4 Implementação

4.1. Estrutura de Dados

Lustig (1990) investiga a influência das linguagens de computador em uma implementação do algoritmo do Simplex para Redes. Suas conclusões mostram que implementações baseadas em endereçamento (ponteiros e listas encadeadas) são mais eficientes do que implementações baseadas em vetores e índices.

Ainda com relação à estrutura de dados, Löbel (1996) observa que a representação e a estrutura de dados em árvore do algoritmo Simplex para Redes já foram profundamente investigadas e que a maioria das implementações têm estruturas bastante similares. A estrutura desenvolvida neste trabalho, que será descrita a seguir, não difere muito de tais estruturas.

4.2. Árvore Geradora

No subitem 3.16 foi definido o seguinte pseudo-código para o algoritmo simplex para redes:

Determinar uma solução inicial
Calcular fluxos e potenciais associados a esta solução
Enquanto houver arcos elegíveis para entrar na base fazer:
Selecionar um arco para entrar na base
Determinar o arco que sai da base
Atualizar a árvore.
Fim

A estrutura de dados, na forma como foi utilizada neste trabalho, apesar de aparentemente mais complexa, representa uma maior flexibilidade na implementação e um salto de eficiência se comparada com as abordagens que utilizam vetores e índices

por evitar longas e repetidas consultas a listas e matrizes cujos tamanhos crescem proporcionalmente ao quadrado do número de arcos e nós. Voltando ao pseudo-código, pode-se dizer que os maiores ganhos apresentados pela estrutura de dados apresentada neste capítulo concentram-se nas rotinas "Determinar o arco que sai da base" e "Atualizar a árvore".

4.2.1. A Classe *Nó.*

A estrutura de árvore geradora será formada basicamente por dois tipos de classes, os nós e os arcos. Um nó qualquer da árvore será composto pelas informações referentes ao *nome* do mesmo, à *demanda*, ao *potencial* e à *profundidade*, sendo todos estes campos variáveis do tipo inteiro.

Além dos campos de informação, os nós de uma árvore geradora irão possuir ponteiros para outros nós e arcos de forma a interligar os elementos componentes da estrutura, facilitar a implementação e otimizar a execução do algoritmo Simplex para Redes. Estes ponteiros são cinco:

- *Predecessor*: Aponta para o próximo nó do caminho entre o nó e a raiz
- *Arco_Predecessor*: Aponta para o arco que liga o nó ao nó predecessor
- *Caminho*: Aponta para o próximo nó do caminho conforme definido no subitem 3.11.3.
- *Caminho_Inverso*: Aponta para o nó anterior do caminho conforme definido no subitem 3.11.3.
- *Arcos_Sucessores*: Aponta para o primeiro arco da lista dos arcos que ligam o nó aos seus nós filho.

Da maneira como foram definidos os ponteiros *caminho* e *caminho_inverso*, pode-se dizer que, a partir destes, os nós da rede formam uma lista circular duplamente encadeada.

4.2.2. A Classe *Arco.*

Os arcos de uma árvore geradora, a exemplo dos nós, são compostos por campos de informação e por ponteiros, sendo o primeiro grupo composto pelos campos *nome, custo, custo_ reduzido, fluxo, capacidade* e *pertinência*. O campo *pertinência* de um arco admite os valores -1, 0 ou 1 nos casos de o arco pertencer aos conjunto **L**, **T**, ou **U** respectivamente (ver subitem 3.8)

Os ponteiros de um arco são em número de quatro. Os dois primeiros, *próximo_irmao* e *irmão_anterior*, apontam para o próximo arco e para o arco anterior da lista de arcos que ligam o mesmo nó aos seus nós filho formando uma lista duplamente encadeada. Os demais ponteiros são *origem* e *destino* que, como o nome já sugere, apontam para os nós origem e destino do arco.

Além destas duas classes, algumas estruturas auxiliares foram criadas de modo a melhorar a eficiência do programa, a saber:

- *gLista_de_Nós*: Variável global que lista todos os nós da rede.
- *gLista_de_Arcos*: Variável global que lista todos os arcos da rede.
- *gLista_de_Candidatos*: Variável global que, a cada iteração maior (ver subitem 3.12.4), lista candidatos a entrar na árvore.
- *conjunto_S*: Classe que será utilizada nos procedimentos de atualização da árvore. Esta classe será discutida com maiores detalhes mais adiante no item 4.3.

Seja, portanto, o exemplo de árvore geradora bastante simples dado pela figura 4.1, composta por três nós e dois arcos além do arco raiz.

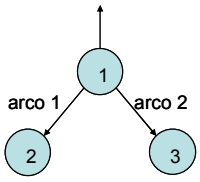


Figura 4.1– Exemplo de árvore geradora.

Agora podem ser utilizadas as estruturas de dados apresentadas para representar a árvore da figura 4.1. Optando, no intuito de simplificar a representação, por não incluir o arco raiz e ainda suprimindo os campos de informação (com exceção dos campos *nome* dos objetos), obtém-se algo parecido com o mostrado na figura 4.2:

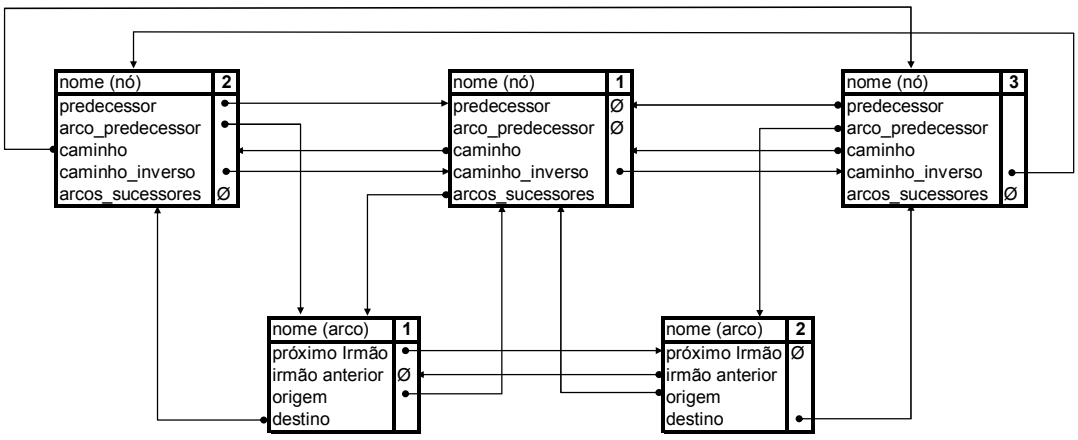


Figura 4.2 – Representação de uma árvore geradora - Exemplo.

O símbolo ϕ está sendo usado para representar um ponteiro nulo.

4.3. Os Conjuntos S

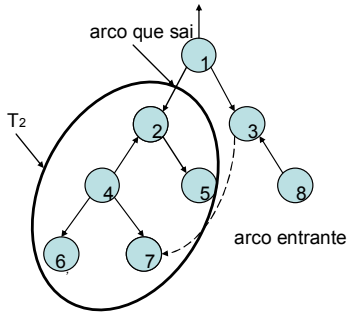
Na atualização da árvore será utilizada a classe *Conjunto_S* que será utilizada para realizar a divisão em subconjuntos dos nós pertencentes ao conjunto T_2 conforme foi mostrado no subitem 3.14.1. Esta classe, para um dado nó pertencente a T_2, é composta da seguinte maneira:

- *Nó_Conjunto_S*: Ponteiro para o nó.
- *Próximo_Conjunto_S*: Ponteiro para o próximo elemento do conjunto atual ou para o primeiro elemento do próximo conjunto.
- *Conjunto_S_Anterior*: Ponteiro para o elemento anterior do conjunto atual ou para o último elemento do conjunto anterior
- *Número_Conjunto_S*: Número do conjunto atual

O conjunto *S* foi, portanto, implementado como uma lista duplamente encadeada de objetos onde todos os elementos de T_2 são alocados em sub conjuntos (S_1, S_2, ..., S_n) determinados pelos valores do campo *Número_Conjunto_S*.

4.4. Atualização dos índices da árvore

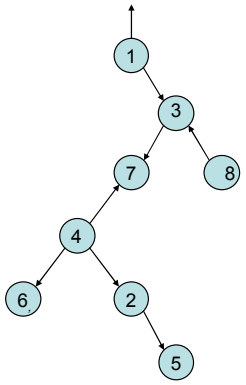
A utilização das estruturas apresentadas no algoritmo Simplex para Redes será demonstrada com maior clareza através das atualizações dos índices da árvore geradora: *predecessor, caminho* e *profundidade*. Para tal será considerado o exemplo dado pela figura 4.3, aonde a solução dada pela figura 4.3a (índices dados pela figura 4.3b), é modificada através da entrada do arco (3,7) e saída do arco (1,2) resultando na árvore da figura 4.3c com índices dados pela figura 4.3d.



(a)

i	1	2	3	4	5	6	7	8
predecessor(i)	0	1	1	2	2	4	4	3
profundidade(i)	0	1	1	2	2	3	3	2
caminho(i)	2	4	8	6	3	7	5	1

(b)



(c)

i	1	2	3	4	5	6	7	8
predecessor(i)	0	4	1	7	2	4	3	3
profundidade(i)	0	4	1	3	5	4	2	2
caminho(i)	3	7	4	6	2	5	8	1

(d)

Figura 4.3 – Atualização dos índices de uma árvore (exemplo prático)**.**

No processo de atualização de tais índices serão utilizados os Conjuntos S, conforme já foi mostrado nos itens 3.14.2 a 3.14.4. Para o exemplo dado pela figura 4.3 serão três conjuntos S, a saber:

$S_1 = \{7\}$

$S_2 = \{4, 6\}$

$S_3 = \{2, 5\}$

Estes conjuntos concatenados geram o vetor $S^* = \{7, 4, 6, 2, 5\}$ que pode ser logicamente representado por uma lista duplamente encadeada de objetos do tipo *Conjunto_S*, conforme se vê na figura 4.4.

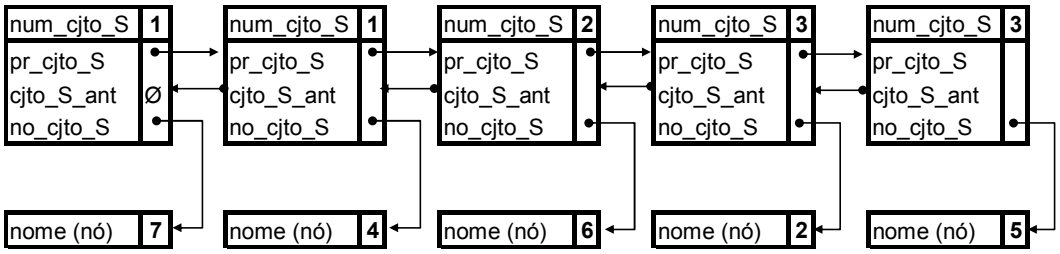


Figura 4.4– Representação lógica dos Conjuntos S

4.4.1. Atualização dos índices caminho.

Utilizando a estrutura de dados da maneira como foi apresentada torna-se bastante simples a atualização dos índices *caminho* seguindo a dinâmica apresentada na figura 3.13 (item 3.14.2). Contudo, antes de prosseguir, é interessante que seja definida a maneira de se referir a um campo específico de um objeto. Sendo assim seja *X* um objeto do tipo *nó*, para se referir ao campo *nome* deste objeto será utilizada a sintaxe *X.nome*. Da mesma maneira, o campo *nome* do nó predecessor ao nó *X* será *X.predecessor.nome* e o objeto *X.caminho.caminho_inverso* será o próprio nó *X*.

Para realizar a atualização dos índices *caminho* primeiramente são encontrados os seguintes nós:

- *X* = Primeiro elemento da lista de *conjuntos_S* (no exemplo, o conjunto que aponta para o nó 7).
- *Y* = Último elemento da lista de *conjuntos_S* (no exemplo, o conjunto que aponta para o nó 5).
- Z_1 = Extremidade do arco entrante pertencente a T_1 (no exemplo, o nó 3).
- Z_2 = Extremidade do arco entrante pertencente a T_2 (no exemplo, o nó 7).
- K_1 = Antecessor ao primeiro nó do caminho pertencente a T_2 (no exemplo, o nó 1)
- K_2 = Sucessor do último nó do caminho pertencente a T_2 (no exemplo, o nó 3)

Pode-se, agora, seguir os passos definidos na figura 3.13 do item 3.14.2. O primeiro passo é, portanto, ligar os nós K_1 e K_2 da seguinte maneira:

$K_1.caminho = K_2$
$K_2.caminho\_inverso = K_1$

O segundo passo é posicionar corretamente o vetor S^* que será inserido entre o nó Z_1 e o seu atual sucessor no caminho da seguinte maneira:

$Z_1.caminho.caminho\_inverso = Y.nó\_conjunto\_S$
$Y.\ nó\_conjunto\_S.caminho = Z_1.caminho$
$Z_1.caminho = X.nó\_conjunto\_S$
$X.nó\_conjunto\_S.caminho\_inverso = Z_1$

Para finalizar, basta interligar corretamente os nós pertencentes ao próprio vetor S^*, o que será feito através do *loop*:

Conjunto_S_temp = X
Até que Conjunto_S_temp seja igual a ϕ *fazer:*
Conjunto_S_temp.nó_conjunto_S.caminho = Conjunto_S_temp.proximo_conjunto_S.nó_conjunto_S
Conjunto_S_temp.proximo_conjunto_S.nó_conjunto_S.caminho_inverso = Conjunto_S_temp.nó_conjunto_S
Conjunto_S_temp = Conjunto_S_temp.proximo_conjunto_S
Fim

4.4.2.Atualização dos índices profundidade.

A atualização dos índices *profundidade* é bastante simples, sendo feita através da expressão (36): $\theta_h = \theta_1 + 2(h-1)$, demonstrada no item 3.14.3, cujo valor para um determinado nó dependerá do número do conjunto S em que estará contido, dado pelo valor de *h*.

O primeiro passo executado pela rotina que faz a atualização destes índices será o cálculo do θ_1, que, da mesma maneira como foi definido no item 3.14.3, havendo um arco entrante (Z_1, Z_2) com $Z_1 \in T_1$ e $Z_2 \in T_2$, corresponderá ao valor dado pela expressão: $\theta_1 = profundidade(Z_1) - profundidade(Z_2) + 1$.

A partir daí aplica-se o algoritmo abaixo:

Conjunto_S_temp = X
Até que Conjunto_S_temp seja igual a ϕ fazer:
Conjunto_S_temp.nó_conjunto_S.Profundidade = θ_1 + 2(Conjunto_S_temp.Num_cjto_S – 1)*
Conjunto_S_temp = Conjunto_S_temp.proximo_conjunto_S
Fim

4.4.3.Atualização dos índices predecessor.

Para a atualização dos índices *predecessor* o algoritmo percorrerá o vetor S^* e só atualizará tais valores para o primeiro elemento de cada conjunto S_h. O primeiro elemento do conjunto S_1, aqui denominado X, terá como predecessor a extremidade do arco entrante pertencente a T_1, mais exatamente, Z_1 enquanto os demais nós serão atualizados segundo o algoritmo a seguir:

Conjunto_S_temp = X
Conjunto_S_temp2 = X.proximo_conjunto_S
Até que Conjunto_S_temp2 seja igual a ϕ fazer:
Se *Conjunto_S_temp2*.Num_cjto_S > *Conjunto_S_temp*.Num_cjto_S fazer:
Conjunto_S_temp2. nó_conjunto_S.predecessor = Conjunto_S_temp. nó_conjunto_S
Conjunto_S_temp = Conjunto_S_temp2
Fim
Conjunto_S_temp2 = Conjunto_S_temp2. proximo_conjunto_S
Fim

4.5. O Programa

O programa foi implementado na versão 6.3 do Microsoft Visual Basic em uma planilha do Microsoft Excel versão 2002, e consiste em um arquivo de nome *"Network Simplex.xls"* contendo duas abas.

4.5.1. Dados de Entrada

A primeira das abas foi chamada de *"Dados de Entrada"* e possui campos para a entrada dos dados que caracterizam a rede além das opções relativas aos parâmetros utilizados na seleção dos arcos entrantes (item 3.12.4) e de um botão que inicia o algoritmo Simplex para Redes. A figura 4.5 mostra a interface de entrada da planilha.

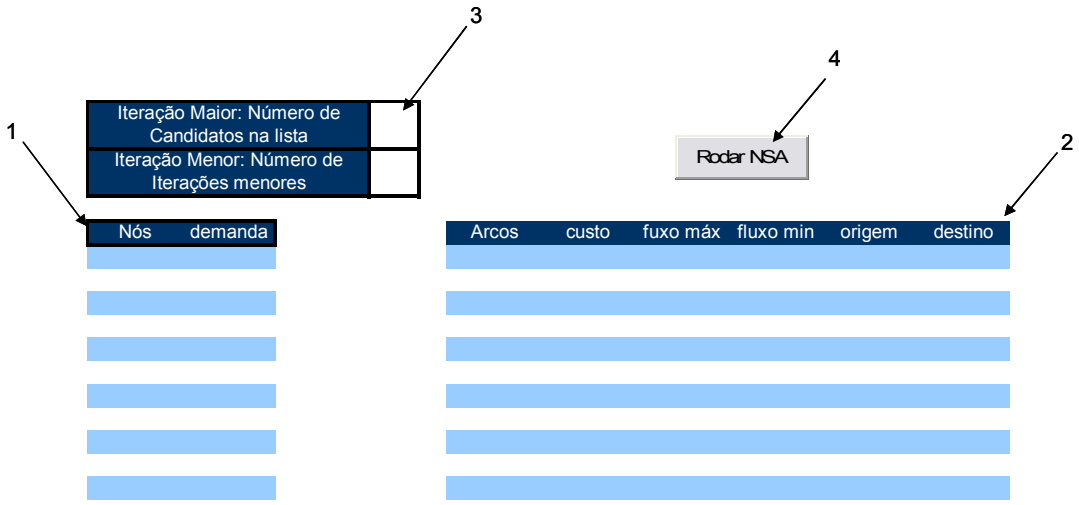


Figura 4.5 – Dados de Entrada

Os campos da figura indicados pelos números 1 e 2 seriam, portanto, aqueles destinados aos dados físicos da rede. As colunas abaixo dos itens *Nós* e *Arcos* deverão ser numerados em ordem crescente, enquanto os demais campos (*demanda*, *custo*, *fluxo máx*, *fluxo min*, *origem*, *destino*) deverão ser preenchidos com os dados referentes a cada nó ou arco lembrando que, conforme foi dito no item 2.1, a demanda é dada pela dotação inicial de um nó e que, neste sentido, uma demanda positiva implica um nó ofertante.

Quanto à capacidade máxima do programa em termos de tamanho da rede não há teoricamente limites, ainda que não tenham sido feitos testes com redes muito extensas.

Os dois campos indicados pelo número 3 referem-se às opções quanto ao número máximo de arcos não básicos que irão integrar a lista de candidatos definida a cada iteração maior e o número máximo de iterações menores a serem realizadas a cada iteração maior. É sabido que a melhor estratégia quanto à busca por candidatos a integrar a base depende de características particulares da rede subjacente, para o exemplo mostrado pela figura 4.6 foi realizado um teste em que se verificou o tempo de execução de diferentes estratégias.

Nós	demanda
1	1
2	4
3	1
4	-4
5	-1
6	3
7	3
8	-3
9	1
10	-8
11	7
12	0
13	-4

Arcos	custo	fuxo máx	fluxo min	origem	destino
1	3	1	0	1	2
2	1	5	0	1	3
3	5	3	0	1	4
4	7	6	2	2	3
5	5	5	0	2	5
6	7	4	0	3	4
7	3	9	0	3	10
8	1	5	0	4	2
9	3	4	0	4	5
10	1	3	0	4	6
11	5	5	0	5	4
12	7	5	0	5	6
13	7	3	0	5	7
14	1	2	0	5	8
15	5	4	0	6	3
16	3	2	0	6	9
17	3	4	0	7	8
18	3	3	0	8	11
19	1	3	0	9	6
20	3	4	0	9	8
21	1	3	0	9	13
22	5	2	0	10	9
23	7	3	0	11	4
24	7	4	0	11	13
25	1	4	0	12	7
26	5	2	0	12	11
27	1	2	0	12	13
28	2	3	0	13	9

Figura 4.6 – Exemplo

Sejam p_1 e p_2, respectivamente, os parâmetros referentes ao número máximo de candidatos em cada iteração maior e o número máximo de iterações menores. A rotina escrita para executar o teste pode ser representada pela função seguinte:

Comparar estratégias de precificação
Para p_1 de 1 a 10 fazer

Para p_2 de 1 a p_1 fazer
Início = hora
Executar Algoritmo 500 vezes
Final = hora
Tempo = Final - Início
Fim

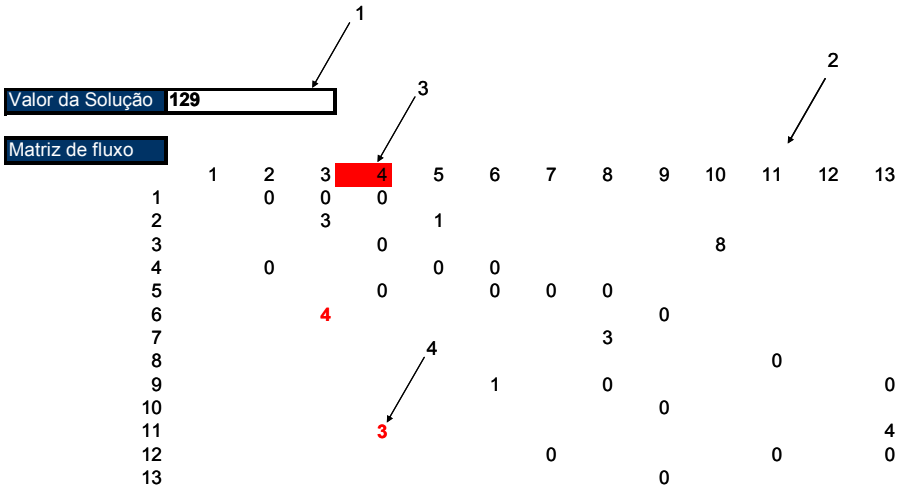
Após a rotina ter sido executada algumas vezes para que fossem expurgados efeitos aleatórios referentes a instabilidades no sistema ou ao uso de memória do computador, foi constatado que, para o exemplo da figura 4.6, que a melhor estratégia seria correspondente a $p_1 = 2$, $p_2 = 1$, 12 iterações menores e 13 iterações maiores, com execução 32% mais rápida do que a pior estratégia ($p_1 = 2$, $p_2 = 2$, 22 iterações menores e 12 iterações maiores).

O teste foi completado em 18,5 minutos tendo sido executadas 441.500 iterações menores (cerca de 400 por segundo) e 221.500 iterações maiores.

O último item apontado na figura 4.5 vem a ser o botão onde se lê *"Rodar NSA"* que além de rodar o Simplex para Redes chama as rotinas *"Matriz Saída"* e *"Verifica Desequilíbrio"* responsáveis, respectivamente, pela saída do programa e pela execução de alguns testes de consistência conforme será mostrado nos próximos itens.

4.5.2. Saída

Na planilha *"Network Simplex.xls"* a aba *Saída* possui os campos, indicados na figura 4.7 pelos números 1 e 2, *Valor da Solução* e *Matriz de Fluxo* que indicam o custo referente à solução ótima e a matriz de fluxo *nxn* onde *n* é o número de nós e cujos elementos, $a_{i,j}$, representam o fluxo entre os nós *i* e *j*.



	1	2	3	4	5	6	7	8	9	10	11	12	13
1		0	0	0									
2			3		1								
3				0						8			
4		0			0	0							
5				0		0	0	0					
6			4						0				
7								3					
8											0		
9						1		0					0
10									0				
11				3									4
12							0				0		0
13									0				

Figura 4.7– Saída

Desta maneira, para o exemplo dado pela figura 4.7, na solução ótima, o fluxo no arco com origem no nó dois e destino no nó três será igual a três unidades.

Após o preenchimento da matriz fluxo, o programa executa uma verificação que identifica e assinala quais arcos estão em sua capacidade máxima (indicado pelo número quatro na figura 4.7) e quais nós eventualmente não tiveram a sua demanda/oferta atendida/despachada (indicado pelo número 3). Este último passo é executado pela função *"Verifica Desequilíbrio"* e facilita a identificação de gargalos e/ou falhas na rede, bem como um descasamento entre oferta e demanda.

4.6. Um Exemplo: O escoamento da produção nacional de açúcar.

Reis (2007) estuda o transporte da produção nacional de açúcar destinada à exportação. Para tal foram efetuadas projeções para o volume total do produto a ser exportado nos períodos de 2007/08 a 2013/14, volume este atribuído a dez pontos centrais que concentrariam a produção em suas respectivas regiões e encerrariam a parte de maior relevância da produção nacional. Os pontos escolhidos foram Campos de Goytacazes (RJ), Itapemirim (ES), Monte Belo (MG), Santa Helena de Goiás (GO), Ourinhos (SP), Santa Adelis (SP), Pradópolis (SP), Nova Olímpia (MS), Nova Andradina (MT) e Maringá (PR)

Reis (2007) faz ainda um levantamento das alternativas disponíveis para o transporte do açúcar produzido em cada um destes pontos centrais até os principais portos exportadores do país, a saber, os portos de Santos, Paranaguá, Imbituba, Itajaí, Murtinho, Vitória, Rio de Janeiro e São Francisco bem como dos custos relativos a cada opção que, já contabilizando os custos de transbordo geraram a seguinte tabela em R$/TU:

		Santos	Paranaguá	Imbituba	Itajaí	Murtinho	Vitória	Rio	S. Francisco
Campos de Goytacazes	Ferrovia								
	Rodovia	79	84	98	88	143	34	37	87
	Bimodal					265			
Itapenirim	Ferrovia								
	Rodovia	79	93	107	96	152	18	51	95
	Bimodal					267			
Monte Belo	Ferrovia								
	Rodovia	59	77	70	74	97	70	57	74
	Bimodal	55	119	119	119	156	84	50	119
Sta Helena de Goiás	Ferrovia								
	Rodovia	73	85	99	89	69	82	78	87
	Bimodal	101	166	200	179	140	125	85	165
Ourinhos	Ferrovia	40	38						
	Rodovia	63	64	73	68	62	97	83	66
	Bimodal			75	54	98			
Santa Adélia	Ferrovia	36	53						
	Rodovia	66	81	77	80	72	93	82	83
	Bimodal			91	70	101			
Pradópolis	Ferrovia	34	51						
	Rodovia	59	78	71	77	79	61	78	77
	Bimodal			91	93	99			55
Nova Olímpia	Ferrovia								
	Rodovia	130	132	146	136	83	140	149	134
	Bimodal								
Nova Andradina	Ferrovia								
	Rodovia	79	78	72	77	63	127	94	77
	Bimodal								
Maringá	Ferrovia	42	40						
	Rodovia	81	66	75	70	71	117	85	68
	Bimodal			75	55	63			40

Figura 4.8– Matriz de custos para o transporte de açúcar

A partir destes dados já se pode montar uma rede tendo como nós ofertantes os pontos produtores de açúcar e como nós demandantes os portos exportadores, de maneira que o volume ofertado por um ponto seja igual à produção a ser exportada e o volume demandado por um porto seja igual à sua capacidade de processamento daquele produto. Repare que, desta maneira, um porto cuja demanda não seja integralmente atendida estará operando com capacidade ociosa enquanto um porto cuja demanda estiver sendo totalmente atendida será um gargalo, não podendo suportar qualquer aumento na produção.

Podem ser utilizados, a título de exemplo, os dados de capacidade atuais dos portos e os volumes de exportação projetados para 2013/14 no intuito de se inferir quanto às

eventuais necessidades de investimento em expansão da capacidade dos portos. Sendo assim considerem-se os dados abaixo apresentados por Reis (2007):

Volume de Exportação para 2013/14 (mil toneladas)	
Campos de Goytacazes	328
Itapenirim	55
Monte Belo	351
Sta Helena de Goiás	1732
Ourinhos	1158
Santa Adélia	8831
Pradópolis	8970
Nova Olímpia	463
Nova Andradina	477
Maringá	1982

Capacidade (mil TU)	
Santos	11200
Paranaguá	13500
Imbituba	400
Itajaí	240
Murtinho	233
Vitória	2000
Rio	4000
S. Francisco	2000

Figura 4.9 – Projeção de volume de exportação e capacidade de escoamento.

Ao rodar o Simplex para Redes para este exemplo obtém-se a seguinte saída (observe que os nós foram numerados na ordem em que foram apresentados na figura 4.9, sendo os de números 1 a 10 os pontos produtores e os de números 11 a 18 os portos exportadores):

Valor da Solução	1107613

Matriz de fluxo

	11	12	13	14	15	16	17	18
1	0	0	0	0	0	328	0	0
2	0	0	0	0	0	55	0	0
3	0	0	0	0	0	0	351	0
4	0	0	0	0	0	0	1732	0
5	0	1158	0	0	0	0	0	0
6	2230	6601	0	0	0	0	0	0
7	8970	0	0	0	0	0	0	0
8	0	230	0	0	233	0	0	0
9	0	0	400	77	0	0	0	0
10	0	1982	0	0	0	0	0	0

Figura 4.10 – Saída do programa

Lembrando que os nós destacados em vermelho denotam demanda não satisfeita (excesso de capacidade), pode-se observar que, apesar de existir uma capacidade ociosa no sistema como um todo (a capacidade dos portos excede a produção em 9.226 mil toneladas), alguns portos (porto de Santos, de Imbituba e Murtinho) estarão operando no limite de sua capacidade caso as previsões de crescimento das exportações se confirmem. Este fato pode significar um custo global maior e, conseqüentemente, perda de

competitividade do açúcar nacional frente aos concorrentes internacionais. Neste sentido, o resultado obtido sugere a necessidade de se investir na capacidade de tais portos ou alternativamente, baratear o acesso aos portos que apresentam capacidade ociosa investindo em novas ferrovias ou integrando as já existentes, diminuindo assim a necessidade de transbordos e diminuindo a participação dos custos associados ao transporte rodoviário.