Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Instituto de Matemática e Estatística

Thiago Soares Pinheiro

Otimização do problema de localização de instalações aplicado ao comércio e distribuição de combustíveis

Rio de Janeiro
2015

Thiago Soares Pinheiro

Otimização do problema de localização de instalações aplicado ao comércio e distribuição de combustíveis

Dissertação apresentada como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado Rio de Janeiro.

Orientadora: Profª. Dra. Maria Clicia Stelling de Castro

Rio de Janeiro

2015

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC-A

P654 Pinheiro, Thiago Soares.
Otimização do problema de localização de instalações aplicado ao comércio e distribuição de combustíveis / Thiago Soares Pinheiro. – 2015.
84 f. : il.

Orientadora: Maria Clicia Stelling de Castro.
Dissertação (Mestrado em Ciências Computacionais) - Universidade do Estado do Rio de Janeiro, Instituto de Matemática e Estatística.

1. Instalações industriais - Localização - Teses. 2. Combustíveis - Brasil - Teses. 3. Logística empresarial - Brasil - Teses. 4. Modelos matemáticos - Teses. I. Castro, Maria Clicia Stelling de. II. Universidade do Estado do Rio de Janeiro. Instituto de Matemática e Estatística. III. Título.
.

CDU 658.21(81)

_____ _____

Assinatura Data

Thiago Soares Pinheiro

Otimização do problema de localização de instalações aplicado ao comércio e distribuição de combustíveis

Dissertação apresentada como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado Rio de Janeiro.

Aprovada em 11 de Março de 2015.
Banca Examinadora:

Profª. Dra. Maria Clicia Stelling de Castro (Orientadora)
Instituto de Matemática e Estatística – UERJ

Prof. Dr. Leandro Augusto Justen Marzulo
Instituto de Matemática e Estatística – UERJ

Prof. Dr. Alexandre da Costa Sena
Instituto de Matemática e Estatística – UERJ

Prof. Dr. Fabricio Alves Barbosa da Silva
Fundação Oswaldo Cruz – FIOCRUZ

Rio de Janeiro
2015

DEDICATÓRIA

Aos meus pais, por desejarem que seus filhos fossem longe. E eles foram mesmo.

AGRADECIMENTOS

A todos que colaboraram na elaboração deste trabalho, especialmente minha orientadora Clicia, pela paciência e principalmente por acreditar que eu chegaria ao final.

Ao meus colegas da Petrobras Distribuidora, em especial, ao amigo Luciano Lanna, por conseguir trabalhar o suficiente para cobrir minha ausência. Sua capacidade de resolver os problemas mais difíceis com uma tranquilidade Budista, enquanto as reclamações não paravam de chegar, servirá para sempre de inspiração para mim.

A minha esposa Luci, por ter cuidado da comida, da casa e de mim enquanto eu me debruçava em frente ao computador. Afinal, esse foi o apoio que mais precisei.

Vale mais o pouco que tem o justo,
do que as riquezas de muitos ímpios.
Bíblia Sagrada. Salmos 37,16

RESUMO

PINHEIRO, Thiago Soares. *Otimização do problema de localização de instalações aplicado ao comércio e distribuição de combustíveis*. 2015. 85 f. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado Rio de Janeiro, Rio de Janeiro, 2015.

Um dos problemas mais relevantes em organizações de grande porte é a escolha de locais para instalação de plantas industriais, centros de distribuição ou mesmo pontos comerciais. Esse problema logístico é uma decisão estratégica que pode causar um impacto significativo no custo total do produto comercializado. Existem na literatura diversos trabalhos que abordam esse problema. Assim, o objetivo desse trabalho é analisar o problema da localização de instalações proposto por diferentess autores e definir um modelo que seja o mais adequado possível ao mercado de distribuição de combustíveis no Brasil. Para isso, foi realizada uma análise do fluxo de refino e distribuição praticado neste segmento e da formação do respectivo custo de transporte. Foram consideradas restrições como capacidade de estoque, gama de produtos ofertados e níveis da hierarquia de distribuição. A partir dessa análise, foi definido um modelo matemático aplicado à redução dos custos de frete considerando-se a carga tributária. O modelo matemático foi implementado, em linguagem C, e permite simular o problema. Foram aplicadas técnicas de computação paralela visando reduzir o tempo de execução do algoritmo. Os resultados obtidos com o modelo *Single Uncapacited Facility Location Problem* (SUFLP) simulado nas duas versões do programa, sequencial e paralela, demonstram ganhos de até 5% em economia de custos e redução do tempo de execução em mais de 50%.

Palavras-chave: Problema da localização de instalações. Logística de combustíveis. Otimização logística.

ABSTRACT

PINHEIRO, Thiago Soares. *An algorithm for the plant location problem optimization applied to oil and gas Logistics*. 2015. 85 f. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado Rio de Janeiro, Rio de Janeiro, 2015.

One of the most relevant problems at large organizations is the choice of locations for estabilishing facilities, distribution centers or retail stores. This logistics issue is an strategic decision which may cause significant impact at the effective cost of the product. There are several papers tackling this issue. The objective of this dissertation is to analyze the Facility Location Problem previously developed by other authors and to define the most applicable model to the fuel distribution industry in Brazil. It started from the analyzis of the upstream and downstream flow in practice at this segment and from the respective transportation cost formation. Some constraints were considered like stock capacity, multicommodity offer and distribution hierarchy levels so it could be possible to define a mathematical model applied to freight economy considering the incident taxes. The dissertation also has the objective of creating a program using the C language which could simulate the problem. It were applied parallel computing techniques to reduce runtime of the algorithm. The results obtained from the *Single Uncapacited Facility Location Problem* (SUFLP) executed in both program versions, sequential and parallel, demonstrate up to 5% of effective costs gain and reduction of more than 50% in execution time.

Keywords: Plant location problem. Oil and gas logistics. Logistics optimization.

LISTA DE ILUSTRAÇÕES

Figura 1 - Configuração da rede de distribuição de combustíveis. 17
Figura 2 - Estrutura do arquivo de cidades 34
Figura 3 - Inicialização do ambiente MPI 38
Figura 4 - Tempo de execução do SUFLP sequencial. 45
Figura 5 - Evolução do tempo de execução do SUFLP paralelo. 45
Figura 6 - Tempo de execução do SUFLP com múltiplos processadores. 47
Figura 7 - *Speedup* do SUFLP com 2856 cidades. 48

LISTA DE ABREVIATURAS E SIGLAS

ALA	Alternate Location and Allocation
CPLP	Capacited Plant Location Problem
IBP	Instituto Brasileiro de Petróleo, Gás e Biocombustíveis
IBPT	Instituto Brasileiro de Planejamento Tributário
ICMS	Impostos Sobre Circulação de Mercadorias e Serviços
ISS	Impostos Sobre Serviços de Qualquer Natureza
MPI	Message Passing Interface
PLI	Problema da Localização de Instalações
SINDICOM	Sindicato Nacional das Empresas Distribuidoras de Combustíveis
SUFLP	Simple Uncapacited Facility Location Problem

SUMÁRIO

	INTRODUÇÃO	12
1	**A LOGíSTICA DE DISTRIBUIÇÃO DE COMBUSTíVEIS NO BRASIL**	15
1.1	**Esquema de Refino e Distribuição**	16
2	**TEORIA SOBRE LOCALIZAÇÃO DE INSTALAÇÕES**	18
2.1	**Localização de instalações simples sem restrição de capacidade**	19
2.2	**Localização de instalações simples sem restrição de capacidade com múltiplos níveis**	20
2.3	**Localização de instalações sem restrição de capacidade com vários produtos**	22
2.4	**Localização de Instalações com Restrição de Capacidade**	23
3	**CARGA TRIBUTÁRIA APLICADA AO SETOR DE TRANSPORTES**	25
3.1	**Imposto sobre Circulação de Mercadorias e Serviços**	25
3.2	**Imposto sobre Serviços de Qualquer Natureza**	27
4	**MODELO PROPOSTO PARA OTIMIZAÇÃO DE CUSTOS DE TRANSPORTE**	29
4.1	**Abordagens para solução do SUFLP**	32
4.2	**Algoritmo Proposto**	33
4.3	**Adaptação do algoritmo ao MPI**	37
4.3.1	*Message Passing Interface*	37
4.3.2	Algoritmo adaptado	38
5	**ANÁLISE E AVALIAÇÃO DOS RESULTADOS**	40
5.1	**Ambientes de Execução**	40
5.2	**Resultados**	40
6	**TRABALHOS RELACIONADOS**	49
	CONCLUSÕES	52
	REFERÊNCIAS	55
	APÊNDICE A – Código fonte do programa `SUFLP_simples.c`	58
	APÊNDICE B – Código fonte do programa `SUFLP_paralelo.c`	72

INTRODUÇÃO

A escolha do local para a instalação de uma planta industrial, centro de distribuição ou ponto comercial é um problema logístico de suma importância nas organizações de grande porte. Esta decisão tem cunho estratégico e pode causar impacto significativo no custo total do produto comercializado. Fatores como mão de obra, infraestrutura disponível e regulamentação ambiental são exemplos de premissas para a escolha do local para instalação. Porém, a proximidade dos fornecedores de insumos e a demanda pelos bens comercializados são critérios quantitativos para estimar o retorno do investimento.

O problema da localização de instalações não é novo para a área de Pesquisa Operacional. Para acomodar a diversidade de aplicações encontradas no mundo dos negócios, o problema foi separado em famílias. A sua complexidade pode aumentar devido a fatores tais como a oferta de uma ou mais mercadorias (*singlecommodity* ou *multicommodity*), se a disponibilidade de estoque deve ser considerada ou não (plantas capacitadas *versus* não capacitadas) e se o modelo matemático é determinístico ou probabilístico (KHUMAWALA, 1972).

Abordagens para a solução destes problemas são estudadas há décadas e incluem heurísticas, simulação e otimizadores. Em 1963, Kuehn e Hamburger desenvolveram uma heurística aplicando técnicas de programação linear e fizeram simulações utilizando um programa de computador. Os resultados já mostravam a preocupação em propor soluções para problemas envolvendo centenas de armazéns e milhares de clientes (KUEHN; HAMBURGER, 1963). Já a partir da década de 1970, podem ser encontrados na literatura diversos artigos sobre a aplicação do algoritmo *branch and bound* ao problema da localização de armazéns. Khumawala estabeleceu um conjunto de regras de decisão para percorrer uma árvore selecionando armazéns, comprovando a eficiência deste método em reduzir o tempo de computação.

Classificação dos Modelos de Localização

O problema da localização de instalações (PLI) e suas variações tem sido amplamente estudado no âmbito do gerenciamento de cadeias de suprimento (*Supply Chain Management*). Tais problemas envolvem estabelecer uma ou várias instalações de uma organização, usualmente vértices de uma rede, de modo a reduzir o custo de distribuição, facilitar o movimento de mercadorias ou otimizar a prestação de um serviço dentro desta rede. Em seu artigo, Aikens (AIKENS, 1985) analisa publicações relevantes de diversos autores de modo a ilustrar uma taxonomia e mostrar como a metodologia de pesquisa operacional influenciou no projeto de cadeias de distribuição. Neste trabalho, o autor se

concentra em descrever o modelo matemático associado a cada variação do problema e classifica sua complexidade em termos do grau da dificuldade da solução.

A formulação do PLI varia conforme as restrições impostas pelo ambiente estudado. Os exemplos mais comuns consideram aspectos como o custo associado a instalação em determinado local, a quantidade de níveis da malha logística[1], a gama de produtos ofertados e a quantidade em estoque disponível em cada unidade de distribuição.

A Carga Tributária Brasileira

O Brasil possui a maior carga tributária entre os países que compõem o BRICS (Brasil, Rússia, Índia, China e África do Sul)[2]. Segundo dados do Instituto Brasileiro de Planejamento e Tributação (IBPT) os impostos correspondem a 36% do produto interno bruto brasileiro. Este percentual é consideravelmente superior à média dos outros componentes do bloco, que em 2013 chegou a 22% (IBPT, 2012). Se considerarmos apenas o setor de transportes, a situação é agravada pela ausência de benefícios ou regimes especiais dependentes do porte da empresa. Isto significa que transportadores de pequeno porte devem contribuir do mesmo modo que os de grande porte. Além disso, o setor é influenciado por tributos das esferas municipal, estadual e federal de modo simultâneo, uma vez que os serviços podem ser prestados em toda a extensão do território nacional.

Atualmente existem 5.570 municípios distribuídos entre 26 estados brasileiros (IBGE, 2013). Ao considerarmos os combustíveis derivados de petróleo como mercadoria essencial e indispensável a qualquer cidadão, temos a noção da capilaridade deste mercado. Ainda, tendo os combustíveis derivados um baixo valor agregado em relação ao seu volume, é possível perceber a grande influência que o custo do frete exerce sobre o preço final deste produto ao consumidor. Torna-se necessário estabelecer a melhor política de atendimento, isto é, aquela que minimiza o custo logístico total. Como é operacionalmente inviável estabelecer um ponto de distribuição em todas as cidades brasileiras, notamos que o ICMS sobre frete, com alíquotas que chegam a 19% sobre o valor do serviço, é um fator relevante a ser considerado no posicionamento logístico, além da distância total percorrida.

[1] Cada nível corresponde a um nó na hierarquia de distribuição, ou seja, locais por onde a mercadoria passa até chegar ao consumidor final.

[2] Acrônimo que se refere aos países membros que formam um grupo político de cooperação.

Objetivos

Com relação à comercialização de combustíveis no Brasil, é evidente a influência que a localização de um terminal de distribuição exerce sobre o preço final da mercadoria. Os principais objetivos deste trabalho são o estudo e a análise de desempenho de algoritmos que possam determinar uma configuração de locais que ofereça a redução do custo logístico da distribuição. Para validar e avaliar o algoritmo implementado é aplicada uma massa de dados que representa uma empresa de distribuição de combustíveis presente na região sudeste brasileira, considerando 1668 municípios, cerca de 50 bases de distribuição e 1668 clientes.

Organização

Este trabalho está organizado da seguinte forma. O Capítulo 1 aborda a logística de distribuição de combustível no Brasil. No Capítulo 2 são apresentadas formulações clássicas sobre o problema da localização de instalações. Em seguida, no Capítulo 3 são introduzidos os principais impostos sobre o transporte rodoviário de cargas e a possibilidade de otimização da contribuição em função da escolha da origem e do destino das mercadorias. No Capítulo 4 é proposto um modelo matemático capaz de otimizar o custo de frete e tentativas de otimizar o tempo de execução do algoritmo. No Capítulo 5 são apresentados os resultados obtidos com as versões sequencial e paralela do algoritmo implementado. No capítulo 6, são apresentados alguns trabalhos de outros autores relacionados ao SUFLP e suas principais características. Em seguida, são apresentadas as conclusões e sugestões de trabalhos futuros.

1 A LOGÍSTICA DE DISTRIBUIÇÃO DE COMBUSTÍVEIS NO BRASIL

Até a década de 1990, o governo exercia forte controle sobre o setor de petróleo e gás no Brasil. A regulamentação vigente tornava quase exclusivo o controle da Petrobras sobre a exploração e o refino de petróleo em solo nacional, inibindo novos investimentos e impondo grandes barreiras à entrada de empresas privadas nesse mercado. Em 1997, com a criação da lei nº. 9.478, foi estabelecida uma política de redução do controle estatal sobre as atividades de exploração e produção de combustíveis derivados de petróleo. Até o advento desta lei, outras empresas só podiam atuar na venda de produtos derivados (BRASIL, 1997).

A *Lei do Petróleo*, como ficou conhecida, permitiu a entrada de empresas privadas e estatais nas atividades até então exercidas apenas pela Petrobras, além de estabelecer a criação da Agência Nacional do Petróleo (ANP) que mais tarde seria a autarquia responsável pela regulação e fiscalização do setor. A Petrobras perdia, então, o monopólio da exploração e refino de petróleo no Brasil.

A flexibilização do mercado provocou o aumento na concorrência ao permitir a entrada de empresas estrangeiras com tradição em atividades de exploração e refino. Contudo, o crescimento mais expressivo se deu na atividade de distribuição. Dados do Sindicato Nacional das Empresas Distribuidoras de Combustíveis e de Lubrificantes (SINDICON) apontam a existência de 16 refinarias no Brasil, sendo 13 delas controladas pela Petrobras (SINDICOM, 2014). Já no ramo de distribuição de combustíveis líquidos atuam 206 distribuidoras (ANP, 2014). Apesar da livre concorrência, a estatal Petrobras Distribuidora S.A. detém uma grande participação no mercado, obtendo um *market-share* de 46% em 2013.

A gama de produtos ofertados pelas distribuidoras é grande e varia por região. Os principais derivados líquidos de petróleo comercializados incluem a gasolina automotiva, o óleo diesel e o querosene de aviação. Já entre os biocombustíveis, estão o álcool anidro, o álcool hidratado e o biodiesel. Asfaltos, lubrificantes e gás liquefeito de petróleo (GLP) são exemplos de produtos com processo industrial e logístico adicional, pois envolvem em sua distribuição processos de mistura, embalagem e compressão, respectivamente.

Entre 2005 e 2012, o Brasil experimentou um crescimento de 47% no volume de vendas dos principais derivados líquidos de petróleo, com destaque para as vendas de gasolina automotiva (gasolina C) e a gasolina de aviação. A Tabela 1 mostra a evolução do volume de vendas em mil m^3.

A abertura total do mercado petrolífero foi concluída em janeiro de 2002, quando a lei do petróleo se estendeu ao setor de distribuição. A abertura do mercado permitiu que as empresas distribuidoras de combustíveis importassem derivados de petróleo. Desde então, a eficiência operacional e metas de redução de custo têm sido essenciais para a

Tabela 1 - Vendas nacionais dos principais derivados de petróleo.

Derivado	2005	2006	2007	2008	2009	2010	2011	2012
Total	84.140	84.486	88.419	92.682	92.332	102.878	111.335	11.9838
Gasolina C	23.553	24.008	24.325	25.175	25.409	29.844	35.491	39.698
Gasolina de aviação	55	52	55	61	62	70	70	76
GLP	11.639	11.783	12.034	12.259	12.113	12.558	12.868	12.926
Óleo combustível	5.237	5.127	5.525	5.172	5.004	4.901	3.672	3.934
Óleo diesel	39.167	39.008	41.558	44.764	44.298	49.239	52.264	55.900
QAV	4.429	4.466	4.891	5.227	5.428	6.250	6.955	7.292

Legenda: Vendas nacionais pelas distribuidoras em mil m^3.
Fonte: Agência Nacional do Petróleo

obtenção da vantagem competitiva.

1.1 **Esquema de Refino e Distribuição**

A indústria de petróleo engloba um conjunto de atividades que se inicia na prospecção de jazidas, passa pelo desenvolvimento de novos produtos e pelo refino, onde o petróleo extraído é transformando em uma imensa gama de produtos derivados. Após as etapas relacionadas ao processamento, entram as atividades de comercialização, transporte e distribuição (ANP, 2011).

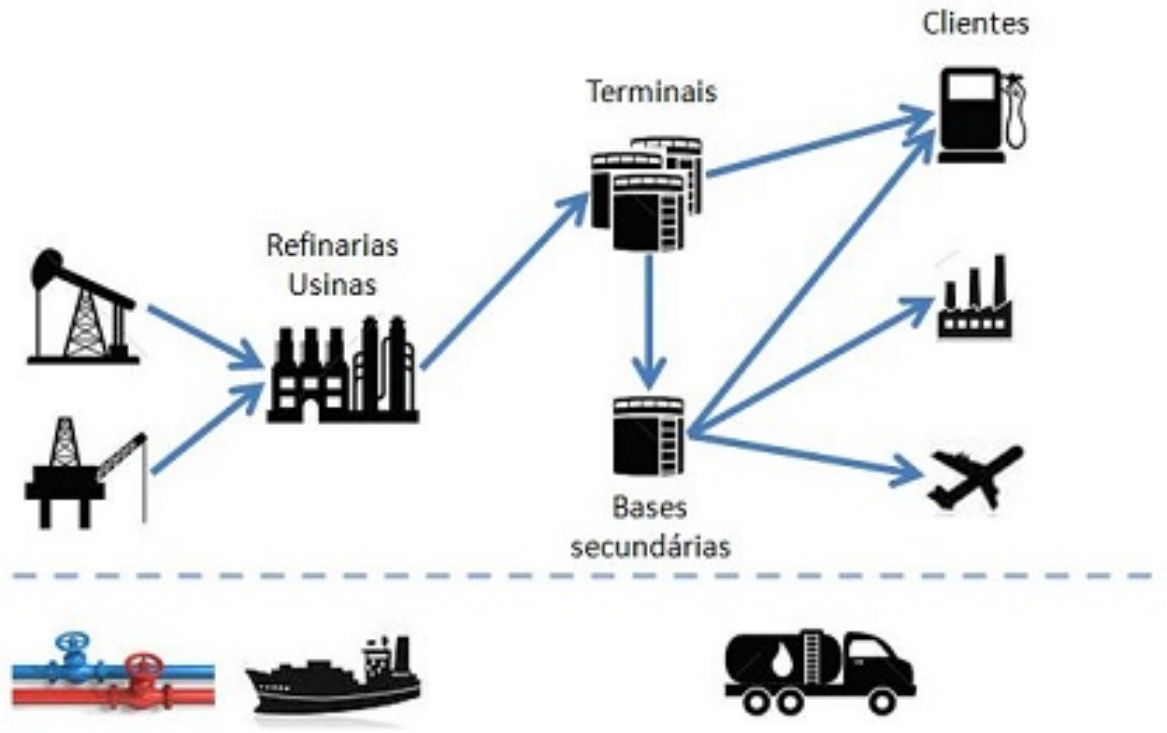
A estrutura logística desse setor reflete este conjunto de atividades, sendo dividida em dois grupos: o setor *upstream* e o setor *downstream*. O setor *upstream* engloba as atividades de exploração e produção de petróleo. Já o setor *downstream* concentra as atividades de aquisição, armazenagem, comercialização e distribuição de derivados.

A Figura 1 ilustra a configuração da rede logística desta indústria aplicada à distribuição de combustíveis.

O petróleo extraído em terra ou mar é levado às refinarias, que realizam processos químicos de limpeza e destilação, produzindo diversos derivados acabados. Tais produtos são transportados através de dutos ou navios para os terminais das empresas distribuidoras, que por sua vez, transferem o estoque para diversas bases secundárias. As bases secundárias entregam o produto acabado ao consumidor final quase totalmente pelo modal rodoviário, tendo menor importância os modais ferroviário e fluvial.

No que se refere a localização das refinarias, das 13 em plena atividade, 7 se concentram na região Sudeste, 3 na região Sul, 2 no Nordeste e uma na região Norte. Existe uma forte concentração da capacidade de refino na região Sudeste, chegando a aproximadamente 63% da capacidade total (ANP, 2002). Em relação à distribuição, existem cerca de 300 bases secundárias entregando combustíveis líquidos diretamente ao consumidor final ou aos postos de serviço automotivo em todas as regiões do país (IBP,

Figura 1 - Configuração da rede de distribuição de combustíveis.



Legenda: Esquema logístico de refino e distribuição de combustíveis.
Fonte: O autor, 2014

2014). Embora a rodovia represente o maior custo de transporte em comparação com o duto e a ferrovia, a distribuição ao consumidor final é feita quase exclusivamente por este modal. Os motivos incluem não apenas a deficiente malha ferroviária e dutoviária brasileira, mas também a inviabilidade de se aplicar tais meios de transporte a volumes pequenos ou fracionados.

2 TEORIA SOBRE LOCALIZAÇÃO DE INSTALAÇÕES

O problema de localização pode ser definido como um problema de alocação espacial de recursos. A hipótese da teoria da localização é a de que cada empresa escolhe a localização mais conveniente que leve à maximização dos lucros da sua atividade.

O problema de localização de instalações (PLI) envolve o estudo de uma área específica a partir das unidades de distribuição de produtos ou de prestação de serviços de modo a maximizar lucros. As decisões de localização, no contexto do planejamento logístico, correspondem à determinação do número, localização e tamanho das instalações a serem utilizadas. O principal objetivo é determinar a quantidade e a localização ideal destas unidades de forma a atender, da melhor maneira possível, um conjunto de usuários cuja localização é conhecida.

De forma geral, os modelos matemáticos para solução do problema de localização de instalações baseiam-se na minimização dos custos envolvidos. Portanto, a solução obtida satisfaz o objetivo de maximização de lucros. Essa é uma simplificação que deve ser realizada de modo cuidadoso. Isso porque, em casos onde a demanda possa apresentar uma variação importante com a localização escolhida, ela pode não representar o problema corretamente. O problema de localização de instalações é considerado bastante complexo, mesmo com essa simplificação.

Em seu estudo, Aikens (AIKENS, 1985) classifica as variações do problema de acordo com os critérios de capacidade, quantidade de níveis da cadeia de suprimentos, quantidade de produtos, ou ainda considera a estrutura de custos, o determinismo, o grau de incerteza e as restrições de produtos. Cada uma dessas variações estão descritas a seguir.

O critério de capacidade gera dois modelos de localização denominados capacitados ou não-capacitados. No modelo capacitado deve-se considerar a restrição de estoque disponível para suprimento das demandas. No modelo não-capacitado o estoque pode ser considerado infinito.

Utilizando o critério de quantidade de níveis da cadeia de suprimento, é necessário que o problema considere se a comercialização de um produto obedece a um fluxo hierarquizado, ou ainda, se possui zero, um ou múltiplos níveis.

A quantidade de produtos também pode gerar dois modelos distintos para o problema de localização de instalações, que são o modelo para um único produto (*single product*) ou o modelo que considera múltiplos produtos (*multi-products* ou *multicommodities*).

Em relação à localização, há a possibilidade de considerar o problema como um sistema linear, onde as bases podem ser localizadas em qualquer ponto do mapa, ou de maneira não linear, se a localização for possível apenas em pontos pré-definidos. De

maneira análoga a um grafo, o PLI linear pode situar bases em arcos da rede, já o PLI não linear só pode posicioná-las nos vértices.

Os modelos podem ainda ser estáticos ou dinâmicos. A maioria dos modelos estudados não consideram variação de demanda, capacidade de estoque ou outras restrições em função do tempo, sendo considerados modelos estáticos. Se características como sazonalidade da demanda, disponibilidade de estoque ou restrições de rota variam em função do tempo, o modelo é considerado dinâmico.

Modelos probabilísticos (estocásticos) e determinísticos são outras duas variações. Assim como os modelos podem ser estáticos ou dinâmicos, eles também podem ser probabilísticos, caso seja necessário considerar incertezas, como por exemplo, riscos de queda na demanda ou acidentes. Já os modelos determinísticos não consideram variáveis deste tipo.

Sobre a oferta de produtos, pode ser opção da indústria escolher trabalhar com um subconjunto de produtos para reduzir custos em determinados trechos.

A complexidade do modelo matemático necessário para a solução do problema de localização de instalações aumenta a medida em que se torna necessário considerar as variações citadas anteriormente. Nas próximas seções são apresentadas algumas formulações clássicas do PLI. As concepções do problema descritas foram selecionadas entre os diferentes trabalhos publicados com aplicações na otimização logística. Os modelos são apresentados em uma progressão do mais simples ao mais complexo, em termos do tempo computacional estimado para a solução.

2.1 **Localização de instalações simples sem restrição de capacidade**

Considerado o mais simples dos casos, o *Simple Uncapacited Facility Location Problem* (SUFLP) (KUEHN; HAMBURGER, 1963) é aplicado quando se deseja planejar a distribuição de apenas um produto, considerando seu estoque infinito. Sua formulação matemática é a seguinte:

$$\sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{i \in I} f_i z_i$$

cujo objetivo é minimizar esse somatório, sujeito as seguintes restrições:

$$\sum_{i \in I} x_{ij} = 1, \qquad j \in J \tag{1}$$

$$z_i - x_{ij} \geq 0, \qquad i \in I, \qquad j \in J \tag{2}$$

$$x_{ij} \geq 0 \tag{3}$$

$$z_i \in \{0, \quad 1\} \tag{4}$$

onde:

$x_{ij} =$ fração da demanda do cliente j fornecida pela instalação i;

$z_i =$ 1 se a instalação estiver aberta, 0 caso contrário;

$c_{ij} =$ custo de produção e distribuição para fornecer aos clientes a partir da instalação i;

$f_i =$ custo fixo de estabelecer a instalação i;

I, J são os conjuntos de possíveis locais para as instalações e a localização dos clientes.

A restrição (1) exige que toda a demanda de cada cliente seja completamente suprida por apenas um fornecedor, ou seja, x_{ij} deve assumir o valor 1 para um par ij e o valor 0 para todos os demais. Em (2) é garantido que os clientes receberão produtos apenas de instalações abertas. Este modelo é considerado sem níveis, uma vez que todas as instalações podem distribuir produtos diretamente aos clientes.

2.2 Localização de instalações simples sem restrição de capacidade com múltiplos níveis

Um modelo para localização simultânea de plantas industriais e armazéns foi proposto por Kaufman, Eede e Hansen (KAUFMAN; EEDE; HANSEN, 1977) baseado no SUFLP. Porém, esse modelo considera uma estrutura logística com dois níveis. Em seu estudo, o artigo propõe que os clientes possam receber produtos diretamente das plantas ou de armazéns. A formulação matemática do problema está descrita a seguir:

$$\sum_{i \in I} \sum_{j \in J} \sum_{k \in K} c_{ijk} x_{ijk} + \sum_{i \in I} f_i z_i + \sum_{j \in J} g_j y_j$$

cujo objetivo é minimizar esse somatório, sujeito as seguintes restrições:

$$\sum_{i \in I} \sum_{j \in J} x_{ijk} = 1, \qquad k \in K \tag{5}$$

$$\sum_{j \in J} x_{ijk} \leq z_i, \qquad i \in I, \quad k \in K \tag{6}$$

$$\sum_{i \in I} x_{ijk} \leq y_j, \qquad j \in J, \quad k \in K \tag{7}$$

$$z_i \leq y_i, \qquad i \in I \tag{8}$$

$$x_{ijk} \geq 0, \qquad i \in I, \qquad j \in J, \quad k \in K \tag{9}$$

$$y_j, \quad z_i \in \{0, 1\}, \qquad i \in I, \quad j \in J \tag{10}$$

onde:

$x_{ijk} =$ fração da demanda do cliente k fornecida pela planta i através do armazém j;

$z_i =$ 1 se a planta estiver aberta, 0 caso contrário;

$y_j =$ 1 se o armazém estiver aberto, 0 caso contrário;

$c_{ijk} =$ custo de produção e distribuição para suprir a demanda dos clientes k a partir da planta i e do armazém j;

$f_i =$ custo fixo de estabelecer a planta i;

$g_j =$ custo fixo de estabelecer o armazém j;

I, J, K representam os conjuntos de possíveis plantas, armazéns e clientes, respectivamente.

A inclusão de um nível na hierarquia de distribuição torna necessário dobrar o número de variáveis binárias (conjuntos I e J). Algumas restrições são equivalentes ao SUFLP como a garantia de satisfação da demanda em (5) e a necessidade do suprimento ser realizado apenas por plantas e armazéns abertos em (6) e (7), respectivamente. A condição imposta em $z_i \leq y_i$ significa que há um ou mais armazéns abertos para cada planta estabelecida, contudo, esta restrição pode ser retirada caso o ambiente estudado permita que clientes recebam diretamente das plantas [3].

[3] Os derivados de petróleo são comercializados desta maneira, entregues diretamente pelos terminais ou pelas bases secundárias.

2.3 Localização de instalações sem restrição de capacidade com vários produtos

Este modelo foi inicialmente proposto por Warszawski (WARSZAWSKI, 1973) e considera que cada instalação pode ofertar diferentes mercadorias, conforme sua localização. Este estudo contribuiu para complementar o desenvolvimento do modelo SUFLP e permitir aplicações práticas, tendo em vista que cada setor da indústria comercializa uma ampla gama de mercadorias, e que a otimização operacional deve considerar todos os produtos ofertados. A formulação proposta pelo autor está mostrada a seguir:

$$\sum_{i\in I}\sum_{j\in J}\sum_{p\in P} c_{ijp}x_{ijp} + \sum_{i\in I}\sum_{p\in P} f_{ip}z_{ip}$$

cujo objetivo é minimizar esse somatório, sujeito as seguintes restrições:

$$\sum_{i\in I} x_{ijp} = 1 \qquad se \quad D_jp \geq 0 \quad e \quad j \in J, \quad p \in P \tag{11}$$

$$\sum_{p\in P} z_{ip} \leq 1 \qquad i \in I \tag{12}$$

$$\sum_{j\in J} x_{ijp} \leq \sum_{j\in J} z_{ip} \qquad i \in I, \quad p \in P \tag{13}$$

$$x_{ijp} \geq 0 \qquad i \in I, \quad j \in J, \quad p \in P \tag{14}$$

$$z_{ip} \in \{0,1\} \qquad i \in I, \quad p \in P \tag{15}$$

onde:

$x_{ijp} =$ fração da demanda do cliente j pelo produto p fornecida pela planta i;

$z_{ip} =$ 1, se a planta estiver aberta e operando com o produto p. 0, caso contrário;

$c_{ijp} =$ custo de produção e distribuição para suprir a demanda dos clientes j pelo produto p a partir da planta i;

$f_{ip} =$ custo fixo de estabelecer a planta i para produzir p;

$D_{jp} =$ demanda total do cliente j pelo produto p;

I, J, P representam os conjuntos de possíveis plantas, clientes e produtos, respectivamente.

De maneira análoga ao SUFLP, a restrição imposta em (11) garante que toda a demanda dos clientes seja atendida, e (13) assegura a operação apenas entre plantas abertas. É importante mencionar que a imposição feita em (12) foi originalmente estabelecida

pelo autor para que cada planta pudesse comercializar apenas um produto, podendo ser desconsiderada para permitir a oferta simultânea de diferentes mercadorias.

2.4 Localização de Instalações com Restrição de Capacidade

A localização de instalações com restrição de capacidade é uma variação muito importante do SUFLP devido a sua aplicação prática. Este modelo considera que cada instalação possui uma capacidade máxima de atendimento às demandas dos seus clientes, ou seja, o estoque do produto comercializado é finito. Sua formulação proposta está mostrada a seguir:

$$Z = min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} + \sum_{i=1}^{n} f_i y_i$$

cujo objetivo é minimizar esse somatório, sujeito as seguintes restrições:

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, \cdots, m; \tag{16}$$

$$\sum_{i=1}^{m} d_i x_{ij} \leq s_j y_j, \qquad j = 1, \cdots, n; \tag{17}$$

$$0 \leq x_{ij} \leq y_j \leq 1, \qquad i = 1, \cdots, m; \quad j = 1, \cdots, n; \tag{18}$$

$$y_j = \{0, \quad 1\}, \qquad j = 1, \cdots, n; \tag{19}$$

$$\sum_{j=1}^{n} s_j y_j \geq \sum_{i=1}^{m} d_i, \tag{20}$$

onde:

$c_{ij} =$ custo total do transporte da instalação j para servir o cliente i;

$d_i =$ demanda do cliente i;

$s_j =$ capacidade da instalação j;

$f_j =$ custo fixo associado a instalação j;

$x_{ij} =$ fração da demanda do cliente i fornecido pela instalação j;

$y_j =$ 1 se a instalação j estiver aberta, 0 caso contrário;

I, J são os conjuntos de possíveis clientes e instalações, respectivamente.

Os modelos apresentados nesse capítulo são apenas exemplos de pesquisas significativas relacionadas a localização de instalações. Aplicações da técnica de otimização *branch and bound* sobre o problema de localização de instalações são comuns desde a década de 1960. Nessa época, já se pode notar a preocupação com o tempo computacional envolvido na solução do problema (EFROYMSON; RAY, 1966). A partir da década de 1970, pode-se observar alguns estudos tratando da localização de fontes de suprimento para diversos produtos *multicommodity* (WARSZAWSKI, 1973). Porém, a maioria dos trabalhos publicados a partir desta época trata de modelos com restrição de capacidade, cuja aplicação prática é latente. Um menor número de autores aborda modelos estocásticos ou modelos com múltiplos níveis, como por exemplo, a aplicação do método *branch and bound* em um modelo de distribuição multinível (TCHA; LEE, 1984).

3 CARGA TRIBUTÁRIA APLICADA AO SETOR DE TRANSPORTES

O sistema tributário brasileiro é conhecido por ser complexo e por ter custos elevados para o contribuinte. Uma comparação realizada entre 177 países revela que o sistema brasileiro é o mais complexo em termos do tempo gasto pelo contribuinte no cumprimento de suas obrigações fiscais, segundo o estudo publicado pela empresa de consultoria *Price Waterhouse & Coopers* e pelo *World Bank* (PRICE WATERHOUSE & COOPERS AND THE WORLD BANK, 2008). Ainda de acordo com o estudo, metade desse tempo é despendido apenas no Imposto sobre Circulação de Mercadorias e Serviços (ICMS). Esta complexidade faz com que o custo para manter as obrigações fiscais quitadas corretamente seja, às vezes, maior do que o valor dos tributos em si.

As obrigações fiscais estão permeadas em todas as esferas da administração brasileira. Há diversos impostos federais como o PIS, COFINS e IR[4], além de contribuições como o INSS[5]. Tais tributos podem ser aplicados em serviços de diferentes naturezas. Contudo, sua incidência está ligada a opção de tributação escolhida durante a constituição da empresa prestadora do serviço.

Como o objetivo deste trabalho é descobrir vantagens fiscais relacionadas à localização geográfica de uma empresa contratante de serviços de transporte, nas próximas seções são analisados o ICMS – tributo estadual incidente sobre transportes intermunicipais e interestaduais – e o ISS – tributo municipal aplicado apenas aos transportes realizados dentro dos limites das cidades.

3.1 Imposto sobre Circulação de Mercadorias e Serviços

O imposto sobre operações relativas à circulação de mercadorias e sobre prestação de serviços de transporte interestadual e intermunicipal (ICMS) é um tributo de competência dos estados e do distrito federal. Sua regulamentação constitucional está prevista na Lei Complementar 87/1996 (BRASIL, 1996). O ICMS é a principal fonte de receita dos estados. Conforme expressa a Constituição Federal no seu artigo 155, "cabe à lei complementar regular a forma como, mediante deliberação dos estados e do distrito federal, isenções, incentivos e benefícios fiscais serão concedidos e revogados". Na prática isto significa que benefícios e isenções do imposto concedidas por um estado podem ser

[4] *Programas de Integração Social*, *Contribuição para Financiamento da Seguridade Social* e *Imposto de Renda*, respectivamente.

[5] Instituto Nacional de Seguridade Social

utilizados como atrativos para que grandes empresas se instalem em seu território. Isso porque cada estado pode reduzir as suas alíquotas para oferecer tais vantagens.

No serviço de transporte interestadual, o ICMS é devido ao estado onde a mercadoria for embarcada. A alíquota é definida dependendo do estado de destino. Os transportes intermunicipais realizados no território de um único estado estão sujeitos à alíquota interna. A Tabela 2 contém os percentuais aplicados ao transporte de cargas em todos os estados e no distrito federal. As linhas representam os estados de origem, enquanto as colunas representam os destinos. Observa-se em negrito na diagonal principal, as alíquotas internas praticadas por cada estado.

Tabela 2 - Alíquotas de ICMS sobre Operações Interestaduais.

%	AC	AL	AM	AP	BA	CE	DF	ES	GO	MA	MT	MS	MG	PA	PB	PR	PE	PI	RN	RS	RJ	RO	RR	SC	SP	SE	TO
AC	**17**	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
AL	12	**17**	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
AM	12	12	**17**	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
AP	12	12	12	**17**	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
BA	12	12	12	12	**17**	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
CE	12	12	12	12	12	**17**	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
DF	12	12	12	12	12	12	**17**	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
ES	12	12	12	12	12	12	12	**17**	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
GO	12	12	12	12	12	12	12	12	**17**	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
MA	12	12	12	12	12	12	12	12	12	**17**	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
MT	12	12	12	12	12	12	12	12	12	12	**17**	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
MS	12	12	12	12	12	12	12	12	12	12	12	**17**	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
MG	7	7	7	7	7	7	7	7	7	7	7	7	**18**	7	7	12	7	7	7	12	12	7	7	12	12	7	7
PA	12	12	12	12	12	12	12	12	12	12	12	12	12	**17**	12	12	12	12	12	12	12	12	12	12	12	12	12
PB	12	12	12	12	12	12	12	12	12	12	12	12	12	12	**17**	12	12	12	12	12	12	12	12	12	12	12	12
PR	7	7	7	7	7	7	7	7	7	7	7	7	12	7	7	**18**	7	7	7	12	12	7	7	12	12	7	7
PE	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	**17**	12	12	12	12	12	12	12	12	12	12
PI	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	**17**	12	12	12	12	12	12	12	12	12
RN	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	**17**	12	12	12	12	12	12	12	12
RS	7	7	7	7	7	7	7	7	7	7	7	7	12	7	7	12	7	7	7	**17**	12	7	7	12	12	7	7
RJ	7	7	7	7	7	7	7	7	7	7	7	7	12	7	7	12	7	7	7	12	**19**	7	7	12	12	7	7
RO	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	**17**	12	12	12	12	12
RR	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	**17**	12	12	12	12
SC	7	7	7	7	7	7	7	7	7	7	7	7	12	7	7	12	7	7	7	12	12	7	7	**17**	12	7	7
SP	7	7	7	7	7	7	7	7	7	7	7	7	12	7	7	12	7	7	7	12	12	7	7	12	**18**	7	7
SE	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	**17**	12
TO	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	**17**

Legenda: Alíquotas interestaduais de ICMS.
Fonte: Decretos Estaduais e do Distrito Federal

Podemos notar que as alíquotas internas de ICMS variam entre 17% e 19%. Na grande maioria dos estados a alíquota interna é de 17%. O único estado com uma alíquota de 19% é o estado do Rio de Janeiro. Com uma alíquota interna de 18% temos os estados de Minas Gerais, Paraná e São Paulo.

Quanto as alíquotas de ICMS interestaduais, elas variam entre 7% e 12%.

3.2 Imposto sobre Serviços de Qualquer Natureza

O Imposto Sobre Serviços de Qualquer Natureza (ISS), é um tributo de competência dos municípios e do distrito federal. Sua regulamentação é prevista em lei, e o imposto é devido no local do estabelecimento prestador ou, na falta do estabelecimento, no local do domicílio do prestador (BRASIL, 2003).

O percentual do tributo varia entre a alíquota mínima de 2% e a alíquota máxima de 5%. Ainda nessa lei complementar, são listados os serviços sujeitos à tributação, incluindo o transporte de cargas realizado dentro do município, e também as exceções.

Aplicado ao escopo dessa dissertação, o ISS só incide nos casos onde a base distribuidora entrega produtos a clientes situados no mesmo município.

Para fins ilustrativos, a Tabela 3 mostra as alíquotas de ISS vigentes nos municípios com presença de bases distribuidoras de derivados (ANP, 2014).

Pode-se observar que a alíquota máxima é adotada na maioria das cidades. Portanto, para nos experimentos, o algoritmo de otimização de custos apresentado neste trabalho considera a alíquota de 5% para todas as demais cidades que não estão presentes nessa tabela.

Tabela 3 - Alíquotas de ISS sobre operações dentro dos municípios.

Cidade	UF	%	Cidade	UF	%
ANAPOLIS	GO	5	JEQUIE	BA	5
ARACATUBA	SP	4	LEME	SP	5
ARAUCARIA	PR	2	LONDRINA	PR	5
ARUJA	SP	5	LUIS ANTÔNIO	SP	5
ASSIS CHATEAUBRIAND	PR	5	MANAUS	AM	5
BARRA MANSA	RJ	5	MANDAGUACU	PR	2
BARUERI	SP	5	MARINGA	PR	3
BAURU	SP	2	MAUA	SP	3
BELO HORIZONTE	MG	5	MONTE CARMELO	MG	5
BRASILIA	DF	5	MOSSORO	RN	5
BRUSQUE	SC	5	NATAL	RN	5
CAMPINA GRANDE DO SUL	PR	5	NOSSA SENHORA DO SOCORRO	SE	5
CAMPINAS	SP	3,5	NOVA AMÉRICA DA COLINA	PR	5
CAMPO GRANDE	MS	5	NOVA ESPERANÇA	PR	4
CAMPO LARGO	PR	5	NOVA SANTA RITA	RS	5
CANDEIAS	BA	5	PASSOS	MG	5
CANOAS	RS	2	PAULINIA	SP	5
CASCAVEL	PR	5	PETROLINA	PE	5
CAXIAS DO SUL	RS	5	PINHAIS	PR	5
CHÃ DE ALEGRIA	PE	5	PORTO ALEGRE	RS	5
CHAPECO	SC	5	PORTO VELHO	RO	5
CIANORTE	PR	5	QUARTO CENTENARIO	PR	5
COTIA	SP	5	RECIFE	PE	5
CUIABA	MT	5	RIBEIRÃO PRETO	SP	5
CURITIBA	PR	5	RIO CLARO	SP	4
DOURADOS	MS	5	RIO DE JANEIRO	RJ	5
DUQUE DE CAXIAS	RJ	5	ROLÂNDIA	PR	4
EMBU	SP	5	SALVADOR	BA	5
ESTEIO	RS	2	SANTA MARIA	RS	4
FEIRA DE SANTANA	BA	5	SANTO ANDRE	SP	3
FORTALEZA	CE	5	SÃO MATEUS DO SUL	PR	4
GUARAMIRIM	SC	5	SAO PAULO	SP	5
GUARULHOS	SP	2	SENADOR CANEDO	GO	4
GURUPI	TO	3,5	SERRA	ES	5
IPOJUCA	PE	5	SINOP	MT	4
ITAJAI	SC	3	SOROCABA	SP	5
ITU	SP	5	TUBARAO	SC	5
JACAREI	SP	3	UBERABA	MG	3
JARAGUA DO SUL	SC	5	UBERLANDIA	MG	3
JARDINÓPOLIS	SP	5	VARZEA GRANDE	MT	5

Legenda: Alíquotas de ISS em cidades com bases distribuidoras.
Fonte: Decretos Municipais e do Distrito Federal

4 MODELO PROPOSTO PARA OTIMIZAÇÃO DE CUSTOS DE TRANSPORTE

Para que seja possível aplicar o PLI de modo a otimizar os custos de transporte de derivados de petróleo, é necessário desenvolver um modelo matemático específico. Embora não seja muito diferente das equações apresentadas no Capítulo 2, é necessário acrescentar o custo do ICMS e do ISS ao custo do transporte entre fornecedor e cliente.

É apropriado ressaltar que este trabalho considera apenas os custos fixos de instalação, o custo com o transporte e a carga tributária aplicada ao frete, sem considerar os tributos aplicados à comercialização do produto ao cliente final.

Apesar da cadeia de refino e distribuição de derivados de petróleo se assemelhar claramente a um problema de distribuição com dois ou mais níveis – incluindo refinarias, terminais e bases secundárias – é factível considerar um modelo sem níveis hierárquicos (*single-echelon*) para as empresas distribuidoras de combustíveis, uma vez que apenas armazenam e revendem os produtos. A infraestrura requerida para levar os produtos até os terminais é composta em maior parte por dutos e ferrovias, modais com custo inferior, maior capacidade de transporte e consequentemente menos propensos a mudanças. Apesar de ser considerada deficitária, a infraestrutura de transportes brasileira não é unicamente responsável pela alta demanda rodoviária. É importante mencionar que a entrega ao consumidor final é fracionada em volumes consideravelmente menores, fator que em muitos casos não justificaria o grande investimento em dutos e ferrovias a postos de combustíveis ou pequenas empresas.

Como produto ofertado, o modelo proposto neste trabalho define como *Combustíveis Claros* o agrupamento da Gasolina, Óleo Diesel e Etanol, que em 2013 representaram 33% do consumo final de energia (MME, 2014). Estes produtos, que possuem características de densidade e temperatura semelhantes, são usualmente comercializados em volume e compartilham a mesma estrutura de armazenamento e transporte. Na prática, isso quer dizer que o mesmo caminhão tanque pode ser utilizado para transportar qualquer um destes produtos sem necessidade de tratamento químico entre as viagens. Assim como um tanque de armazenamento pode alternar entre os produtos sem necessidade de adequações.

Em relação à disponibilidade de estoque, o mercado brasileiro de distribuição não apresenta restrições significativas. Ao longo da história, é possível verificar um esforço contínuo de adequação das refinarias ao perfil de demanda do mercado brasileiro. Atualmente é preciso considerar que o Etanol, cuja demanda representa 4,8% da matriz energética nacional, apesar de compartilhar a mesma malha logística dos derivados de petróleo, é produzido em usinas de cana de açúcar. Tais instalações são consideravelmente mais simples e menos custosas se comparadas às refinarias de petróleo. Desta forma, um

modelo simplificado, porém adequado, é o *singlecommodity*, ou seja, aquele que considera a distribuição de um único produto (DUAILIB, 2012).

Considerando os fatos apresentados anteriormente, pode-se fazer uma adaptação do *Simple Uncapacited Facility Location Problem* (SUFLP) apresentado em 2.1:

$$\sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{i \in I} f_i z_i$$

sujeito as seguintes restrições:

$$\sum_{i \in I} x_{ij} = 1, \qquad j \in J \tag{21}$$

$$z_i - x_{ij} \geq 0, \qquad i \in I, \qquad j \in J \tag{22}$$

$$x_{ij} \geq 0 \tag{23}$$

$$z_i \in \{0, \quad 1\} \tag{24}$$

onde:

$x_{ij} =$ fração da demanda do cliente j fornecida pela instalação i;

$z_i =$ 1 se a instalação estiver aberta, 0 caso contrário;

$c_{ij} =$ custo de produção e distribuição para fornecer aos clientes a partir da instalação i;

$f_i =$ custo fixo de estabelecer a instalação i;

I, J são os conjuntos de possíveis locais para as instalações e a localização dos clientes.

Se assumirmos que a demanda de cada cliente é totalmente suprida por apenas um fornecedor, x_{ij} assume o valor 1, quando o cliente j está sendo suprido pelo fornecedor i e 0 quando não há ligação entre o par. Já o custo pode ser expresso em função da distância percorrida entre instalação i e cliente j e a incidência de imposto neste trecho, conforme a equação a seguir:

$$c_{ij} = t_d d_{ij} + k_{ij}(t_d d_{ij})$$

onde:

$t_d =$ tarifa aplicada a faixa de distância onde d está contido;

$d_{ij} =$ distância entre o cliente i e a instalação j;

$k_{ij} =$ alíquota de imposto incidente no trecho entre i e j.

A alíquota k se comporta de maneira não linear, e pode corresponder ao ISS ou ICMS. No modelo proposto, os possíveis locais para instalação das bases de distribuição e clientes é um subconjunto das cidades brasileiras. Torna-se, então, importante definir a seguinte restrição:

$$k_{ij} = \begin{cases} \%ISS & \text{se } i = j, \\ \%ICMS & \text{se } i \neq j. \end{cases}$$

Esta restrição garante a dualidade da carga tributária incidente. Ela pode assumir a alíquota municipal (ISS) quando o transporte se iniciar e terminar dentro de um mesmo município, ou intermunicipal (ICMS) quando a entrega for realizada em um município diferente. O modelo proposto pode ser expresso, então, da seguinte maneira:

$$\sum_{i \in I} \sum_{j \in J} ((t_d d_{ij} + k_{ij}(t_d d_{ij}))x_{ij}) + \sum_{i \in I} f_i z_i \tag{25}$$

sujeito as seguintes restrições:

$$x_{ij} \in \{0, \quad 1\} \tag{26}$$

$$z_i \in \{0, \quad 1\} \tag{27}$$

onde:

$t_d =$ tarifa aplicada a faixa de distância onde d está contido;

$d_{ij} =$ distância entre o cliente i e a instalação j;

$k_{ij} =$ alíquota de imposto incidente no trecho entre i e j;

$f_i =$ custo fixo de estabelecer a instalação i;

$z_i =$ 1 se a instalação estiver aberta, 0 caso contrário;

I, J os conjuntos de possíveis cidades para estabelecer as bases e clientes.

A existência do custo fixo de implantação cria uma descontinuidade quando não há fornecimento a partir de uma instalação, impedindo o uso de técnicas previstas em Programação Linear. Essas técnicas usualmente tratam problemas de otimização nos quais a função objetivo e as restrições são todas lineares.

O modelo proposto apresenta características semelhantes ao SUFLP. Na próxima seção, são apresentadas e analisadas as abordagens de solução deste problema, desenvolvidas por alguns autores da área de Pesquisa Operacional.

4.1 Abordagens para solução do SUFLP

Uma abordagem para resolver o SUFLP é avaliar todas as combinações possíveis para instalar um ou mais pontos de fornecimento nos locais disponíveis. Ao se conhecer o resultado do somatório para todos os casos, é possível determinar o de menor custo. Contudo, supondo que se deseja otimizar uma cadeia de suprimento presente nas 1668 cidades que compõem a região sudeste brasileira, tendo cada cidade uma demanda específica por combustível e sendo cada uma delas um local em potencial para a instalação de uma base fornecedora, o número de combinações torna esta abordagem inviável.

Considere, como exemplo, que uma empresa distribuidora deseja se instalar em 18 locais diferentes. A quantidade de possibilidades distintas é definida por meio da fórmula:

$$C_{18}^{1668} = \frac{1668!}{18!(1668-18)!}$$

Essa equação representa a combinação matemática entre um subconjunto de 18 elementos contidos no conjunto de 1668 cidades. É fácil notar que o tratamento de problemas envolvendo centenas ou milhares de locais é inviável por meio de abordagens de programação combinatória. Para tais casos, torna-se necessário a aplicação de heurísticas que sejam capazes de determinar a solução ótima sem que seja necessário resolver todas as possibilidades.

Muitos algoritmos encontrados na literatura são baseados na técnica *branch and bound* (CORNUEJOLS; SRIDHARAN; THIZY, 1991), cujo paradigma é resolver a equação do PLI desconsiderando as restrições de números inteiros, obtendo um resultado inicial C_0. Caso algum dos valores z_i seja fracionário, então estes valores são fixados em zero e a equação é resolvida novamente, obtendo-se então C_1. Depois, os mesmos elementos z_i são fixados com o valor 1 produzindo C_2. O limite mínimo (*lower bound*) é então estabalecido como o menor entre os dois resultados anteriores. Ou seja:

$$\overline{C} = min(C_1, C_2)$$

A solução opta pelo menor resultado e resolve mais uma vez a equação, produzindo novos valores fracionados para z_i e obtendo novos limites inferiores. O processo se repete até que seja encontrada uma solução onde todos os elementos de z sejam inteiros.

Estudos anteriores (EFROYMSON; RAY, 1966) e (KHUMAWALA, 1972) provam que esta técnica determina a solução ótima para o problema. Porém, o custo computacional para validar as possíveis soluções pode comprometer o desempenho do algoritmo se este for aplicado em problemas com centenas ou milhares de clientes.

Existem outras alternativas ao método *branch and bound*. Em seu artigo, Ja-

cobsen (JACOBSEN, 1983) propôs melhorias nas heurísticas *ADD* e *DROP*, propostas respectivamente por Kuehn (KUEHN; HAMBURGER, 1963) e Feldman, Lehrer e Ray (FELDMAN; LEHRER; RAY, 1966), na década de 1960. Contudo, estas técnicas são mais adequadas a modelos que consideram a restrição de capacidade dos pontos de fornecimento. As heurísticas mais sofisticadas são baseadas na ideia de se partir de uma solução factível previamente conhecida e aplicar modificações capazes de obter uma solução melhor (JACOBSEN, 1983). Tais abordagens se justificam porque a solução ótima pode não atender a requisitos alheios ao modelo matemático estabelecido, como questões políticas, infraestrutura disponível e mão de obra qualificada na região.

Baseado neste paradigma, podemos citar o *Alternate Location and Allocation* proposto por Cooper (COOPER, 1964), cuja idéia é dividir todos os possíveis destinos em subconjuntos aproximadamente iguais e determinar a melhor opção de fornecedor para cada um destes subconjuntos. A partir de então, outras opções de fornecedor são testadas e seus custos calculados a fim de se obter um melhor ponto de fornecimento para tal subconjunto. O procedimento é, então, repetido para cada subconjunto de clientes determinado inicialmente. De acordo com o autor, não é garantido que a solução ótima seja encontrada. Porém, sua aplicação é útil quando se deseja conhecer um conjunto de soluções melhores em relação à configuração inicial.

4.2 Algoritmo Proposto

Nesse trabalho é feita uma adaptação da técnica *Alternate Location and Allocation* (ALA) ao problema de otimização de custos de transporte com aplicação de impostos. O algoritmo adaptado visa minimizar o resultado da seguinte equação:

$$\sum_{i \in I} \sum_{j \in J} ((t_d d_{ij} + k_{ij}(t_d d_{ij}))x_{ij}) + \sum_{i \in I} f_i z_i$$

onde:

$t_d =$ tarifa aplicada a faixa de distância onde d está contido;

$d_{ij} =$ distância entre o cliente i e a instalação j;

$k_{ij} =$ alíquota de imposto incidente no trecho entre i e j;

$f_i =$ custo fixo de estabelecer a instalação i;

$z_i =$ 1 se a instalação estiver aberta, 0 caso contrário;

I, J os conjuntos de possíveis cidades para estabelecer as bases e clientes.

Para se avaliar o algoritmo foi construído um programa em linguagem *C*, denominado `SUFLP_simples.c`, com o objetivo de encontrar soluções que minimizam os custos de transporte e impostos.

O programa realiza a leitura de dois arquivos de entrada no formato `CSV`. O primeiro é uma matriz quadrada de dimensão $n \times n$, onde cada índice n é um código de 7 números inteiros, representando a cidade. De acordo com o padrão do IBGE, os dois primeiros números representam o estado e os cinco dígitos subsequentes representam o município. Cada elemento a_{ij} representa a distância entre as cidades i e j. A matriz lida é, então, carregada em uma estrutura do tipo vetor, que posteriormente é utilizada como referência para os dados de distância no algoritmo do SUFLP.

O segundo arquivo é uma tabela cujo conteúdo são informações sobre cada uma das n cidades mencionadas anteriormente. Os campos lidos são carregadas em outro vetor, onde cada elemento representa um tipo de dados composto (`struct`) denominado `cidades`. A formação desta estrutura é declarada em linguagem C conforme a Figura 2.

Figura 2 - Estrutura do arquivo de cidades

```
struct cidades{long int cidadeCliente;      //cidade cliente
               char UF[2];                  //Estado - UF
               char nomeCidade[100];        //nome da cidade
               int ativa;                   //'X' se esta cidade possui base
               int ISS;                     //aliquota de ISS
               long int custoFixo;          //custo fixo para implantar
               long int cidadeFornecedora; //cidade que faz o suprimento
               float melhorCusto;           //custo para distribuir
               float imposto;               //valor dos impostos
               };
```

Legenda: Estrutura em linguagem C do arquivo de cidades

O campo `ativa` determina se essa cidade possui uma base fornecedora instalada, e neste caso, o `custoFixo` é o custo fixo de se manter a instalação aberta nesse local. Outro campo importante é o `cidadeFornecedora`, que armazenar o código da cidade onde há uma instalação aberta fornecendo a este cliente. Já os campos `melhorCusto` e `imposto` armazenam o custo de distribuição de produtos entre a cidade fornecedora e a cidade cliente e o valor dos impostos incidentes nessa operação.

Outra fonte de informações utilizada neste programa é a matriz `matrizICMS` de dimensão 28×28, representando a alíquota de ICMS incidente entre os 27 estados brasileiros e o Distrito Federal. Optou-se por declarar esta matriz de forma constante no programa para evitar a leitura de outro arquivo, o que poderia reduzir o desempenho do programa.

Após a leitura dos arquivos de entrada, o programa passa a conhecer o subconjunto das cidades que possuem base fornecedora e, para cada cidade cliente, determina a melhor opção baseado no menor custo. Durante esta etapa, todos os custos parciais e os custos

fixos de implantação são acumulados na variável `custoEfetivoTotal`. Isto é feito de forma trivial ao assumirmos que o arquivo de cidades possui ao menos uma cidade marcada como ativa e que todas as cidades clientes passarão a ter uma cidade fornecedora associada. Com isto, é assegurado que haverá ao menos um subconjunto de cidades a serem supridas por ao menos uma cidade fornecedora.

Em seguida, a heurística ALA é utilizada de forma a avaliar a substituição da cidade fornecedora de cada subconjunto de clientes por cada uma das cidades existentes. O método consiste nas seguintes etapas:

(i) obter uma solução inicial, listando os locais com instalações abertas;

(ii) remover uma das instalações abertas da lista;

(iii) adicionar uma instalação, localizada em um ponto distinto;

(iv) repetir os passos (ii) e (iii) até que o resultado obtido não obtenha ganho.

Se a escolha de alguma cidade alternativa causar uma redução do custo total, esta cidade passa, então, a ser a fornecedora daquele subconjunto. Este procedimento é realizado em todos os demais subconjuntos de clientes até que todos tenham sido avaliados. Por fim, todas as configurações entre cliente e fornecedor avaliadas e seus respectivos resultados obtidos durante o processamento são gravados no arquivo `configuracaoFinalCidades.txt`. O código fonte completo do `SUFLP_simples.c` está contido no Apêndice A desta dissertação.

Para fins de ilustração, o pseudocódigo do programa está listado no Algoritmo 1.

Algoritmo 1 - Algoritmo SUFLP com técnica ALA.

Título
SUFLP com técnica Alternate Location and Allocation

Entradas
vetorCidades, vetorDistancia: lista de cidades e distâncias entre elas
matrizICMS: matriz de alíquotras de ICMS
i: contador auxiliar
k: número de cidades simuladas
custoEfetivo: custo efetivo total
custoFixo: custo fixo de implantação

Saídas
configuraçãoFinal: lista de cidades e respectivos fornecedores

Algoritmo Adaptação do ALA ao SUFLP
1. **declarar** r, p, i, j **numéricos**
2. **declarar** qtdBases, qtdBasesInativas **numéricos**
3. **declarar** custo, imposto, custoTotal, menorCusto **numéricos**
4. **ler** arquivo de distâncias
5. **ler** arquivo de cidades
6. **ler** Matriz de ICMS

7. $qtdBases \leftarrow cidadescombasesabertas$
8. $qtdBasesInativas \leftarrow cidadessembasesabertas$
9. $k \leftarrow quantidadetotaldecidades$

10. **enquanto** $(r \leq qtdBases)$, **fazer** {*Para cada cidade com base*}
11. **enquanto** $(p \leq qtdBasesInativas)$, **fazer** {*Para cada cidade sem base*}
12. **enquanto** $(i \leq k)$, **fazer** {*para todas as cidades*}
{*troca a cidade fornecedora r pela cidade p*}
{*acumula variável custoFixo lida no arquivo de cidades*}
{*recebe a distância d entre cliente e fornecedor do arquivo de distâncias*}
{*recebe a aliquota j de ICMS entre cliente e fornecedor*}
13. $imposto \leftarrow (d * j)/100$
14. $custo \leftarrow custo + (d + imposto)$
15. **fim enquanto**
16. $custoTotal \leftarrow custo + custoFixo$
{*se o custo desta rodada for menor que o melhor custo já obtido*}
17. **se** $(custoTotal \leq menorCusto)$, **então**
18. $menorCusto \leftarrow custoTotal$
19. **escrever** configuração final
20. **fim se**
21. **fim enquanto**
22. **fim enquanto**
Fim algoritmo

4.3 Adaptação do algoritmo ao MPI

Muitos problemas interessantes, em particular os problemas de otimização, não podem ser resolvidos dentro de um tempo razoável. Esse fato inviabiliza a utilização de aplicações reais.

Nos últimos anos tem-se observado um uso crescente de implementações paralelas nas mais diversas aplicações. Esse crescimento é motivado pelo surgimento de novas arquiteturas que integram dezenas de processadores rápidos e de baixo custo. Estas arquiteturas podem ser compostas, por exemplo, por estações de trabalho ou PCs conectados em rede, formando um ambiente de memória distribuída.

Com o objetivo de melhorar o desempenho do algoritmo proposto e descrito na seção anterior, optou-se por aplicar técnicas de computação paralela com utilização da biblioteca *Message Passing Interface* (MPI) (USFCA, 1998).

4.3.1 *Message Passing Interface*

O padrão MPI surgiu do esforço conjunto de pesquisadores, laboratórios governamentais e indústria. A ideia era discutir quais recursos básicos essenciais seriam necessários para uma interface de passagem de mensagens padrão. A primeira versão foi disponibilizada em junho de 1994.

O padrão MPI define a sintaxe e a semântica de rotinas, que estão agrupadas numa biblioteca, que é útil para uma vasta gama de aplicações escritas principalmente em liguagens C e Fortran. Em particular, as principais rotinas dão suporte a criação, execução e comunicação entre processos executados simultaneamente em diversos processadores.

As principais funções da biblioteca MPI incluem iniciar, terminar e identificar processos, obtenção de informações como número total de processos, comunicação ponto-a-ponto e comunicação coletiva. A comunicação pode ser síncrona ou assíncrona.

O MPI é um padrão que possui rotinas para comunicação de dados em computação paralela. É através de rotinas disponibilizadas que são realizadas as trocas de informações entre processos distintos. Tais processos se comunicam através de funções de envio e recepção de mensagens contendo dados inerentes a aplicação como um todo.

O protocolo do MPI foi desenvolvido de forma a garantir a portabilidade da aplicação entre ambientes diferentes. O paradigma é aplicável em várias configurações de ambientes, independente da velocidade da rede ou arquitetura de memória. Ou seja, não é necessário modificar o código fonte do programa desde que haja uma implementação do MPI em todos as plataformas utilizadas na execução da aplicação.

O MPI é apenas um padrão para programas com execução paralela baseado na troca de mensagens. Portanto, para utilizar este padrão é necessário que o ambiente

possua uma implementação instalada. Há diversas opções disponíveis, sendo o *Open MPI* (GABRIEL et al., 2004) e o *MPICH* (ARGONNE NATIONAL LABORATORY, 1998) as mais comuns. Para fins de avaliação do algoritmo abordado neste trabalho, foi escolhida a versão *MPICH 1.2.5*. Esta versão é bastante difundida e possui implementação em diferentes plataformas, incluindo *Windows*. Ainda, esta versão permite execução em múltiplas processadores, *clusters* ou mesmo a distribuição de processos entre diversos núcleos de um mesmo processador.

4.3.2 Algoritmo adaptado

Foram feitas modificações no algoritmo sequencial (`SUFLP_simples.c`) de modo a criar uma versão deste programa capaz de suportar paralelismo e obter um desempenho melhor. Foi necessário inserir comandos de inicialização do ambiente MPI e definir variáveis para controlar a distribuição de dados entre os processos. A Figura 3 mostra os respectivos comandos utilizados no início do programa para inicializar o ambiente MPI.

Figura 3 - Inicialização do ambiente MPI

```
int rank, numtasks;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
```

Legenda: Comandos para inicializar o MPI no programa

O primeiro comando define duas variáveis: `rank` e `numtasks`. A função `MPI_Init` ativa o início do ambiente paralelo (MPI) dentro daquele programa. A execução da função `MPI_Comm_size` retorna, na variável `numtasks`, o número de processos que são disparados simultaneamente. Enquanto que, a função `MPI_Comm_rank` retorna a identificação de cada um dos processos em execução, na variável `rank`.

Na versão paralela foi mantida a lógica *Alternate Location and Allocation* (ALA) para avaliação e substituição de cidades fornecedoras. Nesta versão, a cada substituição, a rotina para cálculo do custo efetivo total divide o vetor de cidades gerado, de maneira que cada processo calcule o custo parcial localmente. Após o resultado ser obtido em cada processo, é feita uma chamada ao comando `MPI_Allreduce`. Este comando é baseado no conceito de redução. É realizada uma operação que envolve a transformação de um conjunto de números em um único resultado, de acordo com a operação definida. Neste caso, a operação utilizada foi `MPI_SUM` com o propósito de somar os custos parciais, calculados localmente em cada processador, de modo a obter o custo efetivo total.

As modificações acrescentadas geraram o programa na versão paralella (`SUFLP_paralelo.c`). Este programa, também, é executado para fins de comparação de desempenho.

O programa parte do princípio onde todos os processos executam o mesmo código, sendo a massa de dados dividida entre os processos existentes. Isto é, segue o modelo de programação *Single Problem Multiple Data*. O código fonte completo do programa na versão paralela (`SUFLP_paralelo.c`) está contido no Apêndice B.

5 ANÁLISE E AVALIAÇÃO DOS RESULTADOS

5.1 Ambientes de Execução

Ambas versões do algoritmo utilizado foram editadas e compiladas utilizando o ambiente de desenvolvimento *Bloodshed Dev-C++* e o compilador de linguagem C denominado *GNU Compiler Collection* (GCC). Para que o compilador reconheça os comandos MPI inseridos no código fonte, foi necessário incluir as pastas `/MPICH/SDK.gcc/include` e `/MPICH/SDK.gcc/lib` na configuração de bibliotecas do *Dev-C++*.

Após a compilação, foram produzidos os arquivos executáveis de ambas as versões do programa. A versão sequencial não requer argumentos específicos e a chamada ao programa é trivial. Contudo, a versão paralela requer a utilização do programa `mpirun`, que é o componente responsável por iniciar programas com chamadas ao MPI. A sintaxe do comando é descrita a seguir:

```
mpirun -np #processos [opções] programa [argumentos ...]
```

onde `#processos` é o número de processos a serem utilizados na execução. Além disso, a chamada ao `mpirun` pode incluir uma ou mais opções. As mais comumente utilizadas estão descritas a seguir:

-localonly executa o programa apenas na máquina local;

-logon solicita o usuário e senha para *logon* nas máquinas utilizadas;

-env especifica variáveis de ambiente;

-hosts especifica os nomes de máquinas a serem utilizados.

As duas versões do algoritmo construídas e utilizadas nesse trabalho foram executadas no mesmo ambiente computacional. Todas as execuções foram realizadas em uma máquina Intel Core i5-2430M com 240GHz e 4Gb de memória RAM. O sistema operacional utilizado foi o *Windows 7 home SP1* de 64 bits. Este processador possui dois núcleos duplos, sendo capaz de executar até 4 processos simultâneos. Em relação ao MPI, foi utilizada a implementação `MPICH 1.2.5` para Windows.

5.2 Resultados

Para analisar os resultados e avaliar o desempenho do algoritmo proposto foram preparados cenários envolvendo 49, 1668 e 2856 cidades.

O cenário de 49 cidades possui as distâncias reais entre as cidades mencionadas de acordo com o IBGE (IBGE, 2013). Os demais cenários tentam reproduzir a dimensão de malha de distribuição logística real, permitindo que o tempo decorrido na solução do problema seja avaliado satisfatoriamente. Porém, as distâncias entre as cidades foram geradas aleatoriamente.

Cenário com 49 cidades

O cenário mais simples considera as 49 cidades da região sudeste com mais de 250 mil habitantes, de acordo com dados do IBGE (IBGE, 2013). Neste caso, estamos interessados em comprovar a eficácia do algoritmo em encontrar uma solução melhor a partir de uma configuração inicial conhecida. Foram criadas variações deste cenário com 4, 8, 12 e 16 fornecedores diferentes de modo a se avaliar a redução de custo obtida. Para cada variação, foram realizadas 20 execuções, tanto na versão sequencial quanto na versão paralela. O tempo médio de execução em segundos de cada cenário e a redução percentual do custo obtida com o algoritmo estão mostrados na Tabela 4.

Tabela 4 - Execução do SUFLP com 49 cidades.

Programa	Fornecedores #	Tempo (s)	Custo Inicial ($)	Melhor Custo ($)	Redução Custo (%)
Sequencial	4	0,016	15.961,00	12.778,07	19,94
	8	0,031	15.393,00	13.831,00	10,15
	12	0,048	16.282,00	14.720,00	9,59
	16	0,061	19.266,00	17.882,00	7,18
Paralelo	4	0,007	15.961,00	12.778,07	19,94
	8	0,016	15.393,00	13.831,00	10,15
	12	0,024	16.282,00	14.720,00	9,59
	16	0,031	19.266,00	17.882,00	7,18

Legenda: Resultados do SUFLP nas versões sequencial e paralela.

A coluna Tempo representa a média entre as 20 execuções realizadas com quatro processadores para cada variação da quantidade de fornecedores. Observa-se que os custos obtidos para cada cenário avaliado, tanto na versão sequencial quanto na versão paralela do algoritmo, são idênticos. Esse resultado é o esperado e valida a técnica de paralelismo aplicada ao programa. Embora, neste cenário, a quantidade de cidades seja pequena e o algoritmo tenha um tempo de execução baixo, os tempos obtidos mostram o potencial de paralelização do algoritmo. Isso porque para uma mesma quantidade de fornecedores o tempo do algoritmo paralelo é praticamente a metade do tempo de execução do algoritmo sequencial.

Neste cenário, deseja-se apenas comprovar que o algoritmo é capaz de encontrar

uma configuração de cidades fornecedoras que representem um menor custo de transporte, incluindo os impostos descritos no Capítulo 3. Note que, neste caso, sempre ocorreu redução no custo. A maior redução pode atingir aproximadamente 20% e a menor está em torno de 7%.

Os tempos de execução, bem como o ganho de desempenho obtido com o paralelismo, para cenários com maior quantidade de cidades, são avaliados e descritos a seguir.

Cenário com 1668 cidades

O segundo cenário representa uma cadeia de distribuição instalada em todos os municípios da região sudeste.

Os arquivos de entrada correspondem as 1668 cidades que compõem os estados do Espírito Santo, Minas Gerais, Rio de Janeiro e São Paulo. Neste caso, desejamos avaliar o tempo de execução do algoritmo para validar a redução obtida com a aplicação da técnica de paralelismo, utilizando o suporte a programação paralela fornecida pela biblioteca do MPI. Neste cenário variamos a quantidade de fornecedores para observar a influência deste parâmetro no tempo de execução da aplicação. Foram avaliadas variações do cenário com 8, 16, 24, 32, 40, 48, 56 e 64 fornecedores em ambas versões do programa, sequencial e paralela. Para cada quantidade de fornecedores foram realizadas 20 execuções com quatro processadores.

Os tempos de execução em segundos obtidos com a versão sequencial do programa estão listados na Tabela 5. As últimas linhas dessa tabela, t_{medio} e σ, representam, respectivamente, a média dos tempos obtidos em cada execução e o desvio padrão.

O desvio padrão para todas as quantidades de fornecedores é aproximadamente 1 ou inferior.

O tempos médios aumentaram a medida que incrementamos a quantidade de fornecedores. O incremento foi sempre de 8 fornecedores.

Tabela 5 - Execução do SUFLP sequencial com 1668 cidades.

#	**8**	**16**	**24**	**32**	**40**	**48**	**56**	**64**
t_1	268,612	499,733	798,461	1099,419	1329,47	1588,980	1856,220	2015,454
t_2	271,739	501,035	798,512	1100,258	1329,95	1589,489	1856,477	2015,912
t_3	272,365	500,482	798,869	1101,640	1329,65	1589,857	1857,186	2016,530
t_4	270,466	502,220	798,773	1101,402	1329,56	1589,665	1859,489	2017,234
t_5	272,483	504,118	799,403	1102,423	1329,98	1589,884	1860,943	2018,178
t_6	270,727	502,471	798,636	1101,453	1329,92	1589,124	1857,652	2016,076
t_7	270,757	500,783	799,381	1101,870	1329,66	1589,882	1860,530	2017,627
t_8	270,691	500,945	799,249	1099,639	1329,78	1589,001	1858,530	2016,538
t_9	271,261	500,762	798,588	1099,483	1329,79	1589,523	1857,633	2016,637
t_{10}	271,816	502,284	798,468	1100,917	1329,50	1589,626	1857,156	2017,033
t_{11}	269,518	502,137	798,991	1100,200	1329,84	1589,604	1860,349	2017,234
t_{12}	269,823	503,379	798,724	1100,975	1329,52	1589,303	1858,080	2017,129
t_{13}	269,929	500,228	799,288	1101,349	1329,94	1589,044	1859,778	2017,906
t_{14}	272,234	499,856	798,710	1102,052	1329,96	1589,010	1859,749	2017,760
t_{15}	271,040	501,799	798,696	1101,718	1329,73	1589,619	1859,096	2016,569
t_{16}	269,703	501,923	799,340	1100,103	1329,93	1589,003	1857,306	2017,294
t_{17}	268,644	500,377	799,099	1100,164	1329,89	1589,071	1856,995	2017,347
t_{18}	271,461	502,428	799,028	1101,386	1329,88	1589,853	1858,791	2016,634
t_{19}	271,346	503,270	799,151	1099,950	1329,56	1589,806	1858,935	2017,695
t_{20}	270,089	503,420	798,542	1099,885	1329,62	1589,323	1856,929	2017,828
t_{medio}	270,735	501,683	798,895	1100,814	1329,76	1589,433	1858,391	2017,031
σ	1,143	1,287	0,327	0,931	0,174	0,343	1,427	0,725

Legenda: Tempo de execução do SUFLP sequencial em função do número de fornecedores.

Utilizando as todas variações do cenário com 1668 cidades, foram realizadas, também, 20 execuções da versão paralela do algoritmo com quatro processadores. Os tempos expressos em segundos estão listados na Tabela 6. Além da média de tempos obtidos e o desvio padrão (σ), está listado na última linha a redução do tempo de execução obtida com a versão paralela em termos percentuais.

O desvio padrão foi inferior a 1 para todas as quantidades de fornecedores. Os tempos médios aumentaram a medida em que incrementamos a quantidade de fornecedores. O incremento, assim como no experimento anterior, foi sempre de 8 fornecedores.

A redução no tempo de execução obtida em todos os cenários variou entre aproximadamente 50% e 60% com a utilização de 4 processadores. Isto demonstra o potencial de escalabilidade da solução, ou seja, com a adição de mais processadores executando o programa em paralelo, espera-se que a redução do tempo seja ainda maior.

Tabela 6 - Execução do SUFLP paralelo com 1668 cidades.

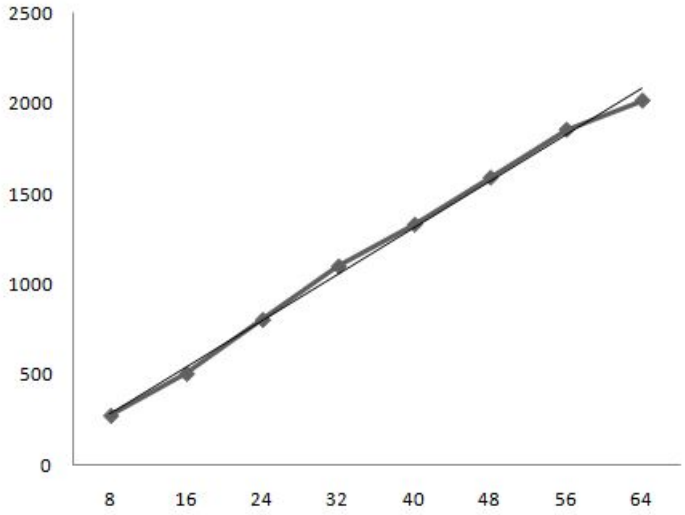
#	**8**	**16**	**24**	**32**	**40**	**48**	**56**	**64**
t_1	125,333	243,275	322,963	456,994	525,567	697,952	794,930	857,848
t_2	128,648	243,304	324,994	458,464	527,854	698,018	796,079	859,463
t_3	125,126	243,864	324,902	458,882	526,185	697,980	796,176	859,345
t_4	128,693	244,018	325,491	458,079	527,839	699,945	796,472	859,370
t_5	125,630	244,222	326,015	460,273	527,876	699,972	797,072	861,167
t_6	125,416	243,869	324,521	458,049	526,856	698,699	795,599	859,838
t_7	125,525	244,089	324,443	460,144	527,828	698,605	795,804	857,912
t_8	125,611	244,211	323,661	457,847	526,910	699,148	796,369	860,248
t_9	125,527	244,187	325,823	460,160	525,809	698,438	795,356	858,996
t_{10}	125,586	243,643	325,938	457,994	525,610	699,750	796,351	858,725
t_{11}	125,541	243,538	326,008	458,045	525,582	699,219	796,493	859,392
t_{12}	125,519	243,339	324,543	460,037	527,854	699,076	796,343	858,417
t_{13}	125,438	243,525	324,640	458,351	527,482	698,839	795,999	858,674
t_{14}	125,454	243,418	325,679	458,688	525,585	698,962	796,908	857,888
t_{15}	125,517	243,514	324,017	459,478	527,772	699,471	796,326	860,198
t_{16}	125,335	243,823	323,834	459,299	525,787	699,770	796,945	860,285
t_{17}	125,480	244,099	325,751	458,999	526,942	698,723	796,819	859,800
t_{18}	125,464	243,451	325,830	457,281	526,285	699,863	796,824	860,859
t_{19}	125,420	244,120	325,849	458,534	527,086	699,955	796,557	860,838
t_{20}	125,616	243,791	325,258	458,165	526,590	698,171	796,735	859,329
t_{medio}	125,794	243,765	325,008	458,688	526,765	699,028	796,308	859,430
σ	0,991	0,329	0,903	0,958	0,906	0,703	0,558	0,995
`Redução`	53,54%	51,41%	59,32%	58,33%	60,39%	56,02%	57,15%	57,39%

Legenda: Tempo de execução do SUFLP paralelo em função do número de fornecedores.

Para demonstrar o comportamento do tempo de execução de cada programa em função do aumento no número de cidades fornecedoras, foram criados os gráficos 4 e 5. O eixo vertical representa o tempo de execução enquanto o horizontal representa o número de fornecedores.

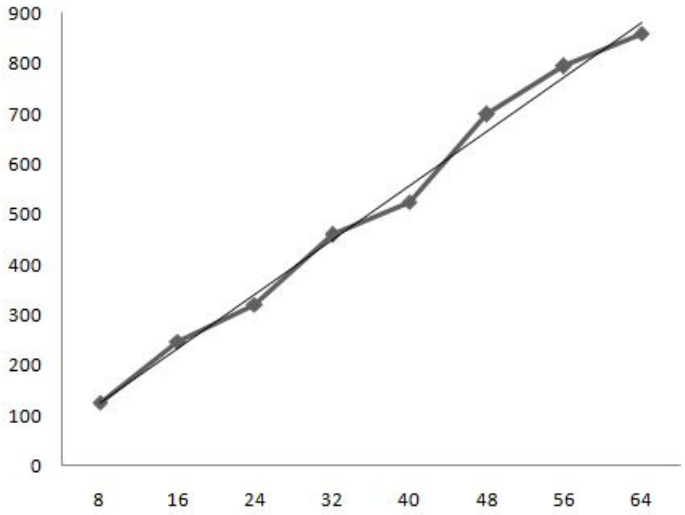
A linha de tendência de ambos os gráficos indica o crescimento linear do tempo de execução do algoritmo em função do aumento na quantidade de bases fornecedoras. É importante ressaltar que a implementação com o MPI permite a escalabilidade, porém, é preciso considerar a limitação da plataforma e da máquina utilizada. Portanto, é esperado que a execução do mesmo programa em máquinas com mais processadores disponíveis, ou mesmo *clusters*, reduza ainda mais o tempo de execução, além de tornar viável a avaliação de cenários com milhares de clientes e muitas centenas de fornecedores.

Figura 4 - Tempo de execução do SUFLP sequencial.



Legenda: Tempo de execução do SUFLP sequencial em função do número de fornecedores.

Figura 5 - Evolução do tempo de execução do SUFLP paralelo.



Legenda: Tempo de execução do SUFLP paralelo em função do número de fornecedores.

Cenário com 2856 cidades

No último cenário, foram consideradas as 2856 cidades existentes nas regiões Sul e Sudeste. De acordo com dados da ANP (ANP, 2014), existem bases distribuidoras presentes em 57 cidades destas regiões. Para este cenário, foram realizadas 20 execuções da versão paralela do algoritmo para cada variação da quantidade de processadores executando em paralelo, entre 1, 2, 3 e 4 processadores. Com os resultados obtidos neste experimento, desejamos observar a redução do tempo de processamento com o aumento de processos executando em paralelo. Os resultados obtidos estão listados na Tabela 7 e

estão expressos em segundos.

Tabela 7 - SUFLP com 2856 cidades e 57 fornecedores.

p	**1**	**2**	**3**	**4**
t_1	20376,114	11169,831	7796,567	5590,617
t_2	20110,395	11989,445	8037,934	5954,607
t_3	19519,653	11447,715	7911,710	5761,726
t_4	20393,247	11704,349	8250,769	6118,495
t_5	20103,135	11182,741	8050,654	5528,814
t_6	20118,137	11439,641	7707,003	6120,723
t_7	19649,748	11836,705	8274,989	5949,007
t_8	19648,815	11544,112	7548,845	5700,574
t_9	20419,480	11967,252	8140,877	5895,367
t_{10}	19839,253	11293,206	7719,793	5729,290
t_{11}	19475,903	11326,937	8137,488	6078,173
t_{12}	19636,897	12012,285	7962,951	5628,068
t_{13}	20449,487	11924,982	7686,544	6099,351
t_{14}	20235,230	11327,091	7800,057	5710,537
t_{15}	19489,582	11481,315	7718,647	5517,940
t_{16}	19697,187	11741,799	7813,127	5977,557
t_{17}	19534,090	11686,258	7560,887	5760,312
t_{18}	20115,970	11647,594	7682,300	5901,209
t_{19}	19661,772	11235,090	8140,960	5656,772
t_{20}	19788,295	11955,204	7941,439	5869,611
t_{medio}	19913,120	11595,678	7894,177	5827,437
σ	0,345	0,289	0,223	0,196

Legenda: Redução do tempo de execução do SUFLP paralelo em função do número de processadores.

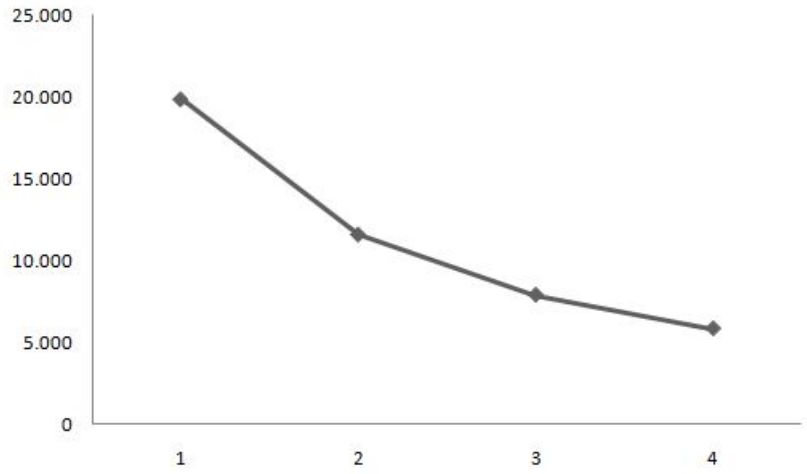
Os tempos médios de execução, assim como o desvio padrão obtido, respectivamente t_{medio} e σ, são mostrados nas duas últimas linhas dessa tabela. O desvio padrão obtido variou entre $0,196$ e $0,345$, indicando a consistência do tempo de execução do algoritmo em cenários maiores.

A Figura 6 mostra o tempo gasto com a execução do programa SUFLP na versão paralela em função do número de processadores executando com suporte da biblioteca MPI. A curva obtida mostra uma queda do tempo de execução em relação ao aumento de CPUs executando em paralelo. Contudo, essa queda não é constante. Isto se explica devido ao tempo gasto com a comunicação entre os processos executando o MPI.

A carga de trabalho foi distribuída igualmente entre os processadores. O resultado consolidado foi obtido através do comando `MPI_Allreduce`.

De modo a se avaliar o ganho computacional com a adição de processadores, foi calculada a razão entre o tempo de execução do programa sequencial T_s e o tempo de execução em paralelo T_p para cada quantidade de processadores utilizada. Esta métrica é conhecida como *Speedup*, ou $S_{(p)}$, onde p corresponde ao número de processadores utilizados em paralelo. O *Speedup* é definido da seguinte maneira:

Figura 6 - Tempo de execução do SUFLP com múltiplos processadores.



Legenda: Tempo de execução do SUFLP paralelo em função do número de processadores.

$$S_{(p)} = \frac{T_s}{T_p}$$

O valor de $S_{(p)}$ define o aumento no desempenho obtido com a técnica de paralelismo aplicada. Neste cenário, em particular, os valores de *Speedup* para cada variação na quantidade p de processadores foram calculados e estão listados na Tabela 8. De modo a ilustrar este critério, a Figura 7 apresenta a curva obtida com esses valores.

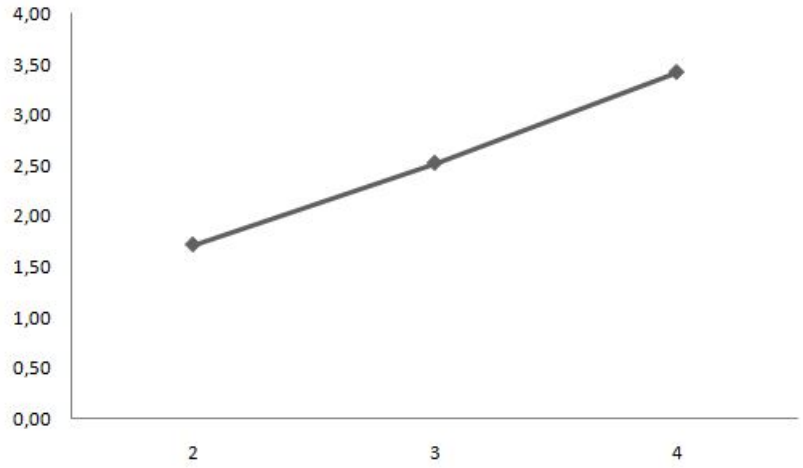
Tabela 8 - *Speedup* do SUFLP.

S_p	$\frac{T_s}{T_p}$	*Speedup*
S_2	$\frac{19913,120}{11595,678}$	1,717
S_3	$\frac{19913,120}{7894,177}$	2,523
S_4	$\frac{19913,120}{5827,437}$	3,417

Legenda: *Speedup* do SUFLP obtido com o paralelismo.

Observa-se a variação do *Speedup* entre $1,717$ e $3,417$, com tendência de cresci-

Figura 7 - *Speedup* do SUFLP com 2856 cidades.



Legenda: *Speedup* do SUFLP obtido com o paralelismo.

mento linear. Este critério indica que o algoritmo desenvolvido neste trabalho tem o potencial de ser paralelizado.

6 TRABALHOS RELACIONADOS

A decisão estratégica que envolve a escolha dos locais para instalar os terminais de expedição e armazenagem de produtos foi estudada por décadas em diversos trabalhos. Neste capítulo, são mencionados alguns estudos relevantes do problema da localização de instalações (PLI), suas principais características e as diferenças entre o que foi proposto por tais autores e o trabalho desenvolvido nesta dissertação.

Uma das abordagens para solução do PLI mais encontradas na literatura, o *branch and bound* foi inicialmente proposto por Efroymson e Ray (EFROYMSON; RAY, 1966) com o objetivo de desenvolver um algoritmo capaz de obter uma boa solução do problema em um tempo computacional viável [6]. O algoritmo proposto neste artigo se diferenciava dos outros até então publicados por buscar a solução ótima para o problema sem utilizar as técnicas de programação linear sugeridas por outros autores. Tais técnicas desconsideravam restrições, como por exemplo, a condição para que uma instalação estivesse aberta ou fechada completamente. Se considerarmos Y o conjunto de possíveis locais para estabelecer uma instalação, cada valor de seus elementos y, em qualquer solução encontrada, precisa ser necessariamente igual a 0 ou igual a 1. As técnicas de programação linear permitem encontrar soluções com valores de y fracionários, o que na prática torna o modelo impossível. Um valor facionário para y significa que uma base poderia estar parcialmente aberta, característica que o PLI não prevê.

Khumawala (KHUMAWALA, 1972) também aplicou o *branch and bound* ao SUFLP. O algoritmo proposto pelo autor busca a solução exata para o problema, ou seja, a quantidade e a localização dos armazéns dentro de uma malha logística de modo que o custo de implantação e operação seja o mínimo possível. Contudo, o próprio artigo conclui que o tempo computacional gasto na solução do problema aumenta exponencialmente conforme o aumento do conjunto de possíveis pontos de instalação. Isto torna a técnica *branch and bound* ineficiente se aplicada a malhas logísticas com um grande conjunto de localizações em potencial.

Tanto Efroymson e Ray quanto Khumawala estavam interessados em obter soluções ótimas para o PLI por meio da abordagem *branch and bound*. Porém, ambos demonstraram a preocupação com a eficiência do algoritmo em cadeias logísticas muito grandes (com muitas centenas de instalações em pontencial). Contudo, já na década de 1960, Leon Cooper (COOPER, 1964) se baseou em diferentes heurísticas para demonstrar a eficácia de outras técnicas na busca por uma solução boa, ou seja, aquela que se aproxima da

[6] O artigo foi publicado em 1966. Desde então, o avanço tecnológico aumentou as expectativas em termos do tempo computacional.

solução ótima, mas sem a necessidade de se calcular e avaliar todas as soluções possíveis. Os métodos aplicados por Cooper incluem o *Destination Subset Algorithm*, o *Random Destination Algorithm* e o *Alternate Location and Allocation Algorithm*.

O *Destination Subset Algorithm* tem como premissa separar os locais de destino em subconjuntos menores e estabelecer fornecedores nestes locais. O método assume que em algum destes subconjuntos reside uma solução aproximada para o problema. A dificuldade desta técnica está na determinação dos subconjuntos a serem avaliados, e não há garantia de que a solução ótima será encontrada. Já o *Random Destination Algorithm* também assume que os locais para instalações são um subconjunto dos locais onde há clientes, contudo, a quantidade de instalações n é determinada de forma aleatória, assim como a localização de cada uma delas. A partir destes pontos, é determinado quais clientes serão supridos por cada um dos fornecedores, com base no menor custo apresentado. Tal procedimento é repetido várias vezes até que a solução pare de apresentar valores melhores.

A heurística *Alternate Location and Allocation Algorithm*, cujo método consiste em variar os locais de fornecimento para subconjuntos de clientes previamente conhecidos, serviu como base para o algoritmo desenvolvido nesta dissertação. Uma das vantagens desta técnica é partir de uma solução já conhecida e conhecer os custos para cada variação do cenário. Esta informação é de grande utilidade em uma malha logística real, pois podem existir restrições de localização não definidas no modelo.

Na década de 1990, Shmoys, Tardos e Aardal (SHMOYS; TARDOS; AARDAL, 1997) desenvolveram um algoritmo baseado na equação do SUFLP e desconsiderando as restrições impostas à fração da demanda x_ij e a abertura ou fechamento total de uma instalação z_i situada em um local i. Tal algoritmo modifica as restrições de forma que $x_ij \geq 0$ e $z_i \geq 0$. Sem estas restrições, o algoritmo encontra uma solução ótima para a equação onde as variáveis x_ij e z_i são fracionárias e faz o arredondamento para valores inteiros. Contudo, não é possível provar que os valores inteiros formados após o arredondamento formam uma solução ótima. Em outro artigo publicado, Chudak e Shmoys (CHUDAK; SHMOYS, 2003) demonstraram que a aproximação dos valores fracionários para valores inteiros pode obter resultados melhores com a aplicação de algoritmos de aproximação, incluindo a aproximação aleatória. O artigo apresenta técnicas de aproximação e compara os resultados obtidos.

Em uma publicação mais recente, foram comparadas diferentes heurísticas para solução do PLI com restrição de capacidade (SAMBOLA; FERNANDEZ; LAPORTE, 2011). Além da restrição de capacidade, é também aplicada a restrição de distância – onde há uma distância limite entre o ponto de estabelecimento do fornecedor e o cliente. O objetivo deste trabalho era definir uma formulação capaz de resolver o problema a partir de algoritmos já existentes. Contudo, o artigo conclui que é muito difícil encontrar uma solução ótima para esta variação do problema. A melhor formulação avaliada pelos

autores se baseia na estratégia de encontrar limites máximos e mínimos para a solução, o que pode ser calculado mais rapidamente que todas as possíveis soluções.

É importante ressaltar que o modelo desenvolvido nesta dissertação tem como premissa a incidência tributária sobre no cálculo do custo total de distribuição. O cenário fiscal brasileiro, onde as alíquotas dependem diretamente da cidade e estado de origem e destino das mercadorias, foi analisado e inserido no modelo SUFLP. Os trabalhos avaliados neste capítulo consideram o custo de distribuição de maneira mais simples e linear.

Outra característica relevante proposta nesta dissertação foi a aplicação da técnica de paralelismo com utilização do ambiente MPI. Tal modelo permite a escalabilidade ao se adicionar mais processadores para a execução do programa. Isto permite que cenários com várias centenas ou até mesmo milhares de possíveis localizações sejam avaliados em um tempo computacional factível.

CONCLUSÕES

Neste trabalho, foi apresentado o fluxo de refino e distribuição de produtos derivados de petróleo no Brasil. Um dos principais problemas envolvendo grandes cadeias logísticas é a decisão sobre a quantidade e a localização dos terminais de distribuição de produtos, visando reduzir o custo total de operação e transporte.

Para atingir esse objetivo foi tratada a alocação dos recursos dentro de um território, considerando o problema da localização de instalações (PLI) estudado anteriormente por diferentes autores no âmbito da Pesquisa Operacional. Foram definidos modelos matemáticos específicos para cada ambiente avaliado. Critérios como capacidade de estoque, níveis na estrutura logística e variedade de produtos ofertados são algumas das características que servem para a definição da equação matemática que visa minimizar os custos de um conjunto de instalações.

Foram apresentados, também, os impostos incidentes sobre o transporte de cargas no território nacional, dentre os quais, destacam-se no escopo deste trabalho o ICMS e o ISS, por dependerem diretamente do local de embarque da mercadoria a ser transportada.

A proposta de adaptação do PLI aplicado a segmento de distribuição de combustíveis considerou diferentes critérios. Uma das premissas foi considerar a carga tributária aplicada ao transporte de cargas, que varia de forma não linear em função da origem e destino da mercadoria. Além disso, para o escopo deste trabalho, foi definido um modelo *singlecommodity* sem restrição de capacidade e sem níveis hierárquicos na cadeia de distribuição. Tais características se mostraram semelhantes a formulação do *Simple Uncapacited Facility Location Problem* (SUFLP), de tal forma que o modelo matemático proposto foi baseado nesta formulação.

Além disso, foi exposta a complexidade desse tipo de problema, bem como exemplos de estratégias aplicadas em sua solução. Dentre essas estratégias descrevemos *branch and bound* e heurísticas como *ADD* e *DROP*. Contudo, a abordagem *Alternate Location and Allocation* (ALA) desenvolvida originalmente por Rapp (RAPP, 1962) e revisada por Cooper (COOPER, 1964) mostrou-se uma alternativa computacionalmente viável para problemas com milhares de localizações em potencial.

Assim, o algoritmo proposto foi baseado no SUFLP com a heurística ALA. Essa proposta inclui uma modificação que avalia todos os locais inicialmente fechados como potenciais locais de instalação de fornecedores. Esse modificação provou ser eficaz ao demonstrar configurações com custo menor em comparação à configuração inicialmente estabelecida.

Os resultados obtidos nas execuções do problema demonstraram redução de até 19% do custo no cenário com 49 cidades. Além disso, foi obtido um ganho de desempenho maior do que 50% com a versão paralela do programa em conjunto com o MPI em uma

máquina capaz de executar 4 processos de forma simultânea. Os custos apresentados nos diversos cenários sugerem apenas um valor monetário estimado. Apesar dos valores utilizados serem reais, eles ainda necessitam uma análise mais aprofundada das tarifas de frete praticadas em cada região do país, bem como isenções de ICMS concedidas pelos estados.

Embora a adaptação do algoritmo ALA ao SUFLP não atinja necessariamente a uma solução ótima, isto é, aquela com o menor custo comprovadamente possível, ela representa uma alternativa viável em termos do tempo computacional. Segundo Jacobsen (JACOBSEN, 1983), técnicas consideradas melhores para a otimização de custos, como as heurísticas ADD e DROP, podem não convergir para uma boa solução em modelos sem restrição de capacidade. Outra técnica bastante estudada, o *branch and bound*, cuja solução é comprovadamente ótima, tem seu desempenho reduzido se aplicado a modelos com milhares de clientes e fornecedores.

Na prática, podemos considerar que os responsáveis pelo gerenciamento de cadeias logísticas estão mais interessados em conhecer várias opções com custos menores, comparados a configuração logística que já está em uso. A solução com menor custo operacional pode não ser viável por aspectos alheios ao interesse da companhia. Neste quesito, o ALA se mostra eficiente ao dispor cada configuração avaliada e seu respectivo custo de implantação, permitindo que os gestores de uma cadeia logística tomem decisões com base em diferentes opções.

O ambiente de execução utilizado na versão paralela do algoritmo, o MPI, é usualmente direcionado a ambientes com ampla oferta de processadores, geralmente dispostos em *clusters*. Contudo, as execuções do programa realizadas neste trabalho foram feitas em uma máquina simples, com apenas 4 processadores capazes de executar em paralelo. É fundamental que futuros trabalhos envolvendo o PLI em ambientes de computação paralela seja realizado com maior quantidade de máquinas, além de considerar todo o território nacional.

Sugestões para Trabalhos Futuros

A maioria dos estudos envolvendo o SUFLP foram realizados na década de 1960. Com o avanço tecnológico dos processadores obtido nos últimos anos, deve-se considerar o aumento da capacidade computacional disponível para execução de grandes massas de dados. Atualmente é factível pensar em algoritmos onde todos as possibilidades são avaliadas, visando obter o resultado ótimo. Para isso, é necessário desenvolver técnicas de programação combinatória aplicada ao cenário a ser avaliado.

O estudo do PLI suscita a necessidade de técnicas desenvolvidas especificamente para cada variação do problema, conforme restrições aplicáveis ao cenário analisado. No caso da logística de distribuição de combustíveis derivados de petróleo, deve-se pensar

em estudos do PLI abordando restrições como capacidade de estoque, oferta de múltiplos produtos por base e variação da demanda em função do tempo (*sazonalidade*). Existem outros segmentos da indústria do petróleo, como asfaltos e insumos para a indústria petroquímica, cuja demanda é concentrada em certas regiões do país.

REFERÊNCIAS

AIKENS, C.H. Facility location models for distribution planning. *European Journal of Operational Research*, Elsevier, Industrial Engineering Department, The University of Tennessee, Knoxville, TN, USA, v. 22, n. 3, p. 263–279, Dezembro 1985. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/0377221785902462⟩.

ANP. Perspectivas para o desenvolvimento do refino de petróleo no brasil. *Estudo Sobre o Desenvolvimento do Refino de Petróleo no Brasil*, Agência Nacional do Petróleo, Rio de Janeiro, 2002. Disponível em: ⟨www.anp.gov.br/?dw=2253⟩.

_____. Indústria nacional do petróleo, gás natural e biocombustíveis. *Anuário Estatístico Brasileiro do Petróleo, Gás Natural e Biocombustíveis*, Agência Nacional do Petróleo, Rio de Janeiro, 2011. Disponível em: ⟨www.anp.gov.br/?dw=57887⟩.

_____. Agência Nacional do Petróleo, 2014. Distribuidoras de Combustíeis Líquidos. Disponível em: ⟨http://www.anp.gov.br/?pg=66388&m=&t1=&t2=&t3=&t4=&ar=&ps=&cachebust=1397524168668⟩. Acesso em: 25 de Março de 2015.

ARGONNE NATIONAL LABORATORY. *Installation and User Guide to MPICH a Portable Implementation of MPI Version 1.2.5*. Chicago, 1998. 47 p.

BRASIL. LEI 9.478 DE 6 DE AGOSTO DE 1997. *Dispõe sobre a política energética nacional, as atividades relativas ao monopólio do petróleo, institui o Conselho Nacional de Política Energética e a Agência Nacional do Petróleo e dá outras providências.* Brasília, DF, 1997.

BRASIL. LEI COMPLEMENTAR 116 DE 31 DE JULHO DE 2003. *Dispõe sobre o Imposto Sobre Serviços de Qualquer Natureza, de competência dos Municípios e do Distrito Federal, e dá outras providências.* Brasília, DF, 2003.

BRASIL. LEI COMPLEMENTAR 87 DE 13 DE SETEMBRO DE 1996. *Dispõe sobre o imposto dos Estados e do Distrito Federal sobre operações relativas à circulação de mercadorias e sobre prestações de serviços de transporte interestadual e intermunicipal e de comunicação, e dá outras providências.* Brasília, DF, 1996.

CHUDAK, F.; SHMOYS, D. Improved approximation algorithms for the uncapacitated facility location problem. *SIAM Journal on Computing*, SIAM, v. 33, n. 1, p. 1–25, 2003. Disponível em: ⟨http://dx.doi.org/10.1137/S0097539703405754⟩.

COOPER, L. Heuristic methods for location-allocation problems. *SIAM Review*, Society for Industrial and Applied Mathematics, v. 6, n. 1, p. 37–53, 1964. Disponível em: ⟨http://www.jstor.org/stable/2027512⟩.

CORNUEJOLS, G.; SRIDHARAN, R.; THIZY, J. M. A comparison of heuristics and relaxations for the capacitated plant location problem. *European Journal of Operational Research*, Elsevier, v. 50, p. 280–297, 1991.

DUAILIB, A. K. *Combustíveis no Brasil*: Desafios e perspectivas. Centro de Estudos de Energia e Desenvolvimento - Rio de Janeiro: SYNERGIA, 2012. 297 p.

EFROYMSON, M. A.; RAY, T. L. A branch-bound algorithm for plant location. *Operations Research*, INFORMS, v. 14, n. 3, p. 361–368, 1966. Disponível em: ⟨http://www.jstor.org/stable/168193⟩.

FELDMAN, E.; LEHRER, F.A.; RAY, T.L. Warehouse location under continuous economies of scale. *Management Science*, INFORMS, p. 670–684, 1966. Disponível em: ⟨http://dx.doi.org.sci-hub.org/10.1287/mnsc.12.9.670⟩.

GABRIEL, Edgar et al. Open mpi: Goals, concept, and design of a next generation mpi implementation. In: *Proceedings, 11th European PVM/MPI Users' Group Meeting*. Budapest, Hungary: [s.n.], 2004. p. 97–104.

IBGE. 2013. Perfil dos Municípios Brasileiros 2013. Disponível em: ⟨http://loja.ibge.gov.br/perfil-dos-municipios-brasileiros-2013.html⟩. Acesso em: 12 de Maio de 2015.

IBP. 2014. Informações e Estatísticas da Indústria - Capacidade de Refino. Disponível em: ⟨http://200.189.102.61/SIEE/dashboard/CapacidadeDeRefino⟩. Acesso em: 07 de fevereiro de 2015.

IBPT. 2012. Evolução da Carga Tributária Brasileira e Previsão para 2013. Disponível em: ⟨https://www.ibpt.org.br/img/uploads/novelty/estudo/1443/20131218asscomEstudoEvolucaodacargatributariabrasileiraPrevisaopara2013.pdf⟩. Acesso em: 28 de Abril de 2015.

JACOBSEN, S. K. Heuristics for the capacited plant location model. *European Journal of Operational Research*, North-Holland Publishing Company, v. 12, n. 3, p. 253–261, 1983. Disponível em: ⟨http://libgen.org/scimag/get.php?doi=10.1016%2F0377-2217%2883%2990195-9⟩.

KAUFMAN, L.; EEDE, M.V.; HANSEN, P. A plant and warehouse location problem. *Operational Research Quarterly*, Palgrave Macmillan Journals, v. 28, n. 3, p. 547–554, 1977. Disponível em: ⟨http://www.jstor.org/stable/3008947⟩.

KHUMAWALA, Basheer M. An efficient branch and bound algorithm for the warehouse location problem. *Management Science*, INFORMS, New York, v. 18, n. 12, p. 718, Agosto 1972. Disponível em: ⟨http://www.jstor.org/stable/2629558⟩.

KUEHN, A.A.; HAMBURGER, M.J. A heuristic program for locating warehouses. *Management Science*, INFORMS, Carnegie Institute of Technology, Pittsburg, Pennsylvania, v. 9, n. 4, p. 643–666, Julho 1963. Disponível em: ⟨http://www.jstor.org/stable/2627368⟩.

MME. Balanco energetico nacional. *Relatório Síntese do Balanço Energético Nacional*, Ministério de Minas e Energia, Rio de Janeiro, 2014. Disponível em: ⟨https://ben.epe.gov.br/BENRelatorioSintese2014.aspx⟩.

PRICE WATERHOUSE & COOPERS AND THE WORLD BANK. 2008. Paying Taxes 2008 - The Global Picture. Disponível em: ⟨http://www.pwc.com/gx/en/paying-taxes/assets/paying-taxes-2008.pdf⟩. Acesso em: 14 de fevereiro de 2015.

RAPP, Y. Planning of exchange locations and boundaries. *Ericsson Technics*, Ericsson Technics, v. 2, p. 1–22, 1962.

SAMBOLA, M. A.; FERNANDEZ, E.; LAPORTE, G. A computacional comparison of several models for the exact solution of the capacity and distance constrained plant location problem. *Computers and Operations Research*, Elsevier, v. 38, n. 8, p. 1109–1116, 2011. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0305054810002613⟩.

SHMOYS, D.; TARDOS, E.; AARDAL, K. Approximation algorithms for facility location problems. *29th ACM Symposium on Theory of Computing*, ACM, p. 265–274, 1997. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=258600⟩.

SINDICOM. 2014. Estrutura do Setor de Combustíveis no Brasil. Disponível em: ⟨http://www.sindicom.com.br/images/file/combustiveis/Estrutura_Setor_Combustiveis_20140402.pdf⟩. Acesso em: 25 de Março de 2015.

TCHA, D.; LEE, B. A branch-and-bound algorithm for the multi-level uncapacitated facility location problem. *European Journal of Operational Research*, Elsevier, Korea Advanced Institute of Science and Technology, Chongyang Seoul, Korea, v. 18, p. 35–43, Setembro 1984. Disponível em: ⟨http://www.sciencedirect.com.sci-hub.org/science/article/pii/0377221784902583⟩.

UNIVERSITY OF SAN FRANCISCO. *A User's Guide to MPI*. San Francisco, CA, 1998. 51 p.

WARSZAWSKI, A. Multidimensional location problems. *Operational Research Quarterly*, Palgrave Macmillan Journals, v. 24, n. 2, p. 165–179, 1973. Disponível em: ⟨http://www.palgrave-journals.com/jors/journal/v24/n2/pdf/jors197335a.pdf⟩.

APÊNDICE A – Código fonte do programa `SUFLP_simples.c`

```
1  /* Experimentacao do problema da localizacao de instalacoes */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5  #include <ctype.h>
6  #include <mpi.h>
7
8  /* Variaveis Globais */
9  struct cidades{ long int cidadeCliente; char UF[2]; char nomeCidade[100];
10                 int ativa; int ISS; long int custoFixo; long int
                       cidadeFornecedora;
11                 float melhorCusto; float imposto; };
12
13 long int colunas = 0;
14 long int *vetorDistancia;
15 long int *vetorBases;          //armazena bases existentes
16 long int *vetorBasesInativas; //armazena cidades sem bases estabelecidas
17 int qtdBases;                  //qtd bases diferentes estabelecidas
18 int qtdBasesInativas;          //qtd cidades sem bases estabelecidas
19
20 FILE *distancias;
21 FILE *cidadesDistribuidoras;
22 FILE *configuracaoFinalCidades;
23 FILE *logExecucao;
24
25 int main (int argc, char** argv)
26 {
27     /************************************************/
28     /* Incidencia de ICMS entre os estados
29     /* Primeira linha e primeira coluna correspondem
30     /* ao codigo IBGE
31     /************************************************/
32     int matrizICMS[28][28] = {
33     { 0,12,27,13,16,29,23,53,32,52,21,51,50,31,15,25,41,
           26,22,24,43,33,11,14,42,35,28,17},
34     {12,17,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
           12,12,12,12,12,12,12,12,12,12,12,12},
35     {27,12,17,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
           12,12,12,12,12,12,12,12,12,12,12,12},
36     {13,12,12,17,12,12,12,12,12,12,12,12,12,12,12,12,12,
           12,12,12,12,12,12,12,12,12,12,12,12},
37     {16,12,12,12,17,12,12,12,12,12,12,12,12,12,12,12,12,
           12,12,12,12,12,12,12,12,12,12,12,12},
```

```
38      {29,12,12,12,12,17,12,12,12,12,12,12,12,12,12,12,12,
            12,12,12,12,12,12,12,12,12,12,12,12},
39      {23,12,12,12,12,12,17,12,12,12,12,12,12,12,12,12,12,
            12,12,12,12,12,12,12,12,12,12,12,12},
40      {53,12,12,12,12,12,12,17,12,12,12,12,12,12,12,12,12,
            12,12,12,12,12,12,12,12,12,12,12,12},
41      {32,12,12,12,12,12,12,12,17,12,12,12,12,12,12,12,12,
            12,12,12,12,12,12,12,12,12,12,12,12},
42      {52,12,12,12,12,12,12,12,12,17,12,12,12,12,12,12,12,
            12,12,12,12,12,12,12,12,12,12,12,12},
43      {21,12,12,12,12,12,12,12,12,12,17,12,12,12,12,12,12,
            12,12,12,12,12,12,12,12,12,12,12,12},
44      {51,12,12,12,12,12,12,12,12,12,12,12,17,12,12,12,12,12,
            12,12,12,12,12,12,12,12,12,12,12,12},
45      {50,12,12,12,12,12,12,12,12,12,12,12,12,17,12,12,12,12,
            12,12,12,12,12,12,12,12,12,12,12,12},
46      {31, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,18, 7, 7,12,  7, 7, 7,12,12, 7,
             7,12,12, 7, 7},
47      {15,12,12,12,12,12,12,12,12,12,12,12,12,12,12,17,12,12,
            12,12,12,12,12,12,12,12,12,12,12,12},
48      {25,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,17,12,
            12,12,12,12,12,12,12,12,12,12,12,12},
49      {41, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,12, 7, 7,18,  7, 7, 7,12,12, 7,
             7,12,12, 7, 7},
50      {26,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
            17,12,12,12,12,12,12,12,12,12,12,12},
51      {22,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
            12,17,12,12,12,12,12,12,12,12,12,12},
52      {24,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
            12,12,17,12,12,12,12,12,12,12,12,12},
53      {43, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,12, 7, 7,12,  7, 7, 7,17,12, 7,
             7,12,12, 7, 7},
54      {33, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,12, 7, 7,12,  7, 7, 7,12,19, 7,
             7,12,12, 7, 7},
55      {11,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
            12,12,12,12,12,17,12,12,12,12,12},
56      {14,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
            12,12,12,12,12,12,17,12,12,12,12},
57      {42, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,12, 7, 7,12,  7, 7, 7,12,12, 7,
             7,17,12, 7, 7},
58      {35, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,12, 7, 7,12,  7, 7, 7,12,12, 7,
             7,12,18, 7, 7},
59      {28,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
            12,12,12,12,12,12,12,12,12,12,17,12},
60      {17,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
            12,12,12,12,12,12,12,12,12,12,12,17}};
61
```

```
62      long int aux;
63      int linhas = 0;
64      char *result;
65      char linha[25000];
66      char *token;
67      time_t inicio, fim, fim_log;
68
69      printf("\nIniciando simulacao\n");
70      inicio = time(NULL);
71
72      distancias = fopen("C:\\Thiago\\TEMP\\sudesteReduzido.csv","r");
73      rewind(distancias);
74      result = fgets(linha, 100000, distancias);
75      token = strtok(linha, ";");
76      colunas++;
77
78      while (token != NULL)
79      {
80            token = strtok(NULL, ";");
81            colunas++;
82      }
83
84      /************************************************/
85      /* Aloca o ponteiro vetorDistancia para
86      /* conter toda a matriz
87      /************************************************/
88      printf("Alocando vetor de distancias\n");
89      vetorDistancia = malloc((colunas*colunas) * sizeof(long int));
90      //vetorCidades   = malloc((colunas - 2) * sizeof(struct cidades));
91      struct cidades vetorCidades[colunas - 2];
92      //Atualiza o n de colunas descontando a posicao 0
93      colunas = colunas - 1;
94
95      int k = 0;
96      rewind(distancias);
97      while (result != NULL)
98      {
99            result = fgets(linha, sizeof(linha), distancias);
100           token = strtok(linha, ";");
101           if (token == '\0' || token == '\n') {continue;}
102           aux = atoi(token);
103           if(aux) { *(vetorDistancia + k) = aux; }
104           else { *(vetorDistancia + k) = 999; }
105           k++;
106
107           while (token != NULL)
108           {
```

```
109             token = strtok(NULL, ";");
110             if (token == '\0' || token == '\n') {continue;}
111             aux = atoi(token);
112             if(aux) { *(vetorDistancia + k) = aux; }
113             else { *(vetorDistancia + k) = 999; }
114             k++;
115         }
116     }
117
118     /************************************************/
119     /* Aloca o ponteiro vetorCidades para conter as
120     /* informacoes sobre cidades, aliquota de ISS
121     /* e qual base supre sua demanda
122     /************************************************/
123     printf("Alocando vetor de cidades\n");
124     cidadesDistribuidoras =
125        fopen("C:\\Thiago\\TEMP\\cidadesDistribuidorasReduzido.csv","r");
126
127     k = 0;
128     result = 1;
129     rewind(cidadesDistribuidoras);
130     while (result != NULL)
131     {
132           result = fgets(linha, sizeof(linha), cidadesDistribuidoras);
133           //1 - codigo da cidade
134           token = strtok(linha, ";");
135           if (token == '\0' || token == '\n') {continue;}
136           else { vetorCidades[k].cidadeCliente = atoi(token); }
137           //2 - codigo do estado
138           token = strtok(NULL, ";");
139           if (token == '\0' || token == '\n') {continue;}
140           else { strcpy(vetorCidades[k].UF, token); }
141           //3 - nome da cidade
142           token = strtok(NULL, ";");
143           if (token == '\0' || token == '\n') {continue;}
144           else { strcpy(vetorCidades[k].nomeCidade, token); }
145           //4 - Flag de ativacao
146           token = strtok(NULL, ";");
147           if (token == '\0' || token == '\n') {continue;}
148           else { vetorCidades[k].ativa = atoi(token); }
149           //5 - aliquota ISS
150           token = strtok(NULL, ";");
151           if (token == '\0' || token == '\n') {continue;}
152           else { vetorCidades[k].ISS = atoi(token); }
153           //6 - Custo fixo de implantacao
154           token = strtok(NULL, ";");
155           if (token == '\0' || token == '\n') {continue;}
```

```
156             else { vetorCidades[k].custoFixo = atoi(token); }
157             //7 - Cidade Fornecedora
158             token = strtok(NULL, ";");
159             if (token == '\0' || token == '\n') {continue;}
160             else { vetorCidades[k].cidadeFornecedora = atoi(token); }
161             //8 - Melhor Custo
162             token = strtok(NULL, ";");
163             if (token == '\0' || token == '\n') {continue;}
164             else { vetorCidades[k].melhorCusto = atof(token); }
165             //9 - Imposto
166             token = strtok(NULL, ";");
167             if (token == '\0' || token == '\n') {continue;}
168             else { vetorCidades[k].imposto = atof(token); }
169 
170             k++;
171         }
172         fclose(cidadesDistribuidoras);
173 
174         /*************************************************/
175         /* Determina quantas bases distribuidoras existem.
176         /* Determina quantas cidades sem bases existem.
177         /* - Cria vetor com bases ativas
178         /* - Cria vetor com cidades onde nao ha bases
179         /*************************************************/
180         int i;
181         int j;
182         //determina a quantidade de bases distintas
183         for (i = 0; i < k; i++)
184         {
185             if (vetorCidades[i].ativa == 1) { qtdBases++; }
186             else { qtdBasesInativas++; }
187         }
188         // aloca os vetores de bases e de cidades sem base
189         vetorBases = malloc((qtdBases) * sizeof(long int));
190         vetorBasesInativas = malloc((qtdBasesInativas) * sizeof(long int));
191 
192         //guarda cidades com bases distintas no vetor de bases
193         j = 0;
194         for (i = 0; i < k; i++)
195         {
196             if (vetorCidades[i].ativa == 1)
197             {
198                 *(vetorBases + j) = vetorCidades[i].cidadeCliente;
199                 j++;
200             }
201         }
202         //guarda cidades sem bases estabelecidas em um vetor
```

```
203     j = 0;
204     for (i = 0; i < k; i++)
205     {
206         if (vetorCidades[i].ativa != 1)
207         {
208             *(vetorBasesInativas + j) = vetorCidades[i].cidadeCliente;
209             j++;
210         }
211     }
212
213     printf("Cidades Fornecedoras: %d\n", qtdBases);
214     i = 0;
215     while (i < qtdBases)
216     {
217           printf("%d\n", *(vetorBases + i));
218           i++;
219     }
220
221     printf("Cidades sem Bases Fornecedoras: %d\n", qtdBasesInativas);
222     i = 0;
223     while (i < qtdBasesInativas)
224     {
225           printf("%d\n", *(vetorBasesInativas + i));
226           i++;
227     }
228
229     /*************************************************/
230     /* Percorre matriz determinando a melhor opcao dentre
231     /* as cidades fornecedoras para cada cidade cliente
232     /* ----------------------------------------------
233     /* Faz o teste antes e depois de aplicar o imposto
234     /* de modo a descobrir as vantagens competitivas
235     /* relacionadas aos tributos aplicados pelos estados
236     /*************************************************/
237     j = 0;
238     int t = 0;
239     int temp = 0;
240     int aliquota = 0;
241     float sumCustoDistancia = 0;
242     float custo = 0;
243     float imposto = 0;
244     float sumCustoFixo = 0;
245     float sumImposto = 0;
246     float custoTotalEfetivo = 0;
247     int indiceOrigem = 0;
248     int indiceDestino = 0;
249     long int codigoOrigem  = 0;
```

```
250     long int codigoDestino = 0;
251     long int origemDistancias, destinoDistancias;
252
253     printf("Abrindo arquivo de cidades\n");
254     configuracaoFinalCidades =
255        fopen("C:\\Thiago\\TEMP\\configuracaoFinalCidades.txt","w+");
256     printf("\nDeterminando configuracao inicial cidade versus fornecedor\n"
           );
257     i = 0;
258     while (i < k)   //Le o vetor cidades, pega destino
259     {
260           codigoDestino = vetorCidades[i].cidadeCliente;
261           // determina o menor custo entre as cidades
262           // fornecedoras existentes.
263           int r = 0;
264           while (r < qtdBases)
265           {
266                 codigoOrigem = *(vetorBases + r);
267                 if (codigoOrigem == codigoDestino)
268                 {
269                    aliquota = vetorCidades[i].ISS;
270                 }
271                 else
272                 {
273                    //encontra indice da origem e destino na matrizICMS
274                    j = 0;
275                    while (matrizICMS[j][0] != (*(vetorBases + r)/100000))
                          { j++; }
276                    indiceOrigem = j;
277                    j = 0;
278                    while (matrizICMS[0][j] != (codigoDestino/100000)) { j
                          ++; }
279                    indiceDestino = j;
280                    aliquota = matrizICMS[indiceOrigem][indiceDestino];
281                 }
282
283                 // varre a matriz de distancias, aplicando a aliquota,
                       fazendo
284                 // o somatorio do custo entre todos os pares Fornecedor Vs.
                        Cliente
285                 for (j = 1; j < colunas; j++)
286                 {
287                     destinoDistancias = (long int) *(vetorDistancia + j);
288                     if (destinoDistancias == codigoDestino)
289                     {
290                        for (t = colunas; t < (colunas*colunas); t = (t +
                              colunas))
```

```
291                         {
292                             origemDistancias = (long int) *(vetorDistancia +
                                    t);
293                             if (origemDistancias == codigoOrigem)
294                             {
295                                 temp = (t + j);
296                                 custo = *(vetorDistancia + temp);
297                                 imposto = custo * aliquota / 100;
298 
299                                 // Se este fornecedor possui um custo menor,
                                        atualiza o vetor
300                                 if ((custo + imposto) < vetorCidades[i].
                                        melhorCusto)
301                                 {
302                                     vetorCidades[i].melhorCusto = (custo +
                                            imposto);
303                                     vetorCidades[i].cidadeFornecedora =
                                            codigoOrigem;
304                                     vetorCidades[i].imposto = imposto;
305                                     break;
306                                 }
307                             }
308                         }
309                     }
310                 }
311                 r++;
312         }
313     i++;
314 }
315 
316 i = 0;
317 while (i < k) // Soma custo total e grava configuracao inicial
318 {
319         // Se a cidade tem base distribuidora, acumula o respectivo
320         // custo fixo de implantacao para ser somado na equacao do SUFLP
321         if (vetorCidades[i].ativa == 1)
322         {
323             sumCustoFixo = sumCustoFixo + (float) vetorCidades[i].
                    custoFixo;
324         }
325 
326         // Calcula valor total do imposto nesta configuracao
327         sumImposto = sumImposto + vetorCidades[i].imposto;
328         // calcula o custo total efetivo
329         custoTotalEfetivo = custoTotalEfetivo + (sumCustoFixo +
                vetorCidades[i].melhorCusto);
330         fprintf(configuracaoFinalCidades,
```

```
331             "Cliente: %d - Melhor Fornecedor: %d - Melhor Custo: %4.2f -
                    Imposto: %3.2f\n",
332                                             vetorCidades[i].cidadeCliente,
333                                             vetorCidades[i].
                                                    cidadeFornecedora,
334                                             vetorCidades[i].melhorCusto,
335                                             vetorCidades[i].imposto);
336         i++;
337     }
338
339     /************************************************/
340     /* Apos a configuracao inicial com a melhor cidade
341     /* fornecedora, vamos testar o custo total efetivo
342     /* em cada cenario, variando uma cidade fornecedora
343     /* de cada vez, mantendo as demais fixas.
344     /************************************************/
345     int r, p;
346     int flagBreak = 0;
347     int cont = 0;
348     long int baseTemp;
349     float melhorCusto = custoTotalEfetivo;
350     float melhorImposto = sumImposto;
351
352     printf(
353        "Custo total efetivo inicial: %.2f - Imposto: %.2f\n",
               custoTotalEfetivo, sumImposto);
354     fprintf(configuracaoFinalCidades,
355        "Custo total efetivo inicial: %.2f - Imposto: %.2f\n",
               custoTotalEfetivo, sumImposto);
356     printf("\a");
357     printf("\nIniciando testes com variacao dos fornecedores:\n");
358
359     r = 0;
360     temp = 0;
361     while (r < qtdBases) //Para cada cidade fornecedora diferente
362     {
363         p = 0;
364         while (p < qtdBasesInativas) // para cada base inativa
365         {
366             // guarda a base original para recolocar no vetor de
                    cidades
367             if (p == 0)
368             {
369                 baseTemp = *(vetorBases + r);
370             }
371             // substitui uma cidade fornecedora de cada vez, para todos
372             // as cidades que estavam na lista de "inativas"
```

```
373                 i = 0;
374                 while (i < k)
375                 {
376                       if (vetorCidades[i].cidadeFornecedora == *(vetorBases
                              + r))
377                       {
378                          vetorCidades[i].cidadeFornecedora = *(
                              vetorBasesInativas + p);
379                       }
380                       i++;
381                 }
382
383                 *(vetorBases + r) = *(vetorBasesInativas + p);
384
385                 // Agora testa o SUFLP com esta configuracao
386                 // ********** INICIO SUFLP ***************
387                 custo = 0;
388                 imposto = 0;
389                 sumCustoDistancia = 0;
390                 sumCustoFixo = 0;
391                 sumImposto = 0;
392                 flagBreak = 0;
393                 custoTotalEfetivo = 0;
394
395                 i = 0;
396                 while (i < k)   //Le vetorCidades, pega origem e destino
397                 {
398                       codigoDestino = vetorCidades[i].cidadeCliente;
399                       codigoOrigem = vetorCidades[i].cidadeFornecedora;
400
401                       if (codigoDestino == codigoOrigem)
402                       {
403                          aliquota = vetorCidades[i].ISS;
404                       }
405                       else //encontra indice da origem e destino na
                              matrizICMS
406                       {
407                          j = 0;
408                          while (matrizICMS[j][0] != (codigoOrigem / 100000)
                                 ) { j++; }
409                          indiceOrigem = j;
410                          j = 0;
411                          while (matrizICMS[0][j] != (codigoDestino /
                              100000)) { j++; }
412                          indiceDestino = j;
413                          aliquota = matrizICMS[indiceOrigem][indiceDestino
                              ];
```

```
414                     }
415
416                     // Se a cidade tem base distribuidora, acumula o
                            respectivo
417                     // custo fixo de implantacao para ser somado na
                            equacao do SUFLP
418                     if (vetorCidades[i].ativa == 1)
419                     {
420                        sumCustoFixo = sumCustoFixo + (float) vetorCidades
                               [i].custoFixo;
421                     }
422
423                     // varre a matriz de distancias, aplicando a aliquota
                            , fazendo
424                     // o somatorio do custo entre todos os pares
                            Fornecedor Vs. Cliente
425                     for (j = 1; j < colunas; j++)
426                     {
427                         destinoDistancias = (long int) *(vetorDistancia +
                                j);
428                         if (destinoDistancias == codigoDestino)
429                         {
430                            for (t = colunas; t < (colunas*colunas); t = (
                                   t + colunas))
431                            {
432                                origemDistancias = (long int) *(
                                       vetorDistancia + t);
433                                if (origemDistancias == codigoOrigem)
434                                {
435                                   temp = (t + j);
436                                   custo = *(vetorDistancia + temp);
437                                   imposto = custo * aliquota / 100;
438                                   sumImposto = sumImposto + imposto;
439                                   sumCustoDistancia = sumCustoDistancia +
                                          (custo + imposto);
440                                   vetorCidades[i].melhorCusto = (custo +
                                       imposto);
441                                   vetorCidades[i].imposto = imposto;
442                                   break;
443                                }
444                            }
445                         }
446                     }
447
448                     // Reducao: Se somatorio ja ultrapassou o melhor,
449                     // descarta este fornecedor e passa ao proximo sem
450                     // concluir o loop.
```

```
451                 if (sumCustoDistancia > melhorCusto)
452                 {
453                     printf("Break: Custo ultrapassado na iteracao %d
                            da rodada #%d\n", i, (cont + 1));
454                     fprintf(configuracaoFinalCidades,
455                     "Break: Custo ultrapassado na iteracao %d da
                            rodada #%d\n", i, (cont + 1));
456                     break;
457                 }
458
459                 i++;
460             }
461
462             // Grava a configuracao testada nesta iteracao e seu
                    respectivo custo
463             cont++;
464             printf("\nResultados da rodada #%d\n", cont);
465             fprintf(configuracaoFinalCidades, "\nResultados da rodada #
                    %d\n", cont);
466
467             i = 0;
468             while (i < k)
469             {
470                 fprintf(configuracaoFinalCidades, "Cliente: %d -
                        Fornecedor: %d\n",
471                                                 vetorCidades[i].
                                                    cidadeCliente,
472                                                 vetorCidades[i].
                                                    cidadeFornecedora
                                                    );
473                 i++;
474             }
475
476             custoTotalEfetivo = sumCustoDistancia + sumCustoFixo;
477             printf(">>> Custo Total Efetivo: %.2f - Impostos: %.2f",
                    custoTotalEfetivo, sumImposto);
478             fprintf(configuracaoFinalCidades,
479                "">>> Custo Total Efetivo: %.2f - Impostos: %.2f",
                        custoTotalEfetivo, sumImposto);
480
481             if (custoTotalEfetivo < melhorCusto)
482             {
483                 printf(" - Melhor custo encontrado: %.2f - Imposto: %.2f
                        #%d \n",
484                     custoTotalEfetivo, sumImposto, cont);
485                 fprintf(configuracaoFinalCidades, " - Melhor custo
                        encontrado! #%d \n", cont);
```

```
486                     melhorCusto = custoTotalEfetivo;
487                 }
488                 else
489                 {
490                     printf("\n");
491                     fprintf(configuracaoFinalCidades, "\n");
492                 }
493 
494                 // ********** FIM SUFLP *****************
495 
496                 // Recoloca a base fornecedora original
497                 if ((p + 1) == qtdBasesInativas)
498                 {
499                     *(vetorBases + r) = baseTemp;
500                     i = 0;
501                     while (i < k)
502                     {
503                         if (vetorCidades[i].cidadeFornecedora == *(
                                vetorBasesInativas + p))
504                         {
505                             vetorCidades[i].cidadeFornecedora = *(vetorBases +
                                    r);
506                         }
507                         i++;
508                     }
509                 }
510 
511                 p++;
512             }
513             r++;
514         }
515 
516         /************************************************/
517         /* comandos para finalizacao do programa,
518         /* registrar tempo, gravar LOG e fechar arquivos
519         /************************************************/
520         fim = time(NULL);
521         fim_log = time(NULL);
522         printf("\n\nfinalizando simulacao\n");
523         printf("Tempo de execucao: %.2f segundos.\n", difftime(inicio,fim) *
                (-1));
524         printf("\nGravando LOG\n");
525         logExecucao = fopen("C:\\Thiago\\TEMP\\log.txt","a+");
526         fseek(logExecucao, 0, SEEK_END);
527         fprintf(logExecucao, "Tempo de execucao: %14.2f segundos - %s",
                difftime(inicio,fim) * (-1), ctime(&fim_log));
528         fclose(logExecucao);
```

```
529     fclose(distancias);
530     fclose(configuracaoFinalCidades);
531     printf("\nLOG regitrado com sucesso.\n");
532     getch();
533     return (0);
534 }
```

APÊNDICE B – Código fonte do programa `SUFLP_paralelo.c`

```
1 /* Experimentacao do problema da localizacao de instalacoes */
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <time.h>
5 #include <ctype.h>
6 #include <mpi.h>
7
8 /* Variaveis Globais */
9 struct cidades{ long int cidadeCliente; char UF[2]; char nomeCidade[100];
10                int ativa; int ISS; long int custoFixo; long int
                      cidadeFornecedora;
11                float melhorCusto; float imposto; };
12
13 long int colunas = 0;
14 long int *vetorDistancia;
15 long int *vetorBases;          //armazena bases existentes
16 long int *vetorBasesInativas; //armazena cidades sem bases estabelecidas
17 int qtdBases;                  //qtd bases diferentes estabelecidas
18 int qtdBasesInativas;          //qtd cidades sem bases estabelecidas
19
20 //struct cidades *vetorCidades;
21 FILE *distancias;
22 FILE *cidadesDistribuidoras;
23 FILE *configuracaoFinalCidades;
24 FILE *logExecucao;
25
26 int main (int argc, char** argv)
27 {
28     /*************************************************/
29     /* Incidencia de ICMS entre os estados
30     /* Primeira linha e primeira coluna correspondem
31     /* ao codigo IBGE
32     /*************************************************/
33     int matrizICMS[28][28] = {
34     { 0,12,27,13,16,29,23,53,32,52,21,51,50,31,15,25,41,
          26,22,24,43,33,11,14,42,35,28,17},
35     {12,17,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
          12,12,12,12,12,12,12,12,12,12,12,12},
36     {27,12,17,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
          12,12,12,12,12,12,12,12,12,12,12,12},
37     {13,12,12,17,12,12,12,12,12,12,12,12,12,12,12,12,12,
          12,12,12,12,12,12,12,12,12,12,12,12},
38     {16,12,12,12,17,12,12,12,12,12,12,12,12,12,12,12,12,
```

```
        12,12,12,12,12,12,12,12,12,12,12},
39      {29,12,12,12,12,17,12,12,12,12,12,12,12,12,12,12,12,
        12,12,12,12,12,12,12,12,12,12,12},
40      {23,12,12,12,12,12,17,12,12,12,12,12,12,12,12,12,12,
        12,12,12,12,12,12,12,12,12,12,12},
41      {53,12,12,12,12,12,12,17,12,12,12,12,12,12,12,12,12,
        12,12,12,12,12,12,12,12,12,12,12},
42      {32,12,12,12,12,12,12,12,17,12,12,12,12,12,12,12,12,
        12,12,12,12,12,12,12,12,12,12,12},
43      {52,12,12,12,12,12,12,12,12,17,12,12,12,12,12,12,12,
        12,12,12,12,12,12,12,12,12,12,12},
44      {21,12,12,12,12,12,12,12,12,12,17,12,12,12,12,12,12,
        12,12,12,12,12,12,12,12,12,12,12},
45      {51,12,12,12,12,12,12,12,12,12,12,17,12,12,12,12,12,
        12,12,12,12,12,12,12,12,12,12,12},
46      {50,12,12,12,12,12,12,12,12,12,12,12,17,12,12,12,12,
        12,12,12,12,12,12,12,12,12,12,12},
47      {31, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,18, 7, 7,12,  7, 7, 7,12,12, 7,
         7,12,12, 7, 7},
48      {15,12,12,12,12,12,12,12,12,12,12,12,12,12,17,12,12,
        12,12,12,12,12,12,12,12,12,12,12},
49      {25,12,12,12,12,12,12,12,12,12,12,12,12,12,12,17,12,
        12,12,12,12,12,12,12,12,12,12,12},
50      {41, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,12, 7, 7,18,  7, 7, 7,12,12, 7,
         7,12,12, 7, 7},
51      {26,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
        17,12,12,12,12,12,12,12,12,12,12},
52      {22,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
        12,17,12,12,12,12,12,12,12,12,12},
53      {24,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
        12,12,17,12,12,12,12,12,12,12,12},
54      {43, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,12, 7, 7,12,  7, 7, 7,17,12, 7,
         7,12,12, 7, 7},
55      {33, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,12, 7, 7,12,  7, 7, 7,12,19, 7,
         7,12,12, 7, 7},
56      {11,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
        12,12,12,12,12,17,12,12,12,12,12},
57      {14,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
        12,12,12,12,12,12,17,12,12,12,12},
58      {42, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,12, 7, 7,12,  7, 7, 7,12,12, 7,
         7,17,12, 7, 7},
59      {35, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,12, 7, 7,12,  7, 7, 7,12,12, 7,
         7,12,18, 7, 7},
60      {28,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
        12,12,12,12,12,12,12,12,12,17,12},
61      {17,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
        12,12,12,12,12,12,12,12,12,12,17}};
```

```
62
63      long int aux;
64      int linhas = 0;
65      char *result;
66      char linha[25000];
67      char *token;
68      time_t inicio, fim, fim_log;
69
70      int rank, numtasks;
71      MPI_Init(&argc,&argv);
72      MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
73      MPI_Comm_rank(MPI_COMM_WORLD,&rank);
74
75      printf("\nIniciando simulacao\n");
76      inicio = time(NULL);
77
78      distancias = fopen("C:\\Thiago\\TEMP\\Sudeste_somente_codigos.csv","r")
            ;
79      if (distancias == NULL ) { printf("P%d - Erro ao abrir distancias\n",
            rank); }
80      rewind(distancias);
81      result = fgets(linha, 100000, distancias);
82      token = strtok(linha, ";");
83      colunas++;
84
85      while (token != NULL)
86      {
87            token = strtok(NULL, ";");
88            colunas++;
89      }
90
91      /************************************************/
92      /* Aloca o ponteiro vetorDistancia para
93      /* conter toda a matriz
94      /************************************************/
95      printf("Alocando vetor de distancias\n");
96      vetorDistancia = malloc((colunas*colunas) * sizeof(long int));
97      //vetorCidades   = malloc((colunas - 2) * sizeof(struct cidades));
98      struct cidades vetorCidades[colunas - 2];
99      //Atualiza o n de colunas descontando a posicao 0
100     colunas = colunas - 1;
101
102     int k = 0;
103     rewind(distancias);
104     while (result != NULL)
105     {
106           result = fgets(linha, sizeof(linha), distancias);
```

```
107             token = strtok(linha, ";");
108             if (token == '\0' || token == '\n') {continue;}
109             aux = atoi(token);
110             if(aux) { *(vetorDistancia + k) = aux; }
111             else { *(vetorDistancia + k) = 999; }
112             k++;
113
114             while (token != NULL)
115             {
116                 token = strtok(NULL, ";");
117                 if (token == '\0' || token == '\n') {continue;}
118                 aux = atoi(token);
119                 if(aux) { *(vetorDistancia + k) = aux; }
120                 else { *(vetorDistancia + k) = 999; }
121                 k++;
122             }
123     }
124
125     /************************************************/
126     /* Aloca o ponteiro vetorCidades para conter as
127     /* informacoes sobre cidades, aliquota de ISS
128     /* e qual base supre sua demanda
129     /************************************************/
130     printf("Alocando vetor de cidades\n");
131     cidadesDistribuidoras = fopen("C:\\Thiago\\TEMP\\
            cidadesDistribuidoras_32.csv","r");
132     if (cidadesDistribuidoras == NULL )
133        { printf("P%d - Erro ao abrir cidadesDistribuidoras\n", rank); }
134
135     k = 0;
136     result = 1;
137     rewind(cidadesDistribuidoras);
138     while (result != NULL)
139     {
140             result = fgets(linha, sizeof(linha), cidadesDistribuidoras);
141             //1 - codigo da cidade
142             token = strtok(linha, ";");
143             if (token == '\0' || token == '\n') {continue;}
144             else { vetorCidades[k].cidadeCliente = atoi(token); }
145             //2 - codigo do estado
146             token = strtok(NULL, ";");
147             if (token == '\0' || token == '\n') {continue;}
148             else { strcpy(vetorCidades[k].UF, token); }
149             //3 - nome da cidade
150             token = strtok(NULL, ";");
151             if (token == '\0' || token == '\n') {continue;}
152             else { strcpy(vetorCidades[k].nomeCidade, token); }
```

```
153             //4 - Flag de ativacao
154             token = strtok(NULL, ";");
155             if (token == '\0' || token == '\n') {continue;}
156             else { vetorCidades[k].ativa = atoi(token); }
157             //5 - aliquota ISS
158             token = strtok(NULL, ";");
159             if (token == '\0' || token == '\n') {continue;}
160             else { vetorCidades[k].ISS = atoi(token); }
161             //6 - Custo fixo de implantacao
162             token = strtok(NULL, ";");
163             if (token == '\0' || token == '\n') {continue;}
164             else { vetorCidades[k].custoFixo = atoi(token); }
165             //7 - Cidade Fornecedora
166             token = strtok(NULL, ";");
167             if (token == '\0' || token == '\n') {continue;}
168             else { vetorCidades[k].cidadeFornecedora = atoi(token); }
169             //8 - Melhor Custo
170             token = strtok(NULL, ";");
171             if (token == '\0' || token == '\n') {continue;}
172             else { vetorCidades[k].melhorCusto = atof(token); }
173             //9 - Imposto
174             token = strtok(NULL, ";");
175             if (token == '\0' || token == '\n') {continue;}
176             else { vetorCidades[k].imposto = atof(token); }
177 
178             k++;
179     }
180     fclose(cidadesDistribuidoras);
181 
182     /*************************************************/
183     /* Determina quantas bases distribuidoras existem.
184     /* Determina quantas cidades sem bases existem.
185     /* - Cria vetor com bases ativas
186     /* - Cria vetor com cidades onde nao ha bases
187     /*************************************************/
188     int i;
189     int j;
190     //determina a quantidade de bases distintas
191     for (i = 0; i < k; i++)
192     {
193         if (vetorCidades[i].ativa == 1) { qtdBases++; }
194         else { qtdBasesInativas++; }
195     }
196     // aloca os vetores de bases e de cidades sem base
197     vetorBases = malloc((qtdBases) * sizeof(long int));
198     if (!vetorBases) { printf("P%d - Erro na alocacao do vetorBases\n",
            rank); }
```

```
199     vetorBasesInativas = malloc((qtdBasesInativas) * sizeof(long int));
200     if (!vetorBasesInativas) { printf("P%d - Erro na alocacao do
            vetorBasesInativas\n", rank); }
201     //guarda cidades com bases distintas no vetor de bases
202     j = 0;
203     for (i = 0; i < k; i++)
204     {
205         if (vetorCidades[i].ativa == 1)
206         {
207             *(vetorBases + j) = vetorCidades[i].cidadeCliente;
208             j++;
209         }
210     }
211     //guarda cidades sem bases estabelecidas em um vetor
212     j = 0;
213     for (i = 0; i < k; i++)
214     {
215         if (vetorCidades[i].ativa != 1)
216         {
217             *(vetorBasesInativas + j) = vetorCidades[i].cidadeCliente;
218             j++;
219         }
220     }
221
222     printf("Cidades Fornecedoras: %d\n", qtdBases);
223     i = 0;
224     while (i < qtdBases)
225     {
226           printf("%d\n", *(vetorBases + i));
227           i++;
228     }
229
230     printf("Cidades sem Bases Fornecedoras: %d\n", qtdBasesInativas);
231     i = 0;
232     while (i < qtdBasesInativas)
233     {
234           printf("%d\n", *(vetorBasesInativas + i));
235           i++;
236     }
237
238     /*************************************************/
239     /* Percorre matriz determinando a melhor opcao dentre
240     /* as cidades fornecedoras para cada cidade cliente
241     /* ----------------------------------------------
242     /* Faz o teste antes e depois de aplicar o imposto
243     /* de modo a descobrir as vantagens competitivas
244     /* relacionadas aos tributos aplicados pelos estados
```

```
245     /*************************************************/
246     j = 0;
247     int t = 0;
248     int temp = 0;
249     int aliquota = 0;
250     float sumCustoDistancia = 0;
251     float custo = 0;
252     float imposto = 0;
253     float sumCustoFixo = 0;
254     float sumImposto = 0;
255     float custoTotalEfetivo = 0;
256     int indiceOrigem = 0;
257     int indiceDestino = 0;
258     long int codigoOrigem  = 0;
259     long int codigoDestino = 0;
260     long int origemDistancias, destinoDistancias;
261
262     printf("Abrindo arquivo de cidades\n");
263     configuracaoFinalCidades = fopen("C:\\Thiago\\TEMP\\
            configuracaoFinalCidades.txt","w+");
264     printf("\nDeterminando configuracao inicial cidade versus fornecedor\n"
            );
265     i = 0;
266     while (i < k)   //Le o vetor cidades, pega destino
267     {
268           codigoDestino = vetorCidades[i].cidadeCliente;
269           // determina o menor custo entre as cidades
270           // fornecedoras existentes.
271           int r = 0;
272           while (r < qtdBases)
273           {
274                 codigoOrigem = *(vetorBases + r);
275                 if (codigoOrigem == codigoDestino)
276                 {
277                    aliquota = vetorCidades[i].ISS;
278                 }
279                 else
280                 {
281                    //encontra indice da origem e destino na matrizICMS
282                    j = 0;
283                    while (matrizICMS[j][0] != (*(vetorBases + r)/100000))
                          { j++; }
284                    indiceOrigem = j;
285                    j = 0;
286                    while (matrizICMS[0][j] != (codigoDestino/100000)) { j
                          ++; }
287                    indiceDestino = j;
```

```
288                     aliquota = matrizICMS[indiceOrigem][indiceDestino];
289                 }
290
291                 // varre a matriz de distancias, aplicando a aliquota,
                        fazendo
292                 // o somatorio do custo entre todos os pares Fornecedor Vs.
                         Cliente
293                 for (j = 1; j < colunas; j++)
294                 {
295                     destinoDistancias = (long int) *(vetorDistancia + j);
296                     if (destinoDistancias == codigoDestino)
297                     {
298                         for (t = colunas; t < (colunas*colunas); t = (t +
                                colunas))
299                         {
300                             origemDistancias = (long int) *(vetorDistancia +
                                    t);
301                             if (origemDistancias == codigoOrigem)
302                             {
303                                 temp = (t + j);
304                                 custo = *(vetorDistancia + temp);
305                                 imposto = custo * aliquota / 100;
306
307                                 // Se este fornecedor possui um custo menor,
                                        atualiza o vetor
308                                 if ((custo + imposto) < vetorCidades[i].
                                        melhorCusto)
309                                 {
310                                     vetorCidades[i].melhorCusto = (custo +
                                            imposto);
311                                     vetorCidades[i].cidadeFornecedora =
                                            codigoOrigem;
312                                     vetorCidades[i].imposto = imposto;
313                                     break;
314                                 }
315                             }
316                         }
317                     }
318                 }
319                 r++;
320             }
321         i++;
322     }
323
324     i = 0;
325
326     // Soma custo total e grava configuracao inicial
```

```
327     while (i < k)
328     {
329           // Se a cidade tem base distribuidora, acumula o respectivo
330           // custo fixo de implantacao para ser somado na equacao do SUFLP
331           if (vetorCidades[i].ativa == 1)
332           {
333              sumCustoFixo = sumCustoFixo + (float) vetorCidades[i].
                     custoFixo;
334           }
335
336           sumImposto = sumImposto + vetorCidades[i].imposto;
337           custoTotalEfetivo = custoTotalEfetivo + (sumCustoFixo +
                 vetorCidades[i].melhorCusto);
338           fprintf(configuracaoFinalCidades,
339              "Cliente: %d - Melhor Fornecedor: %d - Melhor Custo: %4.2f -
                     Imposto: %3.2f\n",
340                                                 vetorCidades[i].cidadeCliente,
341                                                 vetorCidades[i].
                                                        cidadeFornecedora,
342                                                 vetorCidades[i].melhorCusto,
343                                                 vetorCidades[i].imposto);
344           i++;
345     }
346
347     /************************************************/
348     /* Apos a configuracao inicial com a melhor cidade
349     /* fornecedora, vamos testar o custo total efetivo
350     /* em cada cenario, variando uma cidade fornecedora
351     /* de cada vez, mantendo as demais fixas.
352     /************************************************/
353     int r, p;
354     int cont = 0;
355     long int baseTemp;
356     float melhorCusto = custoTotalEfetivo;
357     float melhorImposto = sumImposto;
358
359     printf("Custo total efetivo inicial: %.2f - Imposto: %.2f\n",
           custoTotalEfetivo, sumImposto);
360     fprintf(configuracaoFinalCidades,
361        "Custo total efetivo inicial: %.2f - Imposto: %.2f\n",
               custoTotalEfetivo, sumImposto);
362     printf("\nIniciando testes com variacao dos fornecedores:\n");
363
364     r = 0;
365     temp = 0;
366     float custoGlobal = 0;
367     while (r < qtdBases) //Para cada cidade fornecedora diferente
```

```
368     {
369             p = 0;
370             while (p < qtdBasesInativas) // para cada base inativa
371             {
372                     // guarda a base original para recolocar no vetor de
                            cidades
373                     if (p == 0)
374                     {
375                        baseTemp = *(vetorBases + r);
376                     }
377
378                     // substitui uma cidade fornecedora de cada vez, para todos
379                     // as cidades que estavam na lista de "inativas"
380                     i = 0;
381                     while (i < k)
382                     {
383                             if (vetorCidades[i].cidadeFornecedora == *(vetorBases
                                    + r))
384                             {
385                                vetorCidades[i].cidadeFornecedora = *(
                                       vetorBasesInativas + p);
386                             }
387                             i++;
388                     }
389
390                     *(vetorBases + r) = *(vetorBasesInativas + p);
391
392                     // Agora testa o SUFLP com esta configuracao
393                     // ********** INICIO SUFLP ***************
394                     custo = 0;
395                     imposto = 0;
396                     sumCustoDistancia = 0;
397                     sumCustoFixo = 0;
398                     sumImposto = 0;
399                     custoTotalEfetivo = 0;
400
401                     i = rank;
402                     while (i < k)   //Le vetorCidades, pega origem e destino
403                     {
404                             codigoDestino = vetorCidades[i].cidadeCliente;
405                             codigoOrigem = vetorCidades[i].cidadeFornecedora;
406
407                             if (codigoDestino == codigoOrigem)
408                             {
409                                aliquota = vetorCidades[i].ISS;
410                             }
411                             else //encontra indice da origem e destino na
```

```
                matrizICMS
412             {
413                 j = 0;
414                 while (matrizICMS[j][0] != (codigoOrigem / 100000)
                        ) { j++; }
415                 indiceOrigem = j;
416                 j = 0;
417                 while (matrizICMS[0][j] != (codigoDestino /
                        100000)) { j++; }
418                 indiceDestino = j;
419                 aliquota = matrizICMS[indiceOrigem][indiceDestino
                        ];
420             }
421
422             // Se a cidade tem base distribuidora, acumula o
                    respectivo
423             // custo fixo de implantacao para ser somado na
                    equacao do SUFLP
424             if (vetorCidades[i].ativa == 1)
425             {
426                 sumCustoFixo = sumCustoFixo + (float) vetorCidades
                        [i].custoFixo;
427             }
428
429             // varre a matriz de distancias, aplicando a aliquota
                    , fazendo
430             // o somatorio do custo entre todos os pares
                    Fornecedor Vs. Cliente
431             for (j = 1; j < colunas; j++)
432             {
433                 destinoDistancias = (long int) *(vetorDistancia +
                        j);
434                 if (destinoDistancias == codigoDestino)
435                 {
436                     for (t = colunas; t < (colunas*colunas); t = (
                            t + colunas))
437                     {
438                         origemDistancias = (long int) *(
                                vetorDistancia + t);
439                         if (origemDistancias == codigoOrigem)
440                         {
441                             temp = (t + j);
442                             custo = *(vetorDistancia + temp);
443                             imposto = custo * aliquota / 100;
444                             sumImposto = sumImposto + imposto;
445                             sumCustoDistancia = sumCustoDistancia +
                                    (custo + imposto);
```

```
446                                 vetorCidades[i].melhorCusto = (custo +
                                        imposto);
447                                 vetorCidades[i].imposto = imposto;
448
449                                 break;
450                             }
451                         }
452                     }
453                 }
454
455                 // Reducao: Se somatorio ja ultrapassou o melhor,
456                 // descarta este fornecedor e passa ao proximo sem
457                 // concluir o loop.
458                 if (sumCustoDistancia > melhorCusto)
459                 {
460                     printf("Break: Custo ultrapassado na iteracao %d
                            da rodada #%d\n", i, (cont + 1));
461                     fprintf(configuracaoFinalCidades,
462                     "Break: Custo ultrapassado na iteracao %d da
                            rodada #%d\n", i, (cont + 1));
463                     break;
464                 }
465
466                 i = i + numtasks;
467             }
468             cont++;
469
470             MPI_Allreduce(&sumCustoDistancia, &custoGlobal, 1,
                    MPI_FLOAT, MPI_SUM, MPI_COMM_WORLD);
471
472             //Grava a configuracao testada nesta iteracao e seu
                    respectivo custo
473             printf("P%d - iteracao %4d - custo global = %.2f\n", rank,
                    cont, custoGlobal);
474             printf("\nResultados da rodada # %d\n", cont);
475             fprintf(configuracaoFinalCidades, "\nResultados da rodada #
                    %d\n", cont);
476
477             custoTotalEfetivo = sumCustoDistancia + sumCustoFixo;
478             printf(">>> Custo Total Efetivo: %.2f - Impostos: %.2f",
                    custoTotalEfetivo, sumImposto);
479             fprintf(configuracaoFinalCidades,
480                 ">>> Custo Total Efetivo: %.2f - Impostos: %.2f",
                        custoTotalEfetivo, sumImposto);
481
482             if (custoGlobal < melhorCusto)
483             {
```

```
484                     printf("P%d - Melhor custo encontrado: %.2f - Iteracao
                            #%4d \n", rank, custoGlobal, cont);
485                     fprintf(configuracaoFinalCidades, " - Melhor custo
                            encontrado! #%d \n", cont);
486                     melhorCusto = custoGlobal;
487                 }
488                 else
489                 {
490                     printf("\n");
491                     fprintf(configuracaoFinalCidades, "\n");
492                 }
493                 // ************ FIM SUFLP *****************
494
495                 // Recoloca a base fornecedora original
496                 if ((p + 1) == qtdBasesInativas)
497                 {
498                     *(vetorBases + r) = baseTemp;
499                     i = 0;
500                     while (i < k)
501                     {
502                         if (vetorCidades[i].cidadeFornecedora == *(
                                vetorBasesInativas + p))
503                         {
504                             vetorCidades[i].cidadeFornecedora = *(vetorBases +
                                    r);
505                         }
506                         i++;
507                     }
508                 }
509
510                 p++;
511             }
512             r++;
513         }
514
515         printf("\nResultado custoGlobal: %.2f\n", melhorCusto);
516
517         /*************************************************/
518         /* comandos para finalizacao do programa,
519         /* registrar tempo, gravar LOG e fechar arquivos
520         /*************************************************/
521         fim = time(NULL);
522         fim_log = time(NULL);
523         printf("\n\nfinalizando simulacao\n");
524         printf("Tempo de execucao: %14.6f segundos.\n", difftime(inicio,fim) *
                (-1));
525         printf("\nGravando LOG\n");
```

```
526     logExecucao = fopen("C:\\Thiago\\TEMP\\log.txt","a+");
527     fseek(logExecucao, 0, SEEK_END);
528     fprintf(logExecucao,
529        "Tempo de execucao: %14.6f segundos - %s", difftime(inicio,fim) *
               (-1), ctime(&fim_log));
530     fclose(logExecucao);
531     fclose(distancias);
532     fclose(configuracaoFinalCidades);
533     printf("\nLOG regitrado com sucesso.\n");
534     MPI_Finalize();
535     return (0);
536 }
```