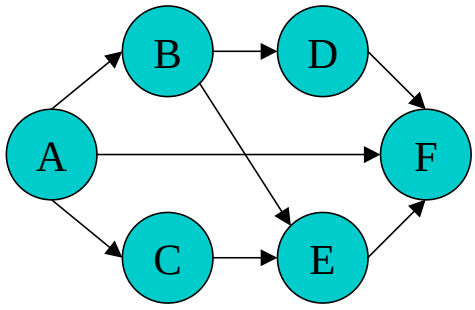


Figura 8 – Passo 1: Seleciona a atividade A (única sem predecessores).

Tarefa	A	B	C	D	E	F
Prioridade	5	1	2	6	4	3

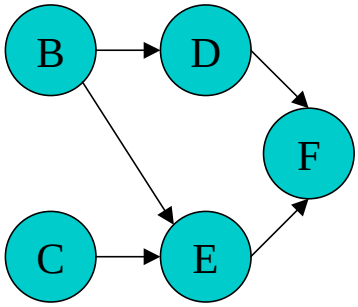


Figura 9 – Passo 2: Seleciona a atividade B (entre B e C, B vem primeiro na ordem de prioridades).

Tarefa	A	B	C	D	E	F
Prioridade	5	1	2	6	4	3

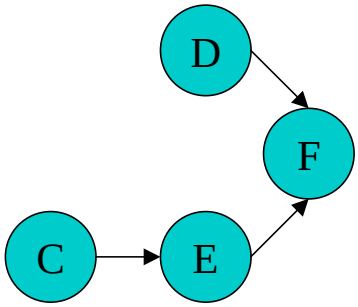


Figura 10 – Passo 3: Seleciona a atividade C (entre D e C, C vem primeiro na ordem de prioridades).

Tarefa	A	B	C	D	E	F
Prioridade	5	1	2	6	4	3

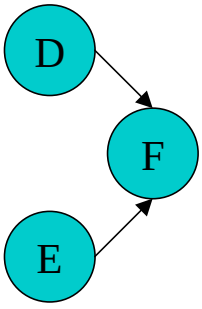


Figura 11 – Passo 4: Seleciona a atividade E (entre D e E, E vem primeiro na ordem de prioridades).

Tarefa	A	B	C	D	E	F
Prioridade	5	1	2	6	4	3

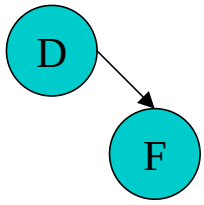


Figura 12 – Passo 5: Seleciona a atividade E (única de predecessor), e assim por diante.

Nestes cinco passos, foi mostrado como o algoritmo utiliza as duas estruturas de dados existentes na primeira espécie para selecionar as atividades numa ordem que não viole as restrições de precedência existentes no problema.

A segunda espécie, chamada de "Alocação de Recurso" conta com um vetor de listas. Cada posição do vetor representa uma tarefa, enquanto que cada lista contém os recursos que podem executá-la. A figura 13 apresenta esta estrutura.

Tarefa	A	B	C	D	E	F
Recurso	5	1	2	3	7	3
	1	4		4	2	5
	8			7		6
				9		

Figura 13 – Representação da espécie "Alocação de Recurso".

Para definir qual recurso alocar utilizando esta estrutura de vetor de listas da espécie 2, o algoritmo, para cada atividade, tenta alocar o recurso na ordem em que ele aparece na lista, se for possível. Se não, tenta alocar o próximo.

Por exemplo, se a atividade A fosse “carga na UDA”, os tanques 1, 5 e 8 são os únicos que têm possibilidade de dar carga na unidade. E, neste cromossomo, o tanque 5 foi escolhido para executar esta tarefa. No caso de uma manutenção programada para o tanque 5 num intervalo de tempo coincidente com a carga, por exemplo, o algoritmo tenta alocar o próximo.

Percebe-se que, com esta estrutura, juntamente com o grafo de precedências, é possível programar uma tarefa respeitando todas as restrições do problema. Por exemplo, ao se programar uma transferência para atender à demanda de um item de venda, o tanque só é selecionado se tiver volume de produto suficiente, se não estiver em manutenção e estiver disponível (sem nenhuma outra atividade no momento) e se, desde a última vez que recebeu um produto, já teve tempo suficiente para a preparação.

4.3. Decodificação.

A decodificação da solução é feita combinando-se as informações de cada uma das espécies. Abaixo pode-se ver a essência do algoritmo em pseudocódigo.

```
procedure Decodifica
begin
  cromossomo 1 ← indivíduo “Alocação de Tempo” selecionado
  cromossomo 2 ← indivíduo “Alocação de Recurso” selecionado
  while (existem tarefas a planejar)
  begin
    seleciona tarefa usando cromossomo 1
    seleciona recurso para tarefa usando cromossomo 2
    programa tarefa
  end while
end
```

Figura 14 – Rotina de decodificação.

Pode-se perceber que, para decodificar e formar uma solução completa, é preciso selecionar um indivíduo de cada espécie. Esta seleção chama-se *colaboração*. Existem vários métodos de seleção de colaboradores, como descrito

na seção 3.3.1.1. Para este trabalho foi escolhido o método otimista, onde se atribui à avaliação do indivíduo o valor da avaliação da melhor colaboração feita por ele.

4.4. Avaliação.

A função de avaliação deve incorporar cada um dos objetivos do *scheduling*, ou seja:

- **Atender à demanda dos produtos**. Os itens de venda ou envio devem ser atendidos sem atraso, pois são eles que determinam a receita da refinaria;
- **Minimizar os custos de produção.** Este objetivo pode ser separado em dois: minimizar o custo da matéria-prima processada e minimizar o custo operacional. O custo da matéria-prima pode ser calculado a partir das quantidades de determinados petróleos que serão destilados durante o horizonte de programação. Já os custos operacionais estão relacionados à forma de operar a planta, evitando trocas freqüentes de campanhas e evitando trocas de tanques numa mesma atividade;
- **Atender, sem desvio, às especificações dos produtos**. Os itens de venda, para poderem ser entregues, devem estar especificados, ou seja, suas propriedades devem ter valores dentro dos limites mínimos e máximos aceitos para o produto a ser vendido;

Estes objetivos devem ser quantificados para ser possível chegar a um valor numérico de avaliação do indivíduo.

O custo por não atendimento da demanda pode ser quantificado na forma de uma multa por atraso na entrega dos produtos (dos itens de envio ou venda). Para que esta multa fique proporcional à importância deste atraso em relação ao atraso de outros itens, a abordagem utilizada foi se calcular o preço da parcela do item atrasada, ou seja:

$$C_{AD} = \sum_{i \in IV} preço produto_i * volume não entregue_i \qquad \textbf{(4.1)}$$

Onde *IV* é o conjunto dos itens de venda do cenário.

Para se calcular o custo de matéria-prima, basta multiplicar o preço dos petróleos utilizados na destilação pela quantidade processada. Ou seja:

$$C_{MP} = \sum_{j \in UDA} \sum_{i \in C} preço_i * volumeprocessado_{i,j} \quad \textbf{(4.2)}$$

Onde *UDA* é o conjunto das unidades de destilação e *C* é o conjunto dos petróleos crus utilizados na refinaria.

Os custos operacionais considerados nesta dissertação compreendem apenas o número de trocas de tanques ou esferas durante uma mesma atividade ou item. Isto porque foi admitido que o número de trocas de campanha é fixo por cenário e conhecido *a priori*. Então este custo pode ser descrito como:

$$C_{OP} = \sum_{i \in AT} trocas_i * custotroca \quad \textbf{(4.3)}$$

Onde *AT* é o conjunto das atividades de transferências, e o *custotroca* é uma constante que representa o custo de uma troca de tanque numa transferência.

O custo por desvio de especificação foi quantificado de uma forma muito simples, que visa a levar em conta o impacto gerado pela não especificação de um produto que deve ser vendido. Ao não atender às especificações de um produto, este não pode ser vendido, então pode-se deixar de computar o valor da parcela do item de venda que não pôde ser vendida por estar fora de espcificação. Ou seja:

$$C_{DE} = \sum_{i \in IV} preçoproduto_i * volumenãoespecificado_i \quad \textbf{(4.4)}$$

Ao final tem-se a seguinte equação para a função objetivo (FO):

$$FO = w_1 C_{AD} + w_2 C_{MP} + w_3 C_{OP} + w_4 C_{DE} \quad \textbf{(4.5)}$$

Onde os w_i são os pesos utilizados para ajustar os objetivos, já que alguns deles são bem mais importantes que outros. Por exemplo, atender à demanda com produtos dentro dos limites de especificação é bem mais importante que minimizar o número de trocas, que por sua vez é bem mais importante que minimizar o custo de matéria-prima, já que os três primeiros podem levar a uma

programação inviável na prática e o último pode apenas reduzir a margem de lucro da refinaria.

4.5. Operadores.

Os operadores utilizados nos indivíduos de ambas as espécies são operadores de problemas de ordem (como o problema do caixeiro viajante, por exemplo). É importante ressaltar que, na espécie "Alocação de Recurso", os operadores atuam nos alelos dos genes, ou seja, na lista de recursos de cada atividade, mas não entre atividades.

Os operadores de crossover utilizados foram: Crossover de Ordem ou *Order Crossover* (OX), Crossover de Mapeamento Parcial ou *Partially Mapped Crossover* (PMX), Crossover de Ciclo ou *Cycle Crossover* (CX).

Os operadores de mutação utilizados foram: Mutação por Troca ou *Swap* (SM), Inversão de Posição ou *Position Inversion* (PI).

Estes operadores são apresentados e explicados em maiores detalhes por Michalewicz (1996), onde são aplicados a vários problemas, inclusive problemas de *scheduling*.

4.6. Representação do Tempo.

Como exposto por Moro (2000), a variável *tempo* assume importância primordial em problemas de programação da produção. E, tendo em vista a representação das atividades apresentada anteriormente, pode-se perceber que a forma de dividir e representar o tempo irá definir o número de atividades de cada tipo a serem planejadas. Isto porque, ao contrário do problema apresentado por Cruz (2003), onde a programação de descarga de minério em um porto é modelada, e os lotes do produto são acompanhados do início ao final do processo, tem-se aqui um problema contínuo, onde as unidades de processo estão operando ininterruptamente e, portanto, os momentos de tomada de decisão não ficam tão claros.

Por exemplo, se o horizonte de programação é de sete dias, durante todo este tempo a unidade de destilação tem que estar sendo alimentada com petróleo a

uma determinada vazão. Quantas cargas diferentes (a partir de tanques diferentes) serão feitas nesta unidade de processo?

Normalmente, fatias de tempo com duração uniforme são definidas ao longo do horizonte da programação, como na abordagem apresentada por Smania (2002). Nesta abordagem, o número e a duração destas fatias é arbitrado, de modo que as decisões relevantes ocorram na fronteira entre duas fatias. Ou seja, dependendo da granularidade exigida pelo problema, um grande número de fatias de tempo pode ser gerado, tornando computacionalmente inviável a obtenção de uma solução ótima.

Isso é particularmente importante quando existem, em conjunto com operações que demandam longos períodos de tempo, outras operações que são executadas rapidamente, o que faz com que a duração das fatias de tempo tenha que ser suficientemente pequena para poder acomodar essas últimas. Na verdade, este é o caso da programação da produção de petróleo (recebimento de cru e carga na destilação), pois existem operações que demandam dias para serem completadas, como é o caso de um tanque alimentar a unidade de destilação, e outras que duram apenas alguns minutos, como é o caso do recebimento de uma interface entre dois diferentes crus do oleoduto. Em uma refinaria típica, o tempo decorrido entre duas tarefas como o recebimento de um item de cru ou a troca do tanque que alimenta a unidade de destilação atmosférica está, em geral, na faixa de várias horas a alguns dias, enquanto que a segregação de uma interface toma somente cerca de quinze minutos (Moro, 2000).

O esquema adotado nesta dissertação é o de dividir (discretizar) *não* o tempo, mas sim as *quantidades* transferidas em cada uma das atividades. Para isso é adotado um parâmetro chamado **volume máximo transferido**, para certas atividades. Este parâmetro determina uma espécie de “tamanho de lote” para as atividades de transferência. É uma solução que permite ao usuário configurar em que tipos de atividade ele precisa de uma granularidade mais fina, e onde isto não é necessário. Por exemplo, no caso de recebimento e carga de petróleo, citado anteriormente, poderia ser definido como **volume máximo** para a atividade de recebimento de item o volume da interface de dois itens subseqüentes (algo em torno de 1.000 m^3, por exemplo), e ser mantido um volume maior (como, por exemplo, 20.000 m^3) para a carga na unidade. Note que, da mesma forma que lotes pequenos demais podem inviabilizar computacionalmente a otimização,

lotes muito grandes podem criar cenários sem soluções possíveis. Por exemplo, se for definido um recebimento de item com volume de 50.000 m3, e nenhum tanque, sozinho, tiver este espaço disponível, o problema será inviável.