COPPE/UFRJ

APLICAÇÕES DE ALGORITMOS PARALELOS E HÍBRIDOS PARA O PROBLEMA DE ÁRVORE DE STEINER EUCLIDIANA NO R^n

Marcelo Lisboa Rocha

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia Elétrica.

Orientador: Amit Bhaya

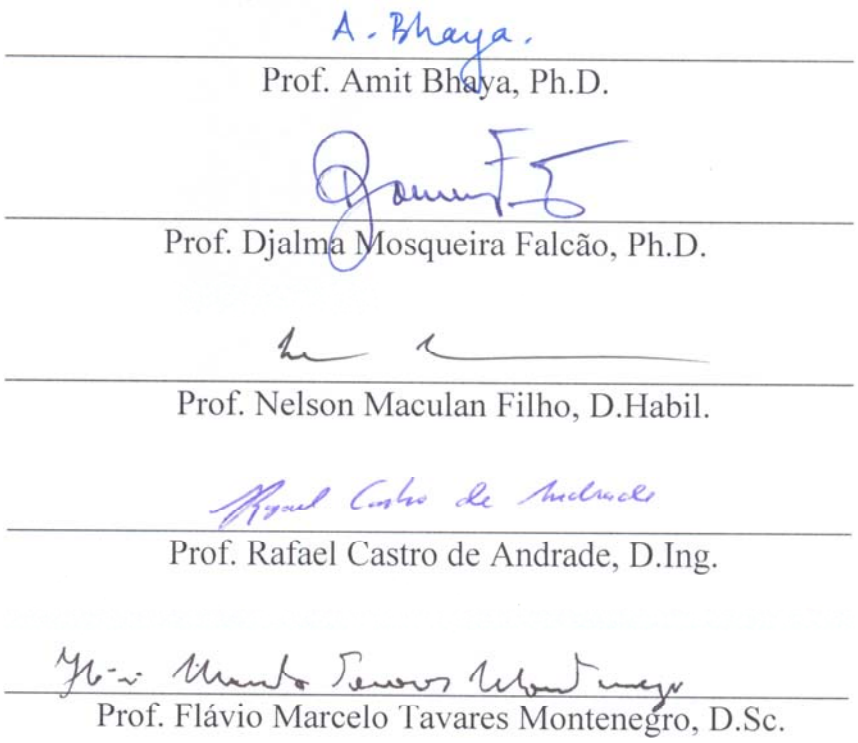
Rio de Janeiro
Outubro de 2008

APLICAÇÕES DE ALGORITMOS PARALELOS E HÍBRIDOS PARA O PROBLEMA DE ÁRVORE DE STEINER EUCLIDIANA NO R^n

Marcelo Lisboa Rocha

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA ELÉTRICA

Aprovada por:



RIO DE JANEIRO, RJ - BRASIL

OUTUBRO DE 2008

Rocha, Marcelo Lisboa

Aplicações de Algoritmos Paralelos e Híbridos para o Problema de Árvore de Steiner Euclidiana no R^n / Marcelo Lisboa Rocha. – Rio de Janeiro: UFRJ/COPPE, 2008.

XIV, 99 p.: il.; 29,7 cm.

Orientador: Amit Bhaya

Tese (doutorado) – UFRJ/ COPPE/ Programa de Engenharia Elétrica, 2008.

Referencias Bibliográficas: p. 90-99.

1. Problema de Árvore de Steiner Euclidiano. 2. Métodos Híbridos. 3. Otimização Combinatória. 4. Computação Paralela. 5. Heurísticas e Metaheurísticas I. Bhaya, Amit. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Titulo.

Agradecimentos

Primeiramente e acima de tudo, agradeço a DEUS por todos os livramentos, bênçãos, oportunidades e o refrigério dado nos momentos de dificuldades no desenrolar deste trabalho, que não foram poucos. E também agradeço por mais esta vitória pela honra e gloria de seu nome Ó SENHOR!!!

Agradeço profundamente ao meu orientador Amit Bhaya, pela orientação acadêmica, pelos ensinamentos que não se limitaram à sala de aula, pela sua grande amizade e enorme paciência que creio eu só um pai tem por seu filho. Sei que sem sua enorme compaixão, a realização deste trabalho não seria possível.

Á minha querida esposa Cristina, pelos momentos de ausência, que não foram poucos. Agradeço também pela paciência e companheirismo nesta jornada cheia de percalços.

Aos meus filhos Marcelo e Matheus por terem compreendido a ausência do pai em diversos momentos e pelo carinho dispensado. Vocês são uns dos grandes motivos da perseverança nesta batalha.

Á minhas amadas mãe Cristina e vó Edna, pelos incansáveis momentos de oração e torcida pelo meu sucesso. Sei que sem vocês eu nunca teria chegado até onde cheguei.

Ao meu querido e amado vô Fritz, sei que esteja onde estiver, sempre estará olhando por mim e desejando o meu sucesso.

Aos irmãos da Igreja Cristã Maranata pelas orações e interseções junto ao SENHOR que com certeza muito me auxiliaram e me deram refrigério nos momentos de dificuldades.

Finalmente, agradeço a todos que direta ou indiretamente me auxiliaram, torceram e em especial oraram por mim para que mais esta vitória chegasse.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

APLICAÇÕES DE ALGORITMOS PARALELOS E HÍBRIDOS PARA O PROBLEMA DE ÁRVORE DE STEINER NO R^n

Marcelo Lisboa Rocha

Outubro/2008

Orientador: Amit Bhaya

Programa: Engenharia Elétrica

Neste trabalho será apresentada uma aplicação proposta de algoritmos paralelos e híbridos para problemas de otimização de grande porte, mais especificamente ao Problema de Árvore de Steiner Euclidiano (*PASE*) no R^n. Como método híbrido, foi proposta uma heurística GRASP mais o procedimento de *Path-relinking*, com várias implementações paralelas do mesmo. Também foi realizada uma implementação paralela de um time assíncrono para o problema em questão. Os métodos propostos neste trabalho ao problema em questão possuem desempenho superior ao dos melhores métodos existentes na literatura, conforme mostram os testes e resultados computacionais obtidos.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

APPLICATION OF PARALLEL AND HYBRID METHODS FOR EUCLIDEAN STEINER TREE PROBLEM IN R^n

Marcelo Lisboa Rocha

October/2008

Advisor: Amit Bhaya

Department: Eletrical Engineering

This work presents a proposed application of parallel and hybrid algorithms for a large scale optimization problem, more specificaly to the Euclidean Steiner Tree Problem (*ESTP*) in R^n. A Greedy Randomized Adaptive Serach Procdure (GRASP) plus a *Path-relinking* procedure is proposed as a hybrid method, with several parallel implementations of it. Also was made a parallel implementation of an Asynchronous Team for the problem in consideration. Methods proposed in this work for the ETSP problem under consideration have better performance than the best methods in the literature, as shown by the tests made and the computational results obtained.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos e Abreviaturas

P	Número de pontos fixos
N	Número de dimensões do espaço
R^n	Espaço n-dimensional
R^3	Espaço tri-dimensional
NP	Não-determinístico polinomial
x^i	Coordenadas no espaço do ponto i
K	Número de pontos de Steiner
$f(P)$	Número de topologias cheias de Steiner
a_i	Valor na posição i do vetor topologia de Smith
S	Conjunto de pontos de Steiner
FST	Full Steiner Tree – Árvore de Steiner Cheia
AGs	Algoritmos Genéticos
μO	Algoritmo de Otimização Microcanônica
ρ	Razão de Steiner
LRC	Lista Restrita de Candidatos
α	Parâmetro que define o tamanho da LRC ($0 \leq \alpha \leq 1$)
NMSE	Indica o número de melhores soluções encontradas de uma técnica em relação a outra(s)
GRASP	Greedy Randomized Adaptive Search Procedure – Procedimento de Busca Gulosa Adaptativa Aleatorizada
GRASP-S	GRASP básico seqüencial
GRASP-PR	GRASP com *Path-relinking* sequencial
GRASP-PRP	GRASP com *Path-relinking* paralelo
GRASP-PRP1	GRASP com *Path-relining paralelo* para máquina de memória compartilhada
GRASP-PRP2	GRASP com *Path-relining paralelo* para máquina de

	memória distribuída
GRASP-PRP3	GRASP com *Path-relining paralelo* para máquina híbrida
A-TEAM	Time Assíncrono
p	Número de processadores
MPI	Biblioteca de Passagem de Mensagens – *Message Passing Interface*
OpenMP	API para programação paralela em máquinas multiprocessadas – *Open Multiprocessor Programming*
PASE	Problema da Árvore de Steiner Euclidiano
POC	Problema de Otimização Combinatória

Capítulo 1

Introdução

Problemas de otimização combinatória têm uma vasta gama de aplicações, tanto práticas quanto teóricas [33], [51], incluindo localização de facilidades, planejamento de produção, e escalonamento. Estes problemas aparecem em situações do mundo real, tais como: telecomunicações (operação, localização, projeto), energia (gás natural, petróleo, eletricidade), transportes (aéreo, ferroviário, rodoviário, naval) [33], [51], [67]. Conforme os tamanhos dos problemas crescem, o tempo necessário para achar uma solução também cresce exponencialmente.

O problema da dieta foi uma das primeiras aplicações da otimização a ser resolvida numericamente. Em 1947, uma versão do problema contendo 9 restrições e 77 variáveis, que era considerado um problema de grande porte para a época, foi resolvida usando calculadoras de mesa em 120 dias de trabalho [21]. Hoje, os sistemas computacionais são capazes de manipular problemas com milhares de variáveis em poucos segundos. De fato, o porte dos problemas a serem resolvidos por um determinado sistema depende principalmente da disponibilidade de memória da máquina onde o sistema está instalado. No entanto, manipular as informações associadas a um modelo de otimização de grande porte, nos padrões de hoje, não é uma tarefa trivial [73]. E isso diz respeito ao armazenamento e manipulação dos dados do problema, bem como à geração e documentação das variáveis e das restrições do modelo associado, além de outros fatores como fornecer e obter informações dos sistemas computacionais de solução.

Neste trabalho, será abordado o Problema da Árvore de Steiner Euclidiano no R^n. Dados P pontos em R^n com métrica euclideana, o problema da árvore de Steiner consiste em encontrar uma árvore de custo mínimo que conecte estes pontos, utilizando ou não pontos adicionais introduzidos para resolver o problema. Estes pontos adicionais introduzidos no decorrer da solução denominam-se pontos de Steiner. A principal motivação para abordar este problema é que existem diversas heurísticas e métodos

exatos para o caso bidimensional (n=2). Já para dimensões maiores ($n \geq 3$), existem poucas heurísticas com desempenho razoável e dois métodos exatos que se tornam computacionalmente inviáveis para $P \geq 18$.

Este problema, como muitos outros, que são classificados como sendo problemas de otimização de grande porte, pertence a uma classe de problemas que freqüentemente levam a formulações envolvendo milhares ou milhões de variáveis continuas e/ou discretas como também milhares ou milhões de restrições [34]. Nestes casos, freqüentemente, os métodos padrão de solução não apresentam bom desempenho, exigindo elevados tempos computacionais. Este fato é agravado ainda mais quando o problema de otimização possui alto grau de dificuldade em sua solução (complexidade computacional). Na maioria dos casos, estes problemas são NP-difíceis (muito difíceis de serem resolvidos computacionalmente), ou seja, são problemas para os quais não existem algoritmos que os resolvam de forma exata em tempo polinomial. O leitor pode obter maiores conhecimentos sobre problemas NP-difíceis em HOPCROFT, ULLMAN e MOTWANI [36].

Para resolução deste e de muitos outros problemas computacionais encontrados na literatura, além da busca incansável por melhores técnicas e algoritmos, tem sido buscada a integração de diversas subáreas da computação. Um esforço nesse sentido é o emprego, cada vez mais freqüente, de computação paralela para "acelerar" a obtenção de soluções aproximadas ou ótimas em problemas de otimização combinatória [22], [73], [24], [28].

Segundo FERREIRA e PARDALOS [24], a implementação paralela de técnicas que solucionam problemas de otimização combinatória dá a elas a oportunidade de reduzir os seus tempos computacionais e possivelmente melhorar a qualidade das soluções obtidas pela realização de uma busca paralela em múltiplas áreas do espaço de busca, provavelmente utilizando estratégias e/ou parâmetros diferentes em cada uma das áreas.

Outro conceito, ainda mais recente, a ser utilizado neste trabalho refere-se à hibridização de algoritmos. Neste caso, a hibridização refere-se à "mistura" dos conceitos de dois ou mais algoritmos com objetivo de obter um sistema com

desempenho melhor que os obtidos pelos algoritmos individualmente, explorando as vantagens de cada método, ou seja, os métodos híbridos buscam se beneficiar da sinergia [61].

Assim sendo, a paralelização de algoritmos híbridos vem ao encontro da necessidade dos problemas de otimização de grande porte, que é a existência de técnicas que propiciem a obtenção de boas soluções com baixo tempo computacional [61], [22].

Desta forma, este trabalho tem como motivação principal desenvolver e implementar algoritmos paralelos e híbridos que encontrem a solução ótima ou próxima da ótima eficientemente para um problema de otimização combinatória de grande porte que seja NP-difícil, abordando o problema de Árvore de Steiner Euclidiano no R^n e comparar os resultados obtidos com os disponíveis na literatura.

1.1 Revisão da Literatura

O problema de árvore de Steiner é um problema de otimização que já vem sendo estudado por muito tempo [15], nas suas mais variadas vertentes, tais como: árvore de Steiner em grafos, árvore de Steiner retilínea, árvore de Steiner Euclidiana no plano e mais recentemente a árvore de Steiner Euclidiana no R^n entre outros.

Este grande interesse pelo problema de árvore de Steiner se dá pelo fato que diversos problemas tanto práticos quanto teóricos podem ser modelados como uma árvore de Steiner. Maiores detalhes sobre as aplicações do PASE serão apresentados na seção 2.5 deste trabalho. Neste ínterim, um foco maior tem sido dado a metodologias que eficientemente busquem por soluções para o problema de árvore de Steiner Euclidiana (PASE) no R^n, mais especificamente quando $n = 3$ (caso tridimensional).

O problema de árvore de Steiner no R^n, quando $n = 2$, possui diversos métodos exatos e heurísticos eficientes que o solucionam. Contudo, quando $n \geq 3$, existem diversas heurísticas [73], [44], [9], [45], [62], [39], [48] que encontram soluções razoáveis para instâncias de tamanho médio e dois métodos exatos [75], [23] aplicáveis apenas a instâncias pequenas. Assim sendo, nesta seção pretende-se apresentar os

principais trabalhos da literatura para o PASE considerando apena os casos onde $n \geq 3$. Estes trabalhos serão divididos basicamente em duas classes: os que proporcionam soluções exatas e os que proporcionam soluções heurísticas.

1.1.1 Algoritmos Exatos

Atualmente na literatura, só existem dois algoritmos exatos aplicáveis ao PASE para $n \geq 3$. O primeiro é o algoritmo de Smith [75] a ser detalhado na seção 2.3 deste trabalho. O segundo algoritmo exato deve-se à FAMPA e ANSTREICHER [23], denominado Smith+, que é uma melhora do algoritmo de SMITH [75].

Em [23], as melhoras feitas em Smith+ são: obtenção de melhores *lower-bounds*; utilização da estratégia *strong-branching* no método *branch-and-bound* e a realização de um método de *bound* mais eficiente, permitindo a eliminação de nós filhos na árvore de enumeração, reduzindo assim o esforço computacional. Apesar destas melhoras desenvolvidas, o algoritmo Smith+ ainda é ineficiente para casos práticos (onde o número de pontos fixos é elevado). Isto foi comprovado nos testes computacionais realizados em [23], que foram realizados sobre conjuntos de instâncias com P=10 (pequena dimensionalidade) e $2 \leq n \leq 5$ (com 5 instâncias para cada valor de n). Para $n = 3$, apresentou tempo médio em segundos de CPU igual a 753.5 em uma máquina com processador Pentium IV 1.8 GHz. Para valores de n maiores, os tempos computacionais ficaram ainda piores.

1.1.2 Algoritmos Heurísticos

Dentre os diversos algoritmos heurísticos existentes na literatura, historicamente tem-se os propostos por RAVADA e SHERMAN [62], MACGREGOR SMITH et al. [39] e PEREIRA et al. [58]. Estes não serão abordados em maior profundidade, por não disponibilizarem nem as instâncias nem tampouco os códigos para testes comparativos. Maiores detalhes a respeito desses trabalhos podem ser vistos em [47].

Nesta seção, os algoritmos heurísticos a serem abordados em maior profundidade foram utilizados em [44] e considerados como objeto de comparação no

capítulo referente aos resultados computacionais. Os mesmos são: Soap Film [48], algoritmos genéticos [45] e otimização microcanônica [44].

A heurística Soap Film foi proposta por CHAPEAU-BLONDEAU et al. [9] e estendida para o caso onde $n \geq 3$ em [48], de um esquema de relaxação desenvolvido no plano e baseado em um modelo físico para o PASE. Tal modelo simula a evolução dinâmica de uma película de sabão sujeita a forças de tensão superficial. Assim, partindo de uma árvore inicial com topologia cheia, tem-se um primeiro estágio na composição das forças de tensão superficial. Sujeita a estas forças, a película relaxa para uma nova configuração, possivelmente com topologia diferente, e o processo segue iterativamente até que se encontre uma configuração de equilíbrio, que corresponderá a uma árvore de Steiner localmente mínima. Apesar de ser uma heurística bastante rápida, de acordo com os testes computacionais executados em [44], a mesma não conseguiu encontrar nenhuma solução ótima para 4 conjuntos de instâncias de pequena dimensão ($8 \leq P \leq 11$).

MONTENEGRO e MACULAN propuseram em [45] uma heurística baseada em algoritmos genéticos (AGs) para o PASE. Neste caso, foi utilizado um algoritmo genético simples, onde cada cromossomo é um vetor topologia de Smith (conforme apresentado na seção 2.2) e com as seguintes características principais:

- a função objetivo é a razão de Steiner ρ, obtida de cada cromossomo, pela aplicação do processo de minimização de Smith;
- a seleção para reprodução é dada por torneio de 3 cromossomos;
- a população inicial possui cromossomos criados pela inserção de pontos em uma Árvore Geradora Mínima (AGM);
- o operador de reprodução utilizado foi o *crossover* de dois pontos;
- cada gene i de um cromossomo tem uma mesma probabilidade de mutação; e quando ocorrer, será sorteado um valor inteiro escolhido no intervalo $1 \leq a_i \leq 2i+1$;
- é utilizada a estratégia elitista;
- o AG finaliza após um determinado número de iterações (gerações) sem melhora da melhor solução encontrada.

Dos testes computacionais apresentados pelo AG em [44], [47], verifica-se que apresenta resultados razoáveis quando comparado aos resultados de uma heurística Soap Film e outra heurística de Busca Tabu implementados em [48] e ao algoritmo de otimização microcanônica de [86] que será apresentado a seguir.

O algoritmo de otimização microcanônica (μO) [86] proposta por MONTENEGRO et al. em [44] para o PASE é um método baseado na física-estatística para resolver problemas gerais de otimização [38]. Neste caso, uma solução possível é um vetor topologia de Smith, conforme apresentado na seção 2.2. O algoritmo começa de uma solução inicial arbitrária (com energia E – custo da solução), e aplica alternadamente dois procedimentos principais, chamados *inicialização* e *sampling*. A fase de *inicialização* busca por uma boa solução inicial, fazendo uso para isto de um processo de busca local. Já na fase de *sampling*, busca-se escapar do mínimo local gerado na fase anterior fazendo mudanças que não alterem demasiadamente a energia E da solução por um determinado número de iterações. Após, o algoritmo inicia a partir de uma nova solução corrente e fica alternando entre as fases de *inicialização* e *sampling* até que algum critério de parada seja ativado.

De acordo com os testes computacionais realizados e os resultados obtidos em [44] e [47], verifica-se que o algoritmo de otimização microcanônica (μO) é o que propicia, entre os métodos propostos anteriormente na literatura, a obtenção de melhores soluções com melhor desempenho, principalmente para as instâncias de maior dimensionalidade. Assim sendo, esta vai ser a heurística da literatura utilizada como padrão de comparação com os métodos propostos neste trabalho.

1.2 Estrutura do Trabalho

Esta tese está estruturada como se segue. No capítulo 2 é apresentada uma introdução ao problema de árvore de Steiner Euclidiano no R^n, aplicações e a formulação matemática adotada. O capítulo 3 apresentada a metodologia adotada, os algoritmos desenvolvidos (híbridos ou não) tanto paralelos quanto seqüenciais. Já no capítulo 4, são mostrados os testes computacionais, resultados e tempo de execução das

técnicas propostas executando seqüencialmente e em paralelo, como também uma comparação com outros trabalhos da literatura e entre as várias versões dos algoritmos propostos. Finalmente, no capítulo 5 são apresentadas as conclusões gerais e as contribuições baseadas nas implementações realizadas e dos resultados computacionais obtidos na aplicação das abordagens propostas sobre o problema em questão. Posteriormente, neste mesmo capítulo são descritas as propostas de continuidade do trabalho.

Capítulo 2

O Problema de Árvore de Steiner Euclidiano

Neste capítulo serão apresentados diversos aspectos a respeito do problema de árvore de Steiner euclidiano, tais como: descrição do problema; da possibilidade de descrever uma solução possível do problema através de um vetor descritor de topologias cheias; de como é possível, dado um vetor descritor de topologias cheias, encontrar a posição dos respectivos pontos de Steiner através do passo de minimização do algoritmo de Smith; apresentação da formulação matemática do problema; e aplicações do problema em questão.

2.1 Descrição do Problema

O Problema da Árvore de Steiner Euclidiana (PASE) no R^n consta da existência de P pontos (obrigatórios) no R^n com métrica euclidiana e encontrar uma árvore geradora mínima (AGM) que conecte estes pontos utilizando ou não pontos extras (pontos de Steiner), como pode ser observado na Figura 1. Assim, um vértice ou ponto de Steiner é um ponto que quando adicionado à árvore reduz o custo total desta. A seguir, serão apresentadas as principais características do PASE.

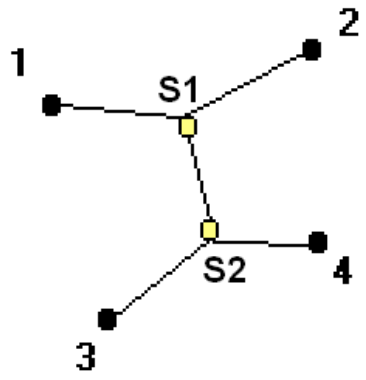


Figura 1: Árvore de Steiner com 4 pontos obrigatórios (P=4) e dois pontos de Steiner (S1 e S2).

Uma solução do PASE, chamada de Árvore de Steiner Mínima (ASM), é a árvore que possui um ou mais pontos de Steiner e custo mínimo e deve apresentar as

seguintes propriedades [75]:

- Dados P pontos obrigatórios $x^i \in R^n$, $i = 1, 2, 3, ..., P$, o número máximo de pontos de Steiner (K) é P-2.
- Um ponto de Steiner deve ter valência (grau) igual a 3.
- As arestas emanando de um ponto de Steiner estão em um mesmo plano e têm ângulo de 120° entre si.
- Os P pontos fixos devem ser nós folha, ou seja, possuir grau igual a 1.

A Figura 2 apresenta exemplos de AGMs e ASMs para P=3 e P=4 no R^2, onde L(AGM) e L(ASM) correspondem respectivamente ao comprimento da AGM e ao comprimento da ASM. No exemplo com P=3, as coordenadas dos pontos fixos são: p1={0.5, 1}, p2={0, 0} e p3={1, 0}. Já no exemplo com P=4, as coordenadas dos pontos fixos são: p1={1, 0}, p2={0, 0}, p3={1, 0} e p4={1, 1}.

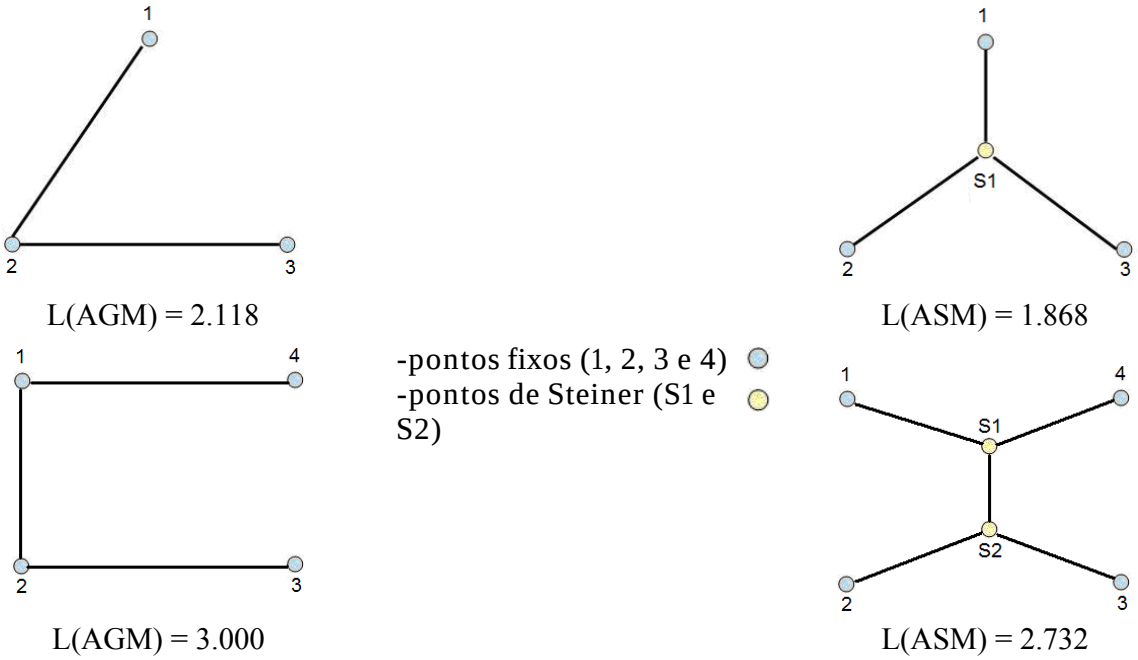


Figura 2: Exemplos de AGMs e ASMs com 3 e 4 pontos obrigatórios (P=3 e P=4) e as ASMs fazendo uso respectivamente de um e dois pontos de Steiner (S1 e S2).

A ASM é um grafo e este grafo que representa uma topologia de Steiner é chamado de topologia de Steiner. A natureza combinatorial do PASE permite o cálculo do número de topologias diferentes com K pontos de Steiner como $f(P,k) = \frac{(P + K - 2)!}{(k!2^K)}$.

Quando K=P-2, tem-se uma topologia de Steiner cheia (*Full Steiner Tree – FST*) e o número de topologias diferentes é $f(P) = \frac{(2P-4)!}{2^{P-2}(P-2)!}$ Considerando-se, por exemplo, P=10, o número total de topologias cheias será de $f(P) = 2.027.025$. Este é o número de topologias cheias a serem checadas em um método de força bruta de encontrar o grafo de comprimento mínimo. Em GAREY, GRAHAM e JOHNSON [29] e GAREY e JOHNSON [30] o PASE é classificado como NP-Difícil, fundamentalmente devido a este aspecto combinatório, ou seja, até o momento não é possível encontrar algoritmos eficientes (de tempo polinomial) para este problema.

Na seção 2.2 a seguir, serão dados maiores detalhes a respeito da topologia cheia para representação de Árvore de Steiner Euclidiana e sua importância neste trabalho.

2.2 Descrevendo Topologias Cheias

Existe uma correspondência 1-1 (um-para-um) entre topologias de Steiner cheias com $P \geq 3$ pontos dados e $(P - 3)$-vetores, aqui identificados como vetores a. Em cada vetor a, a sua i-ésima componente (posição) a_i corresponde a um valor inteiro no intervalo $1 \leq a_i \leq 2i + 1$. Aqui, a i-ésima entrada a_i do vetor topologia está relacionada à inserção do ponto de Steiner P+i+1 sobre a aresta a_i e a respectiva conecção com o ponto dado i+3. Isto envolve a introdução de duas novas arestas: a aresta 2i+2, conectando o novo ponto de Steiner (P+i+1) ao novo ponto dado (i+3), e a aresta 2i+3, conectando este ponto de Steiner (P+i+1) ao extremo (vértice) da aresta a_i que possuir o maior número.

A seguir, na Figura 3, é apresentado um exemplo da construção de uma árvore de Steiner cheia, levando em consideração o vetor topologia a = (2, 4) e considerando P = 5 e K = 3.

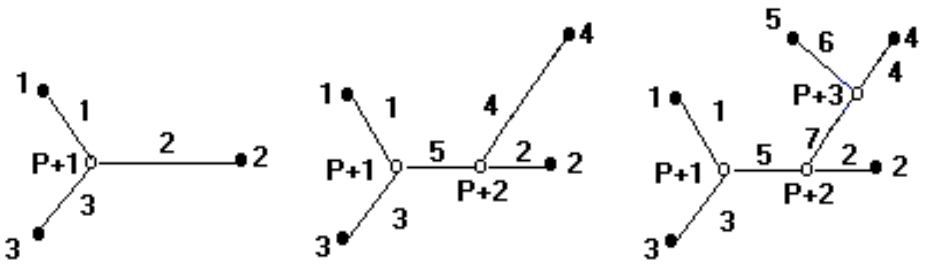


Figura 3: O vetor inicial nulo ( ) corresponde à topologia apresentada em (a); a topologia com conexão do ponto dado 4 na aresta 2 correspondente ao vetor (2) está apresentada em (b), e a topologia com o ponto dado 5 inserido na aresta 4 é apresentado em (c).

Quanto à questão topológica, pode-se considerar todas as árvores (conexas) de topologia não-cheia como topologias de Steiner cheias com um ou mais pontos de Steiner coincidindo com pontos dados. Desta forma, fazendo uso de métodos adequados de minimização, é suficiente focar em topologias cheias, quando buscando por soluções para o PASE. Um método de minimização eficiente é o proposto por SMITH [75] e que será abordado na seção 2.3 a seguir. O segundo método exato existente é o proposto por FAMPA e ANSTREICHER [23], denominado Smith+, que é uma melhora do algoritmo de SMITH [75]. Contudo o método de FAMPA e ANSTREICHER [28] não foi considerado neste trabalho pelos seguintes motivos: primeiramente por não estarem disponíveis tanto o código-fonte do método quanto as instâncias utilizadas no teste e, em segundo lugar, o mesmo só foi aplicado a instâncias pequenas ($P \leq 18$).

2.3 Algoritmo de Smith

Em [75], Smith propôs um método enumerativo para encontrar a solução exata para o PASE em dimensão $n \geq 3$, consistindo basicamente em enumerar todas as possíveis soluções para o problema e minimizá-las, selecionando dentre estas a melhor solução. A enumeração das soluções é realizada através dos vetores topologia, cada um representando uma possível topologia cheia, que são aplicados, em seguida, no contexto de um algoritmo *branch-and-bound*, o que evita a minimização de algumas soluções ruins. Sendo P o número de pontos dados, a estratégia geral deste algoritmo é:

1. Encontrar todas as *FST* de tamanho *P*.
2. Otimizar as coordenadas dos pontos de Steiner de todas as *FST* geradas no passo 1, de forma a encontrar a árvore de menor custo.

A primeira consideração a respeito do algoritmo de Smith é que ele só procura por topologias completas (FST). A razão para esta simplificação é que o método de otimização utilizado por Smith permite considerar as outras topologias como degenerações da topologia completa, onde alguns pontos de Steiner se sobrepõem aos pontos normais. Apesar disto, o número de *FST* ainda é exponencial em *n*.

No segundo passo mencionado acima, o de otimizar as coordenadas dos pontos de Steiner, o algoritmo de Smith procede de maneira iterativa, sendo que a regra de parada utiliza a propriedade de não ter nenhum ângulo entre arestas da árvore alvo (de Steiner) maior que 120 graus.

Smith afirma em seu trabalho que seu algoritmo só é capaz de resolver problemas com doze pontos, aproximadamente, mas em qualquer dimensão. Este fato inviabiliza a utilidade de seu algoritmo em casos práticos (instâncias com número grande de pontos). Apesar disto, o trabalho de SMITH [75] é importante pela possibilidade de utilização de forma eficiente do passo de minimização do método de Smith para encontrar as posições ótimas dos pontos de Steiner para um vetor topologia dado.

2.4 Formulação Matemática para o Problema da Árvore de Steiner Euclidiana no R^n

Em seu trabalho [75], SMITH propôs um esquema enumerativo para o PASE, contudo, não apresentou explicitamente qualquer modelo de programação matemática. Já em [40], XAVIER, MACULAN e MICHELON propuseram uma formulação matemática para o Problema de Árvore de Steiner Euclidiana (PASE) no R^n, onde a mesma é definida como um problema de programação inteira mista não-linear e não

convexa. Na seqüência, serão dados maiores detalhes a respeito das variáveis envolvidas e da formulação em si.

Dado um grafo $G=(V, E)$, sendo $P=\{1, 2, ..., p\text{-}1, p\}$ o conjunto de índices associados com os pontos dados $x^1, x^2, ..., x^{p-1}, x^p$, e $S=\{p+1, p+2, ..., 2p\text{-}3, 2p\text{-}2\}$ o conjunto de índices associados aos pontos de Steiner $x^{p+1}, x^{p+2}, ..., x^{2p-3}, x^{2p-2}$, tem-se que $V=P \cup S$. Aqui, denota-se por $[i, j]$ uma aresta de G, onde $i, j \in V$ e $i < j$. Também são considerados os conjuntos $E_1=\{[i, j] \mid i \in P, j \in S\}$ (conjunto de arestas que ligam um ponto dado qualquer a um ponto de Steiner qualquer) e $E_2=\{[i, j] \mid i \in S, j \in S\}$ (conjunto de arestas que ligam quaisquer dois pontos de Steiner), tal que $E = E_1 \cup E_2$.

Denotando a distância Euclidiana entre x^i e x^j por $| x^i - x^j | = \sqrt{\sum_{l=1}^{n} (x_l^i - x_l^j)^2}$ e definindo $y_{ij} \in \{0, 1\} \ \forall \ [i, j] \in E$, onde $y_{ij} = 1$ se a aresta $[i, j]$ está na solução da árvore de Steiner ótima e $y_{ij} = 0$, caso contrário, tem-se a seguir, na Figura 4, a formulação matemática para o PASE no R^n.

(1) $$\min \sum_{[i, j] \in E} | x^i - x^j | y_{ij}$$

s.a.

(2) $$x^i \in \Re^n, i = p + 1,...,2p - 2$$

(3) $$\sum_{j \in S} y_{ij} = 1, i \in P$$

(4) $$\sum_{i \in P} y_{ij} + \sum_{k < j, k \in S} y_{kj} + \sum_{k > j, k \in S} y_{jk} = 3, j \in S$$

(5) $$\sum_{k < j, k \in S} y_{kj} = 1, j \in S - \{p + 1\}$$

(6) $$y_{ij} \in \{0, 1\}, [i, j] \in E$$

Figura 4: Formulação Matemática para o problema de Árvore de Steiner Euclidiana no R^n.

Na seqüência, será comentada cada uma das restrições da formulação matemática do PASE no R^n apresentada na Figura 4. De acordo com a restrição (2), pode-se verificar que x^i é um ponto de Steiner no R^n. As restrições de (3)-(6)

representam alguma topologia cheia de Steiner para p pontos dados no R^n. O conjunto de restrições (3) é utilizado para fixar cada vértice em P com grau igual a 1, e o conjunto de restrições (4) é utilizado para fixar cada vértice de S com grau igual a 3. Já o conjunto de restrições (5) evita a formação de subciclos entre os vértices de Steiner (pertencentes ao conjunto S).

Da formulação matemática do PASE no R^n é fácil observar que cada solução viável $y_{ij,}$ considerando as restrições de (3) a (6) está associada a um subgrafo conexo de G para o qual cada vértice tem grau 1 (ponto dado/obrigatório) ou 3 (ponto de Steiner), e o número de y_{ij} iguais a 1 é $2p - 3$. Assim, cada solução de (3) a (6) define uma árvore geradora de G.

2.5 Aplicações Práticas do Problema da Árvore de Steiner Euclidiana

O problema da árvore de Steiner Euclidiana (PASE) pode servir de suporte teórico a diversos problemas práticos reais, ou seja, resolvendo-se de forma eficiente o problema em questão, conseqüentemente, o mesmo acontece com os problemas que podem ser modelados como o PASE. Algumas aplicações que podem ser modeladas como um PASE são: configuração molecular (bioquímica) [73], [52], projeto de redes (elétrica, minas, dutos, etc.) e inferência filogenética [46]. A seguir, cada uma destas aplicações será apresentada em maiores detalhes.

2.5.1 Aplicação do PASE na Configuração Molecular

MCGREGOR SMITH [76] apresentou várias propriedades que foram generalizadas do espaço euclidiano bidimensional para o tridimensional, e ainda metodologias algorítmicas que, devido à novidade e potencialidade do problema de Steiner, podem ser utilizadas em diversas áreas de aplicações. Uma das aplicações mais interessantes é aquela relacionada ao Problema de Configuração de Energia Mínima, descrita em outros artigos, e enfatizada por MCGREGOR SMITH em [76], [73], que

fornece boas idéias para o trabalho do Problema de Enovelamento de Proteínas e DNA, sendo muito importante para vários ramos das Ciências na atualidade, principalmente aquelas que buscam a produção de drogas para curas de doenças como Câncer e AIDS.

OLIVEIRA [52] propôs em sua tese de doutorado um algoritmo denominado APOLONIO, baseado no algoritmo de Smith [75], como também realizou um estudo do problema de árvore mínima de Steiner euclidiana tridimensional (PASET), relacionando a função razão de Steiner ao problema de configuração de energia mínima de macromoléculas de interesse biológico. A seguir, na Figura 5 e Figura 6, tem-se exemplos de como a estrutura de árvore de Steiner euclidiana tridimensional aparece em algumas configurações moleculares.

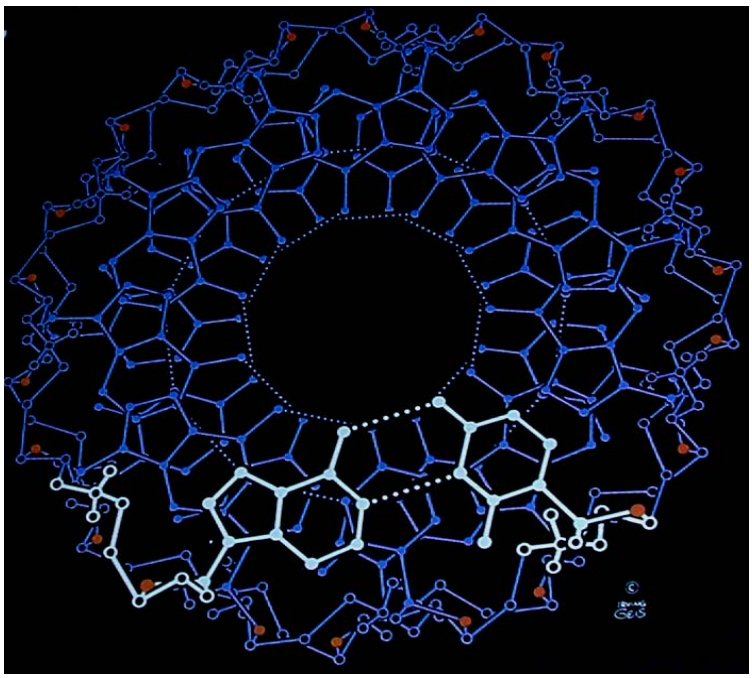


Figura 5: Forma menos hidratada do DNA (A-DNA) [52].

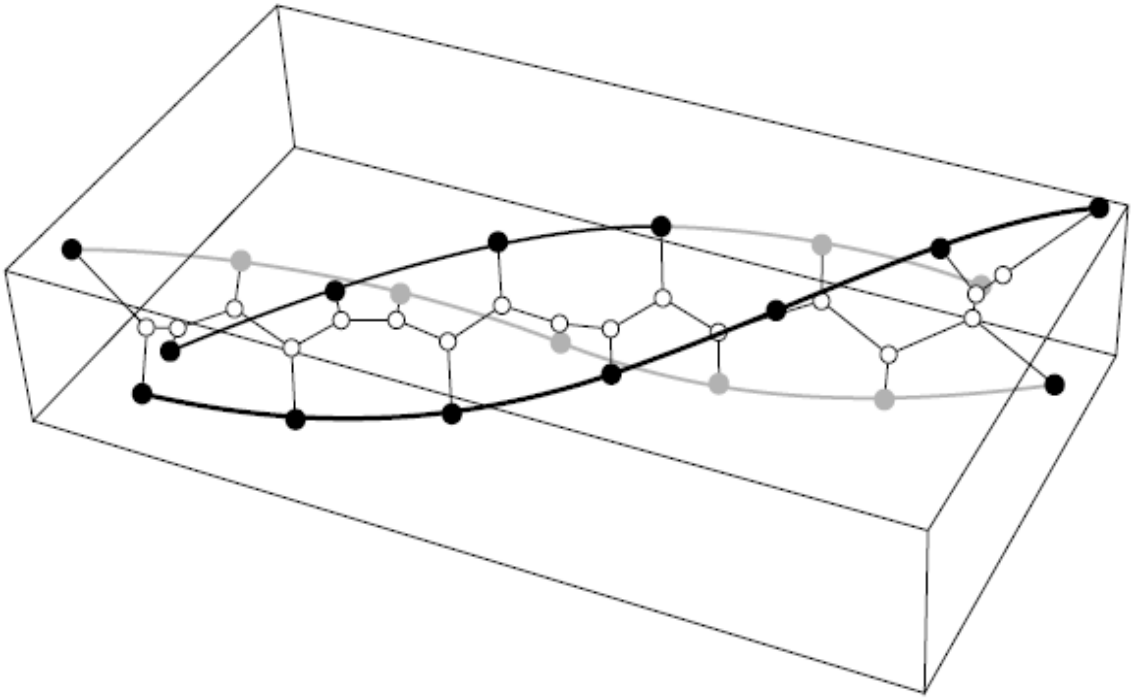


Figura 6: Hélice tripla com uma ASM para P=20 [76].

2.5.2 Aplicação do PASE no Projeto de Redes

O projeto dos mais variados tipos de rede (elétrica, minas, dutos, etc) com comprimento mínimo é de suma importância e tem grande impacto econômico em diversas situações. No caso da necessidade de realização do projeto de uma rede de tamanho mínimo em uma estrutura tridimensional, pode-se criar um modelo baseado no problema da árvore de Steiner euclidiano tridimensional (PASET), onde os pontos fixos são os pontos da rede que se deseja conectar. Assim sendo, resolvendo-se o PASET, obtém-se a rede de custo mínimo que interliga estes pontos fixos, fazendo uso, se necessário de pontos extras de conexão (pontos de Steiner). Na Figura 7 tem-se um exemplo de uma rede de túneis em uma mina subterrânea [4], onde os pontos numerados são os pontos fixos e os não numerados são os pontos de Steiner. Já na Figura 8, tem-se um exemplo de uma rede de dutos de ventilação, onde os círculos brancos são os pontos fixos e os círculos pretos são os pontos de Steiner.

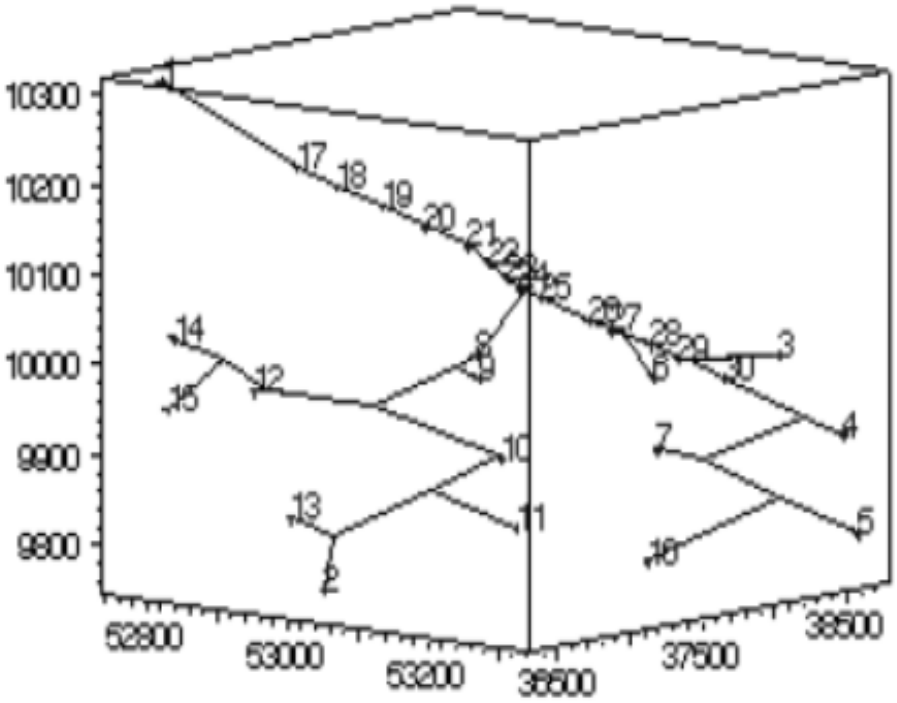


Figura 7: Rede de túneis em uma mina subterrânea.

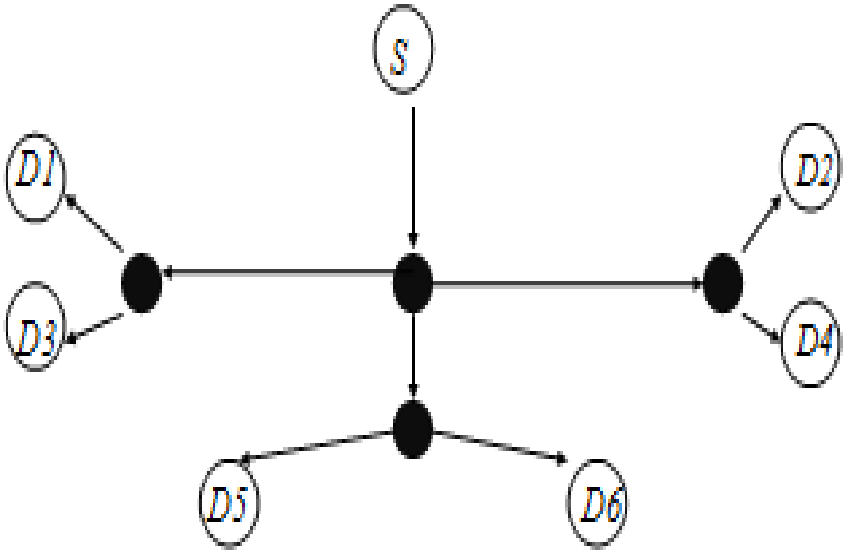


Figura 8: Rede de dutos de ventilação.

2.5.3 Aplicação do PASE na Inferência Filogenética

MONTENEGRO et al. em [46] propôs uma heurística baseada no algoritmo de otimização microcanônica para o problema de árvore de Steiner euclidiano no R^n com aplicação à inferência filogenética. Neste trabalho, MONTENEGRO et al. [46] cita que em dimensões maiores, o PASE é associado com problemas gerais de agrupamentos (*clustering*), incluindo os de inferência filogenética, que derivam árvores evolucionárias. Assim sendo, considerando o método da evolução mínima [85], o problema de encontrar árvores filogenéticas é tratado como o problema de encontrar uma árvore de Steiner de comprimento mínimo. A Figura 9 apresenta um exemplo de uma árvore filogenética onde os pontos fixos numerados de 1 a 19 são espécies, os pontos não numerados (pontos de Steiner) são os supostos ancestrais.

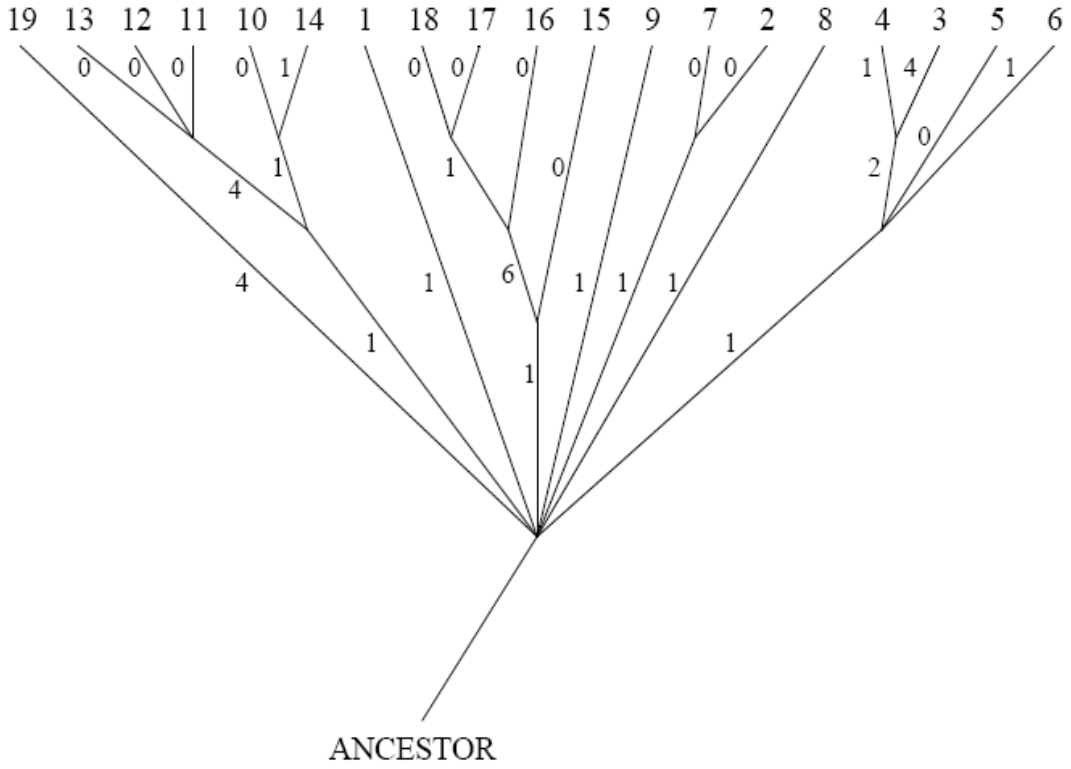


Figura 9: Árvore evolucionária para um conjunto de 19 espécies [46].

No capítulo 3 a seguir, serão apresentados os métodos propostos neste trabalho para solução do PASE.

Capítulo 3

Métodos Propostos Para a Resolução do Problema da Árvore de Steiner Euclidiana

Muitos Problemas de Otimização Combinatória (POC) de importância tanto prática quanto teórica (problema do caixeiro viajante, problema de alocação quadrática, *timetabling*, problema de escalonamento entre outros) consistem de uma busca pela "melhor" configuração (valores) das variáveis envolvidas no processo para alcançar o(s) objetivo(s) desejado(s) [33], [34], [41]. Contudo, os problemas de otimização combinatória de real interesse são NP-difíceis [51], [29], [34], [36], o que indica a não existência de algoritmos polinomiais para os mesmos, a não ser que $P = NP$ [30]. Neste caso, métodos completos e/ou exatos – métodos que garantem a obtenção da solução para um problema de otimização combinatória em tempo finito [51], [54] – em geral necessitam de tempo de computação exponencial no pior caso, levando a tempos de execução computacionais muito altos para fins práticos que envolvem instâncias de grande porte. Assim, o uso de métodos aproximados, principalmente heurísticas e metaheurísticas, têm sido muito aplicados para resolver POCs de grande porte [63], [67], [78]. Neste caso, abre-se mão da garantia de obtenção da solução ótima, em troca de se obter boas soluções em um tempo de execução computacional reduzido. A crescente importância das metaheurísticas é enfatizada pelo evento intitulado *Metaheuristics International Conference* (Conferência Internacional de Metaheurísticas) que acontece a cada dois anos (em 2009 será a sua oitava edição), que reúne os trabalhos no estado-da-arte na área.

Entre os métodos heurísticos e metaheurísticos que vêm já há algum tempo sendo bastante utilizados por obter bons resultados na solução de POCs e merecendo destaque na literatura é a metaheurística GRASP [63], [68] e os times assíncronos [81], [9], [50], [83] que serão abordados neste trabalho.

Este capítulo apresenta os algoritmos implementados para resolver o Problema de Árvore de Steiner Euclidiano no R^n. Dentre eles, há versões da heurística GRASP

(uma básica e outra híbrida) que diferem entre si pela utilização da técnica de aprimoramento chamada *path-relinking* e pelo tipo de implementação paralela (máquina de memória compartilhada ou distribuída ou híbrida). Posteriormente, também é apresentada uma proposta de uma heurística baseada em Times Assíncronos para o problema em questão.

3.1 GRASP

GRASP (*Greedy Randomized Adaptive Search Procedure* – Procedimento de Busca Gulosa Adaptativa Aleatorizada) é uma metaheurística multipartida, proposta por FEO e RESENDE [61], em que cada iteração é composta de uma fase de construção e de uma fase de aprimoramento. Durante o processo, a solução incumbente (melhor solução até o momento) é armazenada e atualizada sempre que a fase de aprimoramento resulta em uma solução melhor. Ao final de um número estabelecido de iterações, o algoritmo retorna a melhor solução encontrada.

A fase de construção gera uma solução viável para o problema através de um procedimento parcialmente guloso e parcialmente aleatório. A cada etapa da construção a seleção dos elementos que vão compor a solução é feita aleatoriamente em um subconjunto dos melhores elementos candidatos, denominada Lista Restrita de Candidatos (LRC). A fase de aprimoramento é uma busca local.

A Figura 10 apresenta o pseudo-código de uma versão tradicional do GRASP pra um problema de minimização [65]. Para facilitar o entendimento em linhas gerais, na Figura 11 é mostrado o fluxograma do GRASP da Figura 10. Neste algoritmo, o critério de parada adotado é ativado quando se atinge um determinado número de iterações (*MaxIter*) sem melhora da melhor solução que é definida pelo usuário (linha 2). A função f(S) retorna o custo da solução S e S^* é a solução incumbente. Este algoritmo tem duas fases principais, onde na primeira, executa-se a fase de construção (linha 3), que procura construir uma solução viável e de qualidade. Logo em seguida, na segunda fase, é executada a fase de busca local (linha 4), procurando refinar a solução inicial. Após a fase de construção e o refinamento da solução (busca local), é verificado se a solução encontrada é a melhor até o momento (linha 5). Em caso positivo, atualiza-

se a melhor solução encontrada (linha 6), e finalmente a melhor solução encontrada é retornada (linha 9).

Procedimento *GRASP-B*

1 $f(S^*) = +\infty$;
2 **Para** i de 1 até *MaxIter* **faça**
3 . Aplicar o procedimento de construção para obter uma solução viável S ;
4 . Aplicar busca local em S gerando uma nova solução S';
5 . **Se** custo de $f(S') < f(S^*)$ **então**
6 . $S^* \leftarrow S'$;
7 $i \leftarrow 0$;
8 . **Fim Se**
9 **Fim Para**
10 **Retornar** S^* ;

Figura 10: Pseudo-código do algoritmo GRASP tradicional.

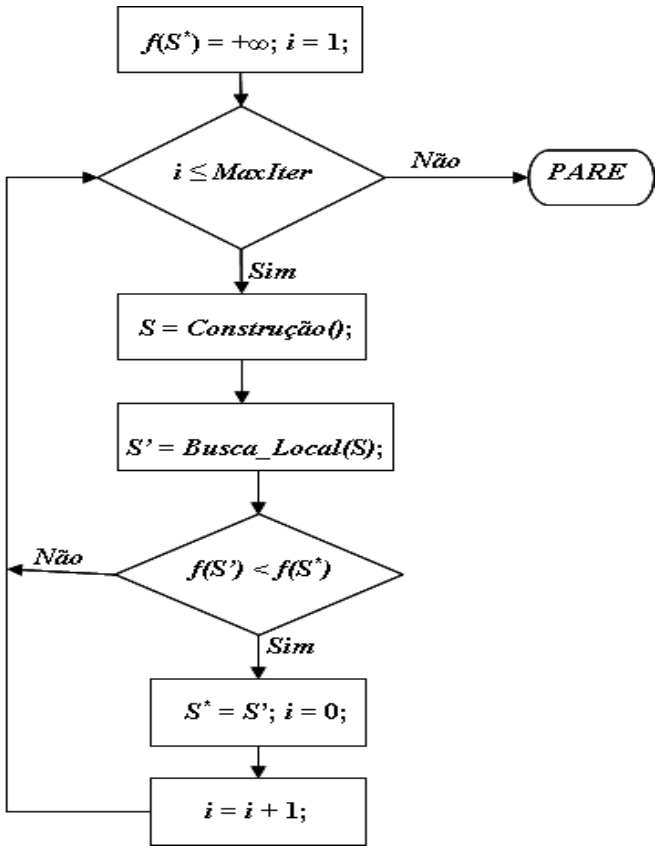


Figura 11: Fluxograma do algoritmo GRASP tradicional.

No GRASP apresentado, cada iteração é um procedimento independente, não havendo nenhum tipo de memória entre as iterações no decorrer da execução.

Porém, muitas técnicas podem ser agregadas ao GRASP tradicional (hibridismo), no intuito de aproveitar de alguma forma os resultados de iterações anteriores para tentar encontrar melhores soluções. Dentre essas técnicas, pode-se citar reconexão de caminhos (*path-relinking*), métodos de pesquisa em vizinhança variável (VND/VNS) e quaisquer procedimentos de aprimoramento sobre soluções já encontradas [66]. Além disso, outros elementos podem ser incorporados para aprimorar o desempenho do GRASP, como o uso de diferentes algoritmos de construção ou mesmo outras metaheurísticas: busca em vizinhança variada (ou VNS), busca tabu, busca local iterada, etc.

A seguir, serão descritas as fases de construção e busca local do GRASP proposto (GRASP-B).

3.1.1 Fase de Construção

Na fase de construção, a cada passo, um ponto dado (fixo) é considerado e um novo ponto de Steiner é adicionado à árvore e sua posição, como a dos outros pontos de Steiner já existentes, é otimizada de acordo com o critério de SMITH [75], formando ao final um determinado vetor topologia. As etapas do processo construtivo são as que seguem.

Etapa 1: Passo Inicial

- Gerar todos os vetores topologia de tamanho 1 e calcular os seus respectivos tamanhos (custos).
- Inseri-las na Lista Restrita de Candidatos (LRC).
- De acordo com os critérios da LRC escolher uma topologia.

Etapa 2: Repetir até que os vetores topologias descendentes tenham tamanho *P* - 3

- Gerar todas os vetores topologia descendentes imediatos do escolhida no passo anterior e calcular seus respectivos tamanhos (custos).
- Inserir os vetores topologia gerados na LRC.
- De acordo com os critérios da LRC escolher um vetor topologia.

O algoritmo construtivo é apresentado na Figura 12 e o seu respectivo fluxograma na Figura 13.

Procedimento Construção
1 Inicializar S = Ø ; //vetor topologia
2 Inicializar i = 1;
3 Defina T = todas as topologias de tamanho i;
4 **Enquanto** (i < P - 3) **Faça**
5 Calcular V^- = min{V(t) : t ∈ T}; //V(t) é o custo da árvore de topologia t
5 Calcular V^+ = max{V(t) : t ∈ T}; //V(t) é o custo da árvore de topologia t
6 Defina LRC = { t ∈ T : V(t) ≤ V^- + α x (V^+ - V^-)};
7 Selecione aleatoriamente t ∈ LRC;
8 i = i + 1;
9 Defina T = todas as topologias de tamanho i que descendem de t;
10 **Fim Enquanto**;
11 S = argmin{V(t): t\in T}; //selecionar a topologia de menor valor em T
12 **Retorne** S;
Fim Procedimento

Figura 12: Algoritmo construtivo para o PASE.

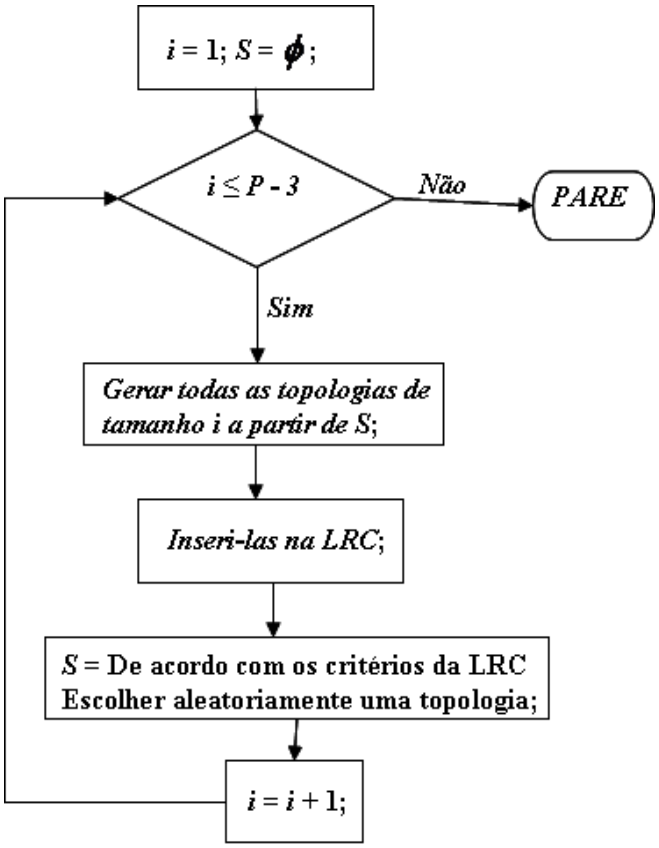


Figura 13: Fluxograma do algoritmo construtivo para o PASE.

Do algoritmo descrito na Figura 12, os passos de 1 a 3 correspondem à Etapa 1 e os passos de 4 a 10 correspondem à Etapa 2.

A seguir, é apresentado um exemplo do processo construtivo considerando $P = 6$ e $\alpha = 0$.

Exemplo para $P = 6$ e $\alpha = 0$

- Na etapa 1, geram-se as (vetores) topologias A(1), A(2) e A(3). Supor que a escolhida foi a A(2).

- Na primeira execução da etapa 2, geram-se todas as topologias descentes da escolhida no passo anterior (A(2)) que são: A(2,1), A(2,2), A(2,3), A(2,4) e A(2,5). Supor que a escolhida foi a A(2,3).

- Na segunda execução da etapa 2, geram-se todas as topologias descentes da escolhida no passo anterior (A(2,3)) que são: A(2,3,1), A(2,3,2),

A(2,3,3), A(2,3,4), A(2,3,5), A(2,3,6) e A(2,3,7). Supor que a escolhida foi A(2,3,4). A última topologia completa escolhida no passo 2 (neste caso A(2,3,4)), será a solução inicial da fase de busca local.

Assim sendo, dados *P* pontos fixos, ao final da fase de construção, o método provê um vetor topologia de tamanho *P* - 3.

3.1.2 Fase de Busca Local

Nesta fase de busca local parte-se de uma solução inicial gerada pelo método implementado na fase de construção. O algoritmo da busca local é apresentado na Figura 14 e o seu respectivo fluxograma na Figura 15.

```
Procedimento Busca_Local
1   parada = valor; //numero de iterações sem melhora
2   cont = 0;  vetor topologia inicial S; ρ = val(S);
3   Repita
4      Para z=1 até numvizinhos (pode ser P - 3) Faça
5        S' = S;
6        i = rand(P-3);        //sorteia posição de 1 a P-3
7        S'[i] = rand(2*i+1)  //sorteia novo valor 1 ≤ S'[i] ≤ 2*i+1 diferente de todos os outros S'[i] (S´[i] ∩ S' = ∅)
8        ρ' = val(S');
9        Se (ρ' < ρ) Então
10         S = S';   ρ = ρ';   cont = 0;
11       Fim Se
12     Fim Para
13     cont = cont+1;
14  Até (cont >= paradaBL);
15  Retorne S;
Fim Procedimento
```

Figura 14: Algoritmo de busca local para o PASE.

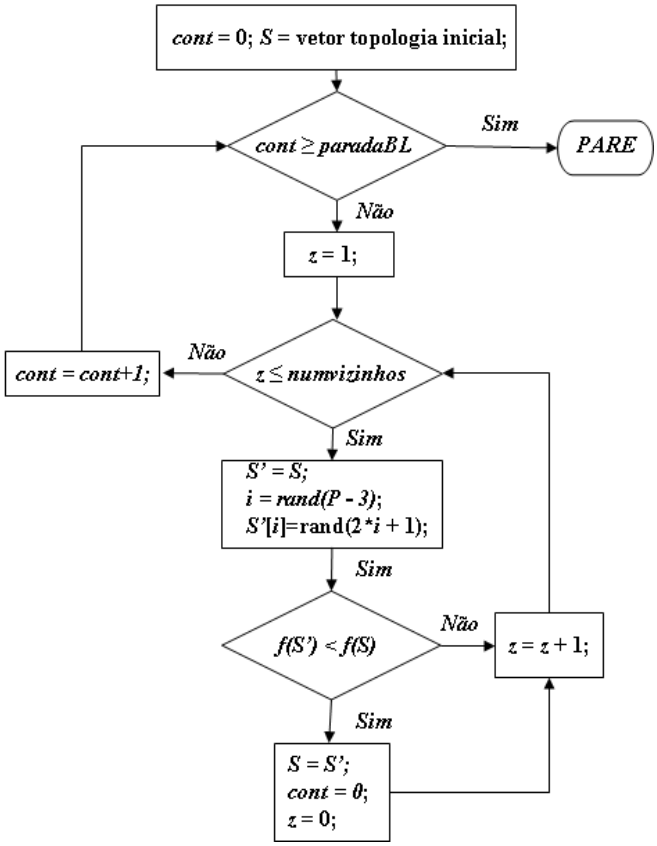


Figura 15: Fluxograma do algoritmo de busca local para o PASE.

A fase de busca local aproveita a solução inicial da fase de construção e explora a vizinhança desta solução, sendo esta uma representação de topologia ($P - 3$) – vetor A, conforme utilizado na fase de construção. O método aqui proposto faz uso de uma vizinhança simples, onde dado um vetor topologia A, seus vizinhos serão os vetores topologia A' obtidos pela mudança de apenas uma das $P - 3$ entradas de A. Continuando a explicação do método proposto, as seguintes considerações devem ser levadas em conta:

- se um melhoramento é encontrado, a solução corrente é atualizada e novamente a vizinhança ao redor da nova solução é pesquisada.
- o processo se repete por um determinado número de vezes especificado pelo usuário (o valor da variável parada - *paradaBL*).
- o número de vizinhos (*numvizinhos*) a serem pesquisado a partir de uma dada topologia é determinado (empiricamente) como se segue:
  - Se P <= 100 então *numvizinhos*=P*0.5; senão *numvizinhos*=P*0.2;

- Este cálculo é feito de modo a não se pesquisar toda a vizinhança de uma dada topologia, o que tornaria o método muito lento.

3.2 Path-Relinking

Uma das maneiras de se melhorar a qualidade das soluções obtidas pelo GRASP básico, é o uso de *path-relinking* (reconexão por caminhos). A técnica de *path-relinking* foi proposta originalmente em [29] como uma estratégia de intensificação, explorando trajetórias que conectavam soluções de elite obtidas por busca tabu ou *scatter search* [29], [32]. Neste contexto, na busca por melhores soluções são gerados e explorados caminhos no espaço de soluções, partindo de uma ou mais soluções de elite e levando a outras soluções de elite. Isto é alcançado selecionando-se movimentos que introduzem atributos das soluções guia na solução corrente. Esta técnica pode ser vista como uma estratégia que busca incorporar atributos das soluções de boa qualidade, favorecendo a seleção de movimentos que os contenham.

De acordo com [1] essa abordagem é denominada de *path-relinking* porque quaisquer duas soluções visitadas por um algoritmo de busca estão ligadas por uma seqüência de movimentos. Na Figura 16 são mostrados dois caminhos hipotéticos, que ligam uma solução inicial a uma solução guia através de uma seqüência de movimentos (para um problema de minimização, sem perda de generalidade). A linha pontilhada mostra soluções visitadas por um algoritmo de acordo com a metaheuristica GRASP e a linha contínua mostra as soluções visitadas pela *path-relinking*. Observa-se que as trajetórias seguidas pelas duas estratégias são diferentes. Isso ocorre principalmente porque em cada iteração do algoritmo GRASP, os movimentos são escolhidos de forma "gulosa", isto é, escolhe o movimento não proibido que minimize localmente o valor da função objetivo. Durante a *path-relinking*, o objetivo principal é incorporar atributos da solução guia à solução inicial, melhorando assim o valor da função objetivo. Como apresentado na Figura 16, o objetivo de realizar a *path-relinking* no contexto da metaheurística GRASP é obter soluções não visitadas pela trajetória original, que melhoram a função objetivo, possibilitando encontrar soluções melhores do que o GRASP trabalhando sozinho.

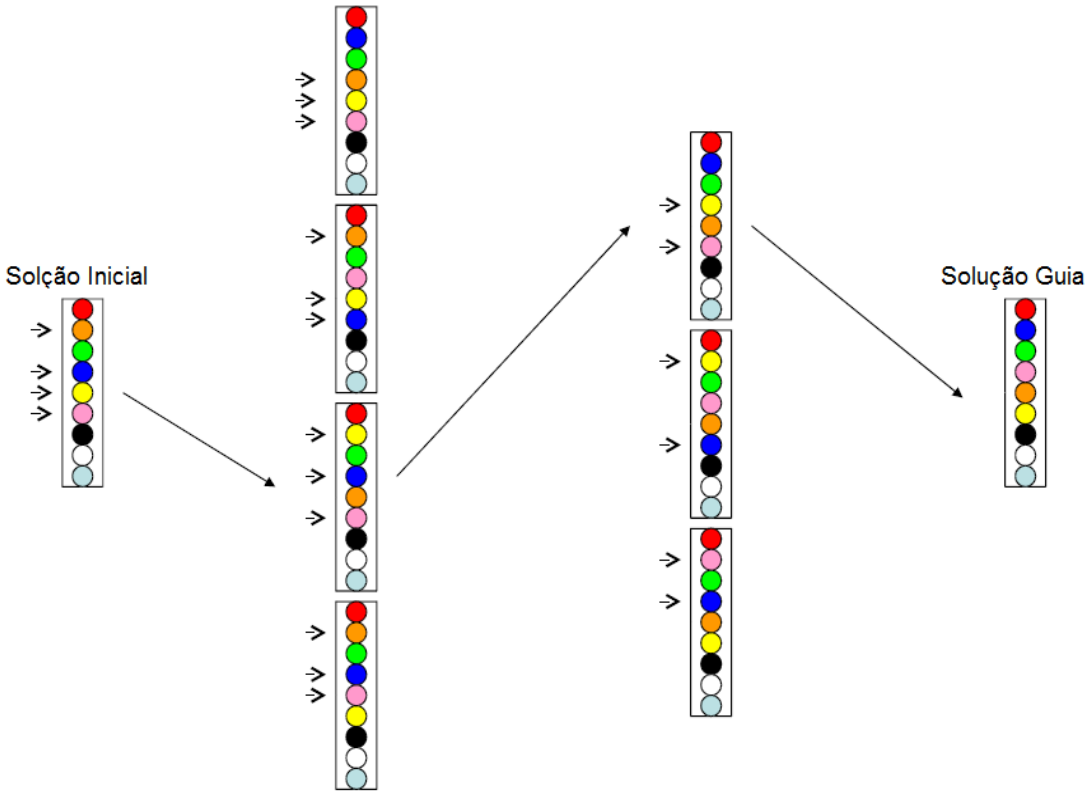


Figura 16: Exemplo do Path-relinking no contexto da metaheuristica GRASP [66].

Diversas estratégias de implementação são investigadas com detalhes por RESENDE e RIBEIRO [66]. De acordo com [66] as duas estratégias básicas de aplicação de *path-relinking* em procedimentos GRASP são as seguintes:

- *Path-relinking* aplicada como uma estratégia de pós-otimização entre todos os pares de soluções de elite;
- *Path-relinking* aplicada como uma estratégia de intensificação a cada ótimo local obtido após a fase de busca local.

A aplicação da técnica de *path-relinking* como uma estratégia de intensificação a cada ótimo local é mais eficaz do que empregá-la como um procedimento de pós-otimização [66]. Neste contexto, a *path-relinking* é aplicada a pares (x_1, x_2) de soluções, onde x_1 é uma solução localmente ótima obtida após o uso do procedimento de busca local e x_2 é uma solução selecionada aleatoriamente de um conjunto formado por um número limitado, *MaxElite*, de soluções de elite encontradas ao longo da execução do GRASP. Este conjunto está inicialmente vazio. Cada solução obtida pela busca local é

considerada como uma candidata a ser inserida no conjunto de elite, se ela é diferente de todas as outras soluções que estão atualmente neste conjunto. Se o conjunto de elite já possui *MaxElite* soluções e a candidata é melhor que a pior solução existente no conjunto, então a primeira substitui a última. Se o conjunto de elite ainda não está completo, a candidata é simplesmente inserida.

De acordo com [72] o algoritmo inicia computando a diferença simétrica $\Delta(x_1, x_2)$ entre x_1 e x_2, resultando no conjunto de movimentos que devem ser aplicados a uma delas (a solução inicial) para alcançar a outra (a solução guia). A partir da solução inicial, o melhor movimento ainda não executado de $\Delta(x_1, x_2)$ é aplicado à solução corrente, até que a solução guia seja atingida. A melhor solução encontrada ao longo desta trajetória é considerada como candidata à inserção no conjunto de elite e a melhor solução globalmente já encontrada é atualizada. Diversas alternativas têm sido consideradas e combinadas em implementações recentes [1]:

- Não aplicar *path-relinking* a cada iteração GRASP, mas sim apenas periodicamente;
- Explorar duas trajetórias potencialmente diferentes, usando primeiramente x_1 como solução inicial e posteriormente x_2;
- Explorar apenas uma trajetória, usando ou x_1 ou x_2 como solução inicial;
- Não percorrer a trajetória completa de x_1 até x_2, mas sim apenas parte dela (*path-relinking* truncada).

Dado que a técnica de *path-relinking* é a melhor forma de proporcionar ao GRASP básico a obtenção de melhora na qualidade das soluções obtidas [66], [25], a mesma será implementada como melhoria no GRASP com este intuito.

3.2.1 Path-Relinking para o PASE

Nesta seção é apresentada a implementação do *path-relinking* desenvolvido para o PASE seguindo as indicações descritas na seção 3.2.

Neste caso, o algoritmo parte de uma solução T_0, e passo-a-passo a transforma em outra solução T_d, onde cada solução é um vetor topologia. Neste trajeto, entende-se que pode ser encontrada uma solução melhor que T_0 e T_d. A solução T_0, de onde o procedimento inicia, é dita solução base e T_d, a solução a que se pretende chegar, é a solução guia.

Se T_0 e T_d são duas soluções (vetores topologia) com *d* arestas de inserções de ponto de Steiner diferentes entre si, um movimento de T_0 para T_d é a substituição de uma aresta de inserção de ponto de Steiner em T_0 por uma aresta de inserção de ponto de Steiner em T_d. Ou seja, tornando uma solução base T_0 e uma solução guia T_2, há duas arestas de inserção de pontos de Steiner (d = 2) diferentes entre as duas soluções, havendo duas possibilidades para a solução intermediária T_1.

O método *path-relinking* é realizado da seguinte forma: iniciando de T_0, um movimento para T_d é gerado e tem-se uma solução intermediária T_1; a seguir, mais um movimento é gerado, de T_1 a T_d; e assim por diante até que se chegue a uma solução T_{d-1}, após *d*-1 movimentos realizados.

A Figura 17 ilustra um exemplo do funcionamento do mecanismo. Como T_0 é o vetor topologia composto pelas arestas de inserção de pontos de Steiner {2, 4} e a solução guia T_d é o vetor topologia composto pelas arestas de inserção de pontos de Steiner {1, 5}, há duas diferenças entre as duas soluções (d = 2). Há, portanto, dois candidatos de T_0 para gerar T_1 (os vetores de arestas de inserção de pontos de Steiner associados são T^1_0={1, 4} e T^2_0={2, 5}). Suponha que a melhor opção é a troca da aresta de inserção de pontos de Steiner 2 por 1. Assim, é gerada a solução intermediária T_1 composta pelas arestas de inserção de pontos de Steiner {1, 4}. Para o movimento seguinte, a única possibilidade é a troca da aresta de inserção de ponto de Steiner 4 por 5. Com essa mudança, tem-se a solução guia T_2 e o procedimento termina.

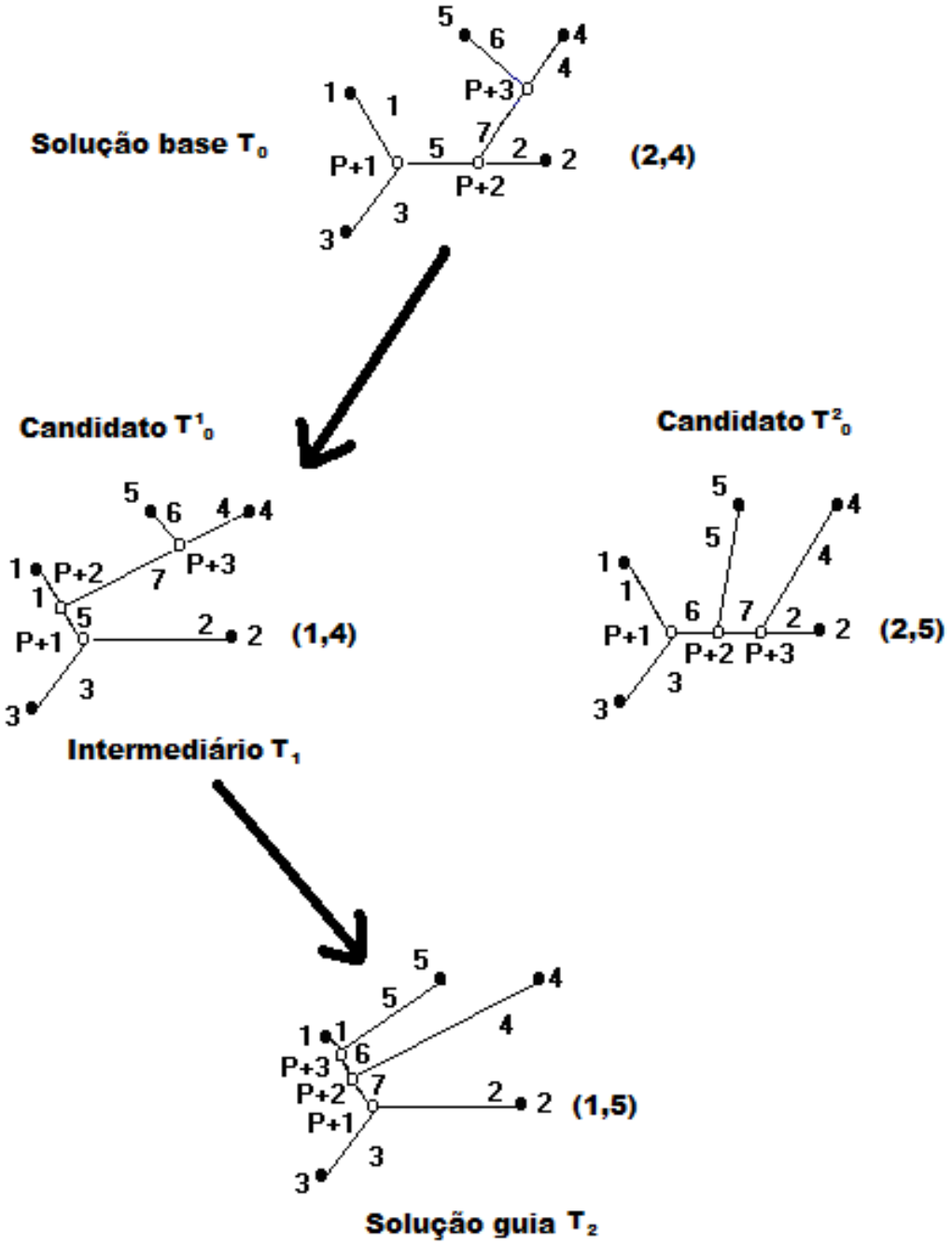


Figura 17: Ilustração do mecanismo de path-relinking para o PASE.

3.3 GRASP com Path-Relinking

As metaheuristicas têm provado ser altamente úteis e eficientes para encontrar soluções ótimas ou próximas da ótima para problemas difíceis de otimização combinatória [78], [10]. Mesmo assim, os pesquisadores da área continuam buscando por métodos cada vez mais efetivos. Nesta direção, têm sido desenvolvidas várias

heurísticas que não seguem os conceitos de uma única metaheurística. Estas heurísticas combinam um ou mais conceitos de diferentes heurísticas e metaheurísticas, como também algumas vezes fazem uso de idéias e técnicas de outras áreas, principalmente de programação matemática, inteligência artificial e estatística [61]. Estas abordagens alternativas são comumente chamadas de heurísticas ou metaheurísticas híbridas [61], [10]. A grande eficiência proporcionada por estes métodos têm levado a uma grande aplicabilidade e aceitação pela comunidade científica. De modo a corroborar esta afirmativa, verifica-se a realização de um evento anual na área, intitulado *International Workshop on Hybrids Metaheuristics* (Seminário Internacional de Metaheuristicas Híbridas) [61] que em 2008 se encontra em sua quinta edição.

Dentre diversas alternativas de hibridização, uma que vem obtendo bastante sucesso é a aplicação da técnica de *path-relinking* em conjunto com a heurística GRASP que tem como objetivo melhorar a qualidade das soluções obtidas pelo mesmo [66], [25], [69], [73]. O uso de *path-relinking*, conjuntamente com o procedimento GRASP, como uma estratégia de intensificação foi primeiramente proposto por LAGUNA e MARTIN [37]. De acordo com AIEX et al. [2] a estratégia de hibridização do GRASP com *path-relinking* tem levado a melhoras significativas na qualidade das soluções obtidas quando comparadas às soluções obtidas pelo método puro. Como resultado, a heurística GRASP hibridizada com *path-relinking* é mais efetiva que GRASP ou *path-relinking* executados separadamente [25], [69].

Na Figura 18 é ilustrado o pseudocódigo do algoritmo híbrido proposto (denominado GRASP-PR) utilizando a metaheurística GRASP com *path-relinking*.

Procedimento *GRASP-PR*

1 $f(S) \leftarrow 0$; *ConjElite* $\leftarrow$ Ø; *MaxElite* $\leftarrow n$;
2 **Para** *i* de 1 até *MaxIter* **Faça**
3 . Aplicar o procedimento de *construção*
. para obter uma solução viável *S* ;
4 . Aplicar *busca local* em *S* gerando
. uma nova solução *S'*;
5 . **Se** |*ContElite*| < *MaxElite* **então**
6 . . Verificar se S' $\in$ *ConjElite*;
7 . . **Se** *S'* $\notin$ *ConjElite* **Então**
8 . . . Inserir *S'* no *ConjElite*;
9 . . **Fim Se**
10 . **Fim Se**
11 . **Senão**
12 . . Verificar se *S'* é melhor que o pior
. do *ConjElite*;
13 . . **Se** *S'* melhor **Então**
14 . . . Verificar se S' $\in$ *ConjElite*;
15 . . . **Se** *S'* $\notin$ *ConjElite* **Então**
16 . . . . Inserir *S'* no *ConjElite*;
17 . . . **Fim Se**
18 . . **Fim Se**
19 . **Fim Senão**
20 . Aplicar *path_relinking em S'*;
21 . **Se** custo de $f(S^*) > f(S')$ **Então**
22 . . $S^* \leftarrow S'$;
23 . **Fim Se**
24 **Fim Para**
25 **Retornar** S^* ;
Fim Procedimento

Figura 18: Algoritmo GRASP com *path-relinking*.

No GRASP-PR, o usuário define o número de execuções que ele deseja para o algoritmo e o tamanho do conjunto de elite *MaxElite*. A partir de então, executa-se a

fase de construção (linha 3), que procura construir uma solução viável e de qualidade. Logo em seguida, é executada a fase de busca local (linha 4), procurando refinar a solução inicial.

Após os métodos de construção e busca local, é criado o conjunto de soluções de elite, conforme os passos a seguir:

- Inicialmente verifica-se se a quantidade de soluções elite é menor que *MaxElite* (linha 5); também é verificado se a solução S' pertence ao conjunto elite (linha 6).
- Se a solução S' não pertence ao conjunto elite (linha 7), então S' é inserido no conjunto elite (linha 8).
- Se a quantidade de soluções elite não for menor que *MaxElite*, é verificado se a solução S' é melhor do que a pior solução do *ConjElite* (linha 12). Caso a solução S' seja melhor (linha 13), verificar se a solução S' pertence ao *ConjElite* (linha 14).
    - Se a solução S' não pertence ao conjunto elite (linha 15), então a S' é inserida no conjunto elite (linha 16).
- Após a criação do conjunto de soluções de elite é aplicado o *path-relinking* (linha 20).
- Finalmente, após os métodos de construção, busca local e a aplicação do *path-relinking* a melhor solução é retornada (linha 25).

Seguindo as orientações fornecidas em Resende e Ribeiro [66], quanto às características do *path-relinking*:

- Iniciar o procedimento, considerando como solução base a melhor solução entre a solução elite selecionada (aleatoriamente) e a solução provida pela busca local S'.
- O tamanho do conjunto elite (*MaxElite*) considerado é 5.
- O algoritmo *path-relinking* é ativado (linha 20) sempre que a solução S', resultante da busca local, for no máximo *p*% pior que a melhor solução

do conjunto elite (*ConjElite*). Aqui, *p=iter*, onde *iter* é o número de iterações sem atualização do conjunto elite. Assim, quanto maior o número de iterações sem atualização do conjunto elite (*iter*), maior a probabilidade de se executar o *path-relinking*.

- Aplica-se o procedimento entre a solução S' e a solução *CE* do conjunto elite que maximize *d* (a mais diferente), sendo a solução base, a melhor dentre as duas.

Dado que o procedimento *path-relinking* é custoso computacionalmente para ser aplicado a cada iteração do GRASP, faz-se interessante utilizar artifícios que evitem a realização de esforço desnecessário [66]. Um dos artifícios que podem ser utilizados é a adição do recurso de memória. Neste caso, todos os pares de soluções já utilizadas como soluções guia e base são armazenadas numa tabela *hash*. Assim, um novo par de soluções (guia e base) só vai ser submetido ao procedimento de *path-relinking* se não constar na tabela *hash* (colisão). A técnica de *hash* utilizada neste trabalho foi com a função de *hash* pelo método da multiplicação e tratamento de colisão por encadeamento [14]. Outro artifício utilizado foi a utilização de uma estratégia de filtragem. Neste caso, uma solução só é utilizada como solução guia caso ela seja no máximo λ% pior do que a solução incumbente (melhor solução até o momento). Neste trabalho o valor utilizado para λ foi de 1%, baseando-se nos resultados obtidos por MARTINS et al. em [42].

3.4 Estratégias de Implementação para o GRASP Paralelo de Acordo com o Tipo de Máquina Utilizada

As heurísticas e metaheurísticas em geral são eficientes para resolver problemas de otimização combinatória. Contudo, quando são utilizadas para instâncias de problemas de otimização de grande porte, os tempos computacionais para exploração do espaço de soluções podem ser muito altos [16]. Com a grande disponibilidade de computadores pessoais, estações de trabalho e computadores paralelos cada vez mais velozes e redes de comunicações de alta velocidade, o desenvolvimento de implementações paralelas de heurísticas e metaheurísticas vem se apresentando como

uma alternativa muito promissora para acelerar o processo de busca por soluções aproximadas, aumentando em muito a possibilidade de se encontrar a solução ótima.

Além disto, implementações paralelas de heurísticas e/ou metaheurísticas possibilitam a resolução de problemas de otimização combinatória maiores e/ou encontrar soluções melhores que as encontradas em um único processador, devido ao particionamento do espaço de busca e as grandes possibilidades de intensificar e diversificar a busca no espaço de soluções [22], [17]. Outra vantagem da implementação paralela de heurísticas e/ou metaheurísticas é o aumento da robustez do método. Esta robustez é obtida pela combinação de diversas estratégias, oferecendo um nível consistente de desempenho sobre todos os tipos de instâncias do problema em questão e com pouco esforço na calibragem dos parâmetros do método [16].

Em suma, a implementação paralela de heurísticas e/ou metaheurísticas dá a elas a oportunidade de reduzir o tempo computacional e melhorar a qualidade das soluções obtidas por meio de buscas paralelas em múltiplas áreas do espaço de busca e utilizando diferentes parâmetros e/ou estratégias.

Dada a importância da área de metaheurísticas paralelas, ela já é atualmente um grande e importante campo de pesquisas [3].

Ressalta-se que a metaheurística a ser paralelizada será a metaheurística híbrida composta pelo procedimento GRASP em conjunto com *path-relinking*, aqui denominada GRASP-PRP. Como pode ser visto em [22], [2], [68], [70], tanto GRASP quanto GRASP em conjunto com *path-relinking* são bem adaptadas para implementações em paralelo.

Dado que a eficiência da implementação paralela da metaheurística adotada depende fortemente da estratégia adotada e da máquina utilizada, cada um destes fatores será explicado em maiores detalhes na seção 3.4.1.

3.4.1 Detalhes de Implementação

Nesta seção será apresentada a estratégia de paralelização adotada na implementação da metaheurística híbrida GRASP-PRP. Posteriormente, serão apresentados os tipos de máquinas paralelas utilizadas na execução de GRASP-PRP e como isto pode afetar o seu desempenho.

3.4.1.1 Estratégia de Paralelização Adotada

Nesta seção, será apresentado um esquema de paralelização do método GRASP com *path-relinking* (GRASP-PRP). Dado que a grande maioria das estratégias de paralelização do método GRASP encontradas na literatura [2], [16], [19], [68], [70], de acordo com as taxonomias descritas em [19], [81], seguem a abordagem de *Múltiplas Trajetórias Independentes* (MTI), esta será a adotada neste trabalho. Nesta abordagem a troca de informações é limitada à fase inicial e final de cada um dos p processos do sistema, onde um processo (mestre) faz na fase inicial a leitura dos dados, que são repassados aos p-1 processos restantes. Esta abordagem é baseada na passagem do número total de iterações (*MaxIter*) entre os p processos, onde cada processo é interrompido após executar $K = MaxIter/p$ iterações, ou encontrar uma solução tão boa quanto a solução alvo. Já na fase final, este processo mestre coleta a melhor solução obtida pelos outros processos. Ressalta-se que o interesse e a utilização da abordagem de *múltiplas trajetórias independentes* se dão pela fácil extensão, escalabilidade e adequação a máquinas paralelas de diversos tipos.

A heurística GRASP com *path-relinking* paralelo (GRASP-PRP) proposta é construída com base na versão seqüencial GRASP-PR descrita na Figura 18. No método GRASP-PRP cada processo executa uma cópia do método seqüencial GRASP-PR. Um detalhe importante que deve ser levado em consideração nesta abordagem, para garantir que os processos irão realizar iterações independentes, é assegurar que nenhum deles tenha a mesma semente do gerador de números aleatórios, buscando evitar desta forma, que sejam geradas soluções iguais. O processo mestre gera p sementes diferentes, para cada um dos p processos. Como nesta estratégia as iterações são completamente independentes e poucas informações são trocadas entre os processos durante toda a

execução do método, é esperado que o mesmo apresente *speed-up* (aceleração) linear [5], [6], desde que não haja um forte desbalanceamento de carga entre os processos. A Figura 19 mostra o esquema da abordagem utilizada.

As diferenças entre a versão seqüencial (GRASP-PR) e a versão paralela (GRASP-PRP) são descritas a seguir. A versão paralela da heurística GRASP-PR foi desenvolvida seguindo o modelo mestre-escravo, de acordo com o seguinte esquema:

- O mestre recebe o número de iterações a serem realizadas.

- O mestre divide o número de iterações pelo número de escravos mais um (aqui, o mestre também processa).

- O mestre e os escravos executam o mesmo trecho de código referente a heurística GRASP-PR seqüencial, com o seu respectivo número de iterações.

- O mestre é responsável por coletar as soluções advindas dos escravos em conjunto com a sua própria solução gerada e finalmente selecionar a melhor entre todas.

Alguns detalhes da implementação realizada de acordo com o esquema especificado na Figura 19 são:

- *MaxIter* iterações GRASP são divididas entre os *p* processadores. Cada um dos processadores executa, independentemente e em paralelo, *K* iterações ($K = MaxIter/p$) do procedimento GRASP.

- Cada processador deve ter uma semente (inicializador do gerador de números aleatórios) diferente.

- Levando-se em conta que cada iteração requer aproximadamente a mesma quantidade de computação (carga de trabalho regular) e que cada

iteração GRASP é independente das demais, cada processador deve executar o mesmo número (K) de iterações.

- A melhor solução dentre as melhores encontradas pelos processadores é identificada como a solução encontrada pelo algoritmo. O custo de comunicação é baixo, uma vez que uma única variável global (a melhor solução) é necessária.

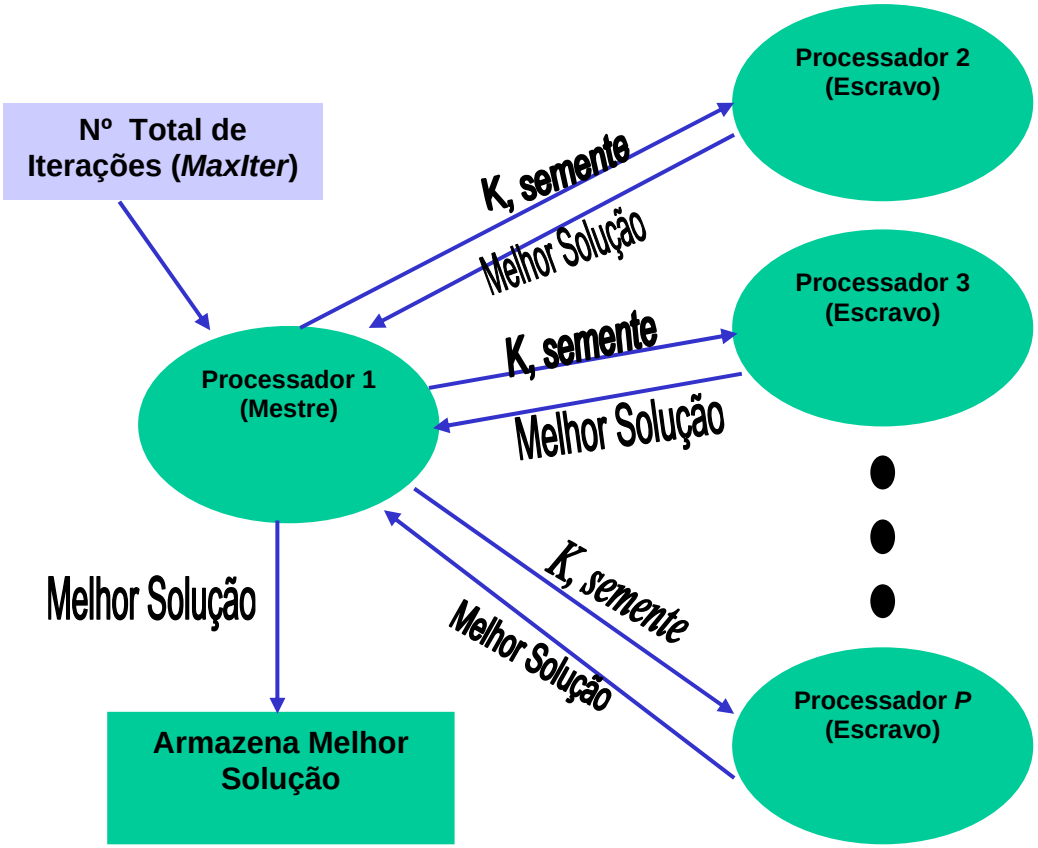


Figura 19: GRASP paralelo implementado.

Em suma, o interesse em se implementar uma versão paralela da heurística GRASP se faz claramente com objetivo de se ter um aumento de desempenho da técnica e pela sua facilidade de implementação, conforme estudos já realizados e apresentados em [1], [2].

3.4.1.2 Máquina Paralela Utilizada

Como já citado no início da seção 3.4, a grande disponibilidade de máquinas paralelas (computadores pessoais com processadores com múltiplos-núcleos, agregados (*clusters*) computacionais, supercomputadores, grades (*grids*) computacionais entre outros) e de redes de alta-velocidade já se faz presente [88], [27]. Assim, nada mais óbvio do que desenvolver aplicações para aproveitar todo este recurso computacional disponível com intuito de se ganhar desempenho, escalabilidade e possibilidade de resolver problemas maiores do que o possível em uma única máquina/processador. A implementação paralela de metaheurísticas faz parte deste rol de aplicações [22], [16].

Para que uma implementação paralela de uma metaheurística seja bem sucedida, é necessário que a estratégia adotada seja adequada. Esta adequação depende fortemente da arquitetura da máquina paralela na qual a implementação irá executar [16], [19]. Assim sendo, vale a pena citar os principais detalhes da máquina utilizada para execução dos métodos computacionais paralelos, realização dos testes e obtenção dos resultados.

A máquina paralela utilizada nos experimentos computacionais é uma SGI Altix 450, com a seguinte configuração:

- 32 CPUs Dual Core Intel Itanium2, totalizando 64 núcleos (*cores*);
- 128 Gb de memória global compartilhada NUMA;
- Interligação por uma rede local Infiniband [71]; e
- Sistema operacional Suse Linux Enterprise Server.

Esta máquina pertence ao Núcleo de Computação de Alto Desempenho (NACAD) [49] da Coordenação dos Programas de Pós-graduação de Engenharia (COPPE) da Universidade Federal do Rio Janeiro (UFRJ).

De acordo com a taxonomia de FLYNN [26], segundo fluxos de controle e dados, esta máquina é da família MIMD (*Multiple Instruction Stream, Multiple Data Stream*). Isto se deve ao fato da máquina SGI Altix 450 ser uma máquina de arquitetura

NUMA (*Non-Uniform Memory Access* – Memória de Acesso Não-Uniforme). A arquitetura NUMA é uma extensão à arquitetura SMP (*Symetric Multi-Processors* – multiprocessadores simétricos) bastante escalável, onde diversos processadores, cada um com sua memória local, são interligados por uma rede de altíssima velocidade (neste caso a Infiniband [71]), permitindo que cada processador acesse localmente sua memória e globalmente a memória dos outros [18], ou seja, o espaço de endereçamento é único (memória global compartilhada). Caso o acesso a posição de memória seja local ao processador, o tempo de acesso é mínimo. Por outro lado, se o acesso a posição de memória não for local, o tempo de acesso dependerá da rede de interconexão, sendo portanto, um modo de acesso não-uniforme. A Figura 20 apresenta um exemplo de arquitetura NUMA.

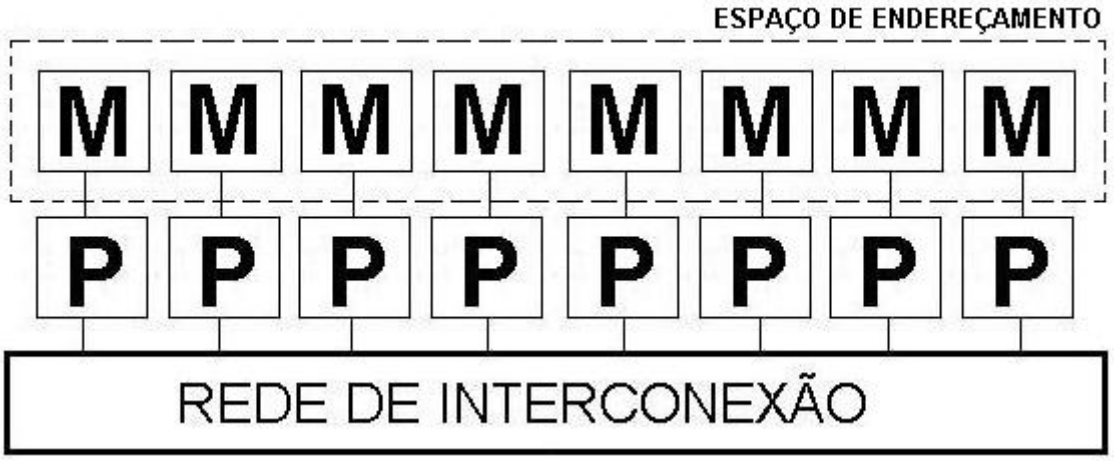


Figura 20: Arquitetura de uma máquina NUMA**,** onde **P** são os processadores e **M** são suas memórias associadas.

Uma máquina com a arquitetura NUMA também pode ser vista como um *Cluster* (agregado) computacional fortemente acoplado [35].

Ressalta-se que dada a arquitetura da máquina SGI Altix 450, ela pode ser vista como:

- uma máquina de processamento fortemente acoplado (memória compartilhada), quando da utilização de apenas um processador. Isto se deve ao fato dos processadores desta máquina serem *multi-core*, como pode ser visto na Figura 21;

- uma máquina de processamento distribuído (memória distribuída), como um *cluster* computacional, quando não se leva em consideração a existência do espaço de endereçamento único e se faz uso de bibliotecas de programação de passagem-de-mensagens (*message passing*), como por exemplo MPI (*Message Passing Interface*) e PVM (*Parallel Virtual Machine*) [77], [88], [35]; e

- uma máquina de processamento híbrido, onde se realiza computação paralela considerando os dois casos anteriores simultaneamente.

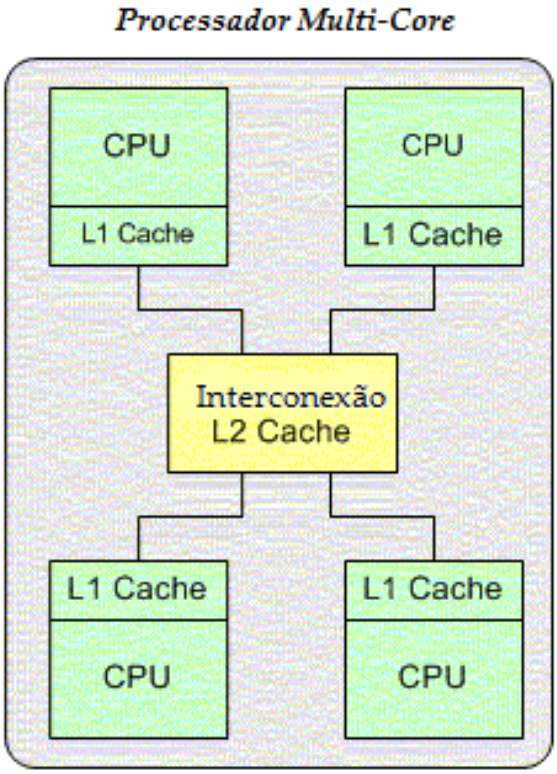


Figura 21: Arquitetura de processador multi-core.

Estes três tipos de arquiteturas de computadores paralelos são os mais comuns hoje em dia, assim influenciando diretamente na forma de implementar metaheurísticas paralelas [22], [19].

Nas seções seguintes, serão apresentados os esquemas de implementação da metaheurística GRASP de acordo com o tipo de máquina, conforme especificado

anteriormente para a máquina Altix 450. Os três tipos de máquinas a serem considerados são: memória compartilhada, memória distribuída e híbrida.

3.4.2 Implementação para Máquina de Memória Compartilhada

A implementação paralela da metaheurística GRASP (GRASP-PRP1) para a máquina de memória compartilhada, conforme especificada na seção 3.4.1.2, é realizada da seguinte maneira:

- segue o modelo especificado na seção 3.4.1.1 e exemplificado na Figura 19;
- somente um processador é utilizado;
- cada processo é executado em um *core* (núcleo) de um único processador; e
- foi utilizada a API (*Application Program Interface* – Interface de Programação Aplicativa) OpenMP (*Open Multi-Processing*) [53] para implementação do método paralelo de modo a explorar os recursos do processador *multi-core*.

Utilizou-se a API OpenMP (*Open Multi-Processing*), pois ela é apoiada por diversos fabricantes importantes de hardware e software (IBM, Microsoft entre outros), é aberta, é multiplataforma e está virando o padrão de referência para programação paralela para máquinas de arquitetura de memória compartilhada [53]. A API do OpenMP é baseada na combinação de diretivas de compilação, rotinas de bibliotecas e variáveis de ambiente que são usadas para especificar paralelismo para máquinas de memória compartilhada em programas codificados em Fortran e C/C++, o que torna o desenvolvimento de aplicações paralelas para este tipo de máquina (memória compartilhada) mais simples que as ferramentas disponíveis anteriormente, tais como: MPI, *threads* em Java, Open POSIX *threads* entre outros [13].

Ressalta-se que a exploração deste tipo de paralelismo se faz interessante, dado que a partir do ano de 2008, a grande maioria dos processadores lançados no mercado para computadores pessoais e servidores é *multi-core* (múltiplos núcleos).

3.4.3 Implementação para Máquina de Memória Distribuída

A implementação paralela da metaheurística GRASP (GRASP-PRP2) para a máquina de memória distribuída, cuja especificação foi fornecida na seção 3.4.1.2, foi realizada considerando que:

- o modelo é o especificado na seção 3.4.1.1 e exemplificado na Figura 19;
- são utilizados *p* processadores;
- cada processo é executado em um único processador;
- foi utilizada a biblioteca MPI [77] para implementação do método paralelo de processamento distribuído.

Para a implementação da metaheurística GRASP-PRP para a máquina de memória distribuída, utilizou-se a biblioteca MPI [77], [88], pelos fatores descritos a seguir. O MPI é uma biblioteca para passagem de mensagens, proposta como um padrão suportado por uma vasta base de parceiros acadêmicos e da indústria, e vem se tornando o padrão de referência para programação paralela para máquinas de memória distribuída, por ser aberta e multiplataforma. Dentre as diversas implementações do padrão MPI, a adotada neste trabalho foi a MPICH.

Faz-se importante citar que a exploração deste tipo de paralelismo já vem sendo realizada há tempos em *clusters* computacionais, NOWs (*Network of Workstations* – rede de estações-de-trabalho) e mais atualmente em *grids* (grades) computacionais.

3.4.4 Implementação para Máquina Híbrida

Considerando-se a máquina híbrida cuja especificação foi fornecida na seção 3.4.1.2, a implementação paralela da metaheurística GRASP (GRASP-PRP3) para tal computador foi feita levando em conta:

- seguir o modelo especificado na seção 3.4.1.1 e exemplificado na Figura 19;
- cada processo é executado em um único core (núcleo) de cada um dos *p* processadores; e
- foram utilizadas a API OpenMP [53] e a biblioteca MPI [77] para implementação do método paralelo de processamento para máquina híbrida.

Para a implementação da metaheurística GRASP-PRP para a máquina híbrida, utilizou-se OpenMP e MPI. Isto se deve ao fato que os computadores atuais permitem explorar muitas formas de paralelismo simultaneamente para tirar um máximo efeito combinado. Assim, um programa de memória distribuída usando MPI executa os processos em um conjunto de processadores (nós), onde cada nó é uma máquina de memória compartilhada (máquina *multi-core*) e executa em paralelo em múltiplos núcleos utilizando OpenMP.

3.5 Times Assíncronos

Nesta seção serão apresentados os principais conceitos e fundamentos a respeito de Times Assíncronos (*Asynchronous Teams* – A-Teams), como também a arquitetura proposta de um A-Teams para solucionar o problema da árvore de Steiner euclidiana no R^n de forma eficiente.

3.5.1 Fundamentos

Desde 1990, os A-Teams têm sido pesquisados e foi quando surgiu a primeira aplicação do método com sucesso, passando a ser aplicado na busca de soluções eficientes aos problemas de otimização.

O princípio básico de A-Team é que a combinação dos esforços de vários agentes (nome genérico utilizado para algoritmos neste contexto) construindo e modificando a solução potencialmente conduz a soluções de melhor qualidade do que aquela obtida por cada agente isoladamente. Em outras palavras, um A-Team é um conjunto de algoritmos atuando de forma coordenada, de modo que haja troca de informações entre os membros do conjunto (algoritmos).

Dentro da estrutura de um A-Team, a coordenação (também denominado controle) é descentralizada. Os agentes (algoritmos) são autônomos e agem nas memórias compartilhadas (lendo ou escrevendo soluções), de acordo com os seus próprios critérios. Toda troca entre os agentes (algoritmos) acontece através das memórias compartilhadas e a comunicação é assíncrona.

Uma característica fundamental de A-Teams é o fluxo cíclico de dados dentre os agentes que compõem o time. Agentes recuperam, transformam e depositam informações nas memórias compartilhadas. As soluções geradas são geralmente usadas como entrada de outros algoritmos. Este fluxo contínuo de informações permite a interação entre os agentes e conseqüentemente o aprimoramento das soluções geradas pelos algoritmos que compõem os A-Teams [81].

TALUKDAR e SOUZA [84] definem que:

A-Team é qualquer super-agente no qual os agentes são autônomos, os quais comunicam-se assincronamente e o fluxo de dados é cíclico.

Outra definição de times assíncronos é dada por RAIDL [61] que vê um time assíncrono como uma heurística e/ou metaheurística híbrida onde algoritmos (agentes)

diferentes colaboram e trocam informações, mas não são partes uns dos outros, numa estrutura heterogênea. Assim sendo, um A-Team é uma arquitetura para solução de problemas consistindo de uma coleção de agentes e memórias em uma rede direcionada fortemente cíclica.

A arquitetura de uma solução utilizando A-Teams é constituída empregando-se apenas dois elementos básicos: agentes e memórias compartilhadas. Ver a estrutura de A-Teams na Figura 22.

Figura 22: Representação de um A-Teams composto de dois agentes, uma memória compartilhada e fluxos de dados (setas).

Agentes são unidades autônomas, cujo sistema de controle é totalmente auto-contido. Não aceitam nenhuma instrução de seleção ou escalonamento de outros agentes ou qualquer outro tipo de elemento de controle externo. O agente tem três componentes: um operador (algoritmo que gera a solução), um sistema de comunicação que estabelece como ele vai escrever e ler nas memórias compartilhadas, e um sistema de controle que determina quando e como vai trabalhar [83].

Memórias são coleções de soluções que podem ser lidas ou depositadas (geradas) pelos agentes. Os agentes lêem as memórias para ter uma solução inicial a partir da qual podem gerar novas soluções, que por outro lado tornam-se disponíveis para os outros agentes que têm acesso a estas memórias. Componente de gerenciamento é dividido em duas funções de controle: programação e seleção. O controle de programação determina quando um agente é ativado e o controle de seleção determina onde ou em quais dados da memória de entrada o agente trabalhará. Os agentes se dividem em construtores e destrutores. Os agentes construtores são os que colocam novas soluções (tentativas) nas memórias. Já os agentes destruidores apagam soluções das memórias eliminando assim as menos apropriadas. Os destrutores podem ainda ser

empregados para eliminar padrões de produção de soluções indesejadas tais como seqüências de soluções repetidas.

Aplicações de A-Teams têm obtido êxito na busca de boas soluções para problemas de otimização combinatória, tais como: Caixeiro Viajante [79], Flow Shop Scheduling [56] e Job Shop Scheduling [9]. Isto trouxe motivação para investigar a adequabilidade de A-Teams como estratégia de Problemas de Otimização Combinatória. O A-Teams tende a ser melhor do que a "soma" dos resultados obtidos isoladamente por cada um dos agentes que o compõem. Além disso, o A-Teams tende a ser eficiente em escala, ou seja, o seu desempenho pode melhorar com a introdução de novos elementos (agentes e memórias) [83].

Devido às características de resolver problemas de otimização de forma sinérgica e intrinsicamente paralela, os A-Teams proporcionam diversas possibilidades de exploração de paralelismo, sendo facilmente implementados em sistemas paralelos multiprocessados com passagem de mensagens [50]. Alguns trabalhos [9], [50], [9], [79], [55] trazem resultados sobre a execução paralela de A-Teams, apontando para a possibilidade de se conseguir *speed-up* linear em relação ao tempo exigido para se produzir boas soluções.

3.5.2 Arquitetura Proposta para o Time

Nesta seção será apresentada a arquitetura proposta neste trabalho para o Time Assíncrono para resolver o problema da árvore de Steiner euclidiana no R^n, visando encontrar soluções ótimas ou próximas da ótima de forma eficiente.

A Figura 23 mostra a arquitetura do time proposto, com todos os seus agentes, memórias e fluxos de dados. Na seqüência, cada um destes componentes é explicado em detalhes.

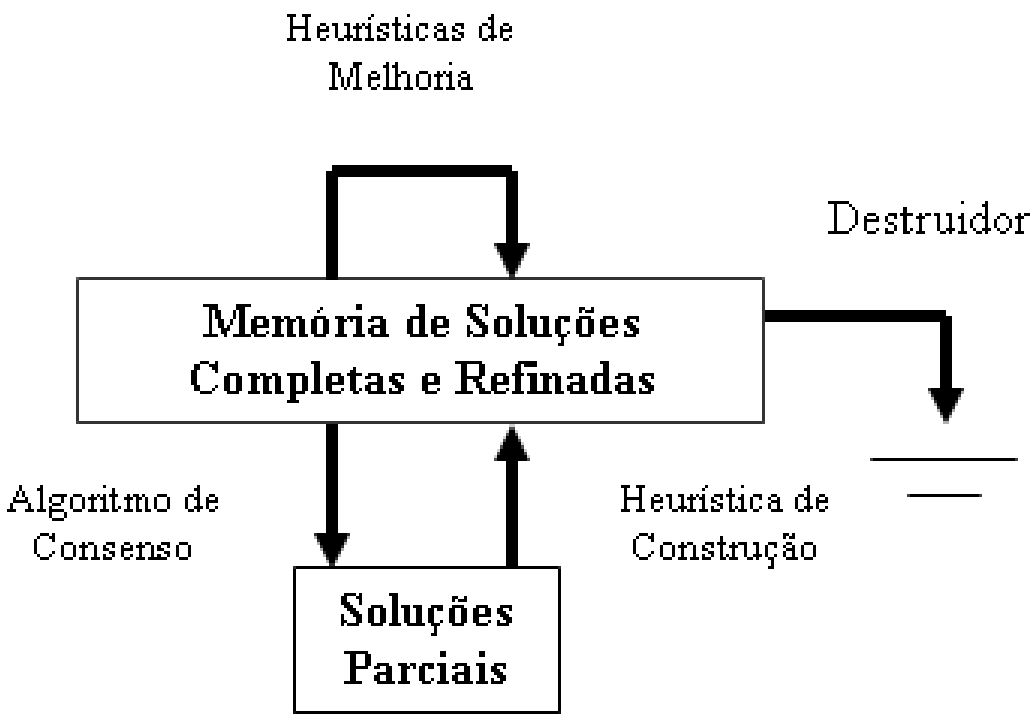


Figura 23: Arquitetura do Time Assíncrono proposto para solucionar o PASE.

De acordo com a Figura 23, cada componente do time assíncrono é classificado de acordo com a proposta de [55] da seguinte forma:

- A memória de soluções completas e refinadas armazena um número (*TamMem*) de soluções válidas e completas para o problema, onde cada solução consiste em um vetor topologia de Smith. Os critérios de entrada (aceitação) de uma nova solução na memória de soluções completas e refinadas são:
  - a memória ainda não está cheia; e
  - encontrou-se uma nova solução que não tenha uma outra igual na memória e que seja melhor que a pior armazenada (que será descartada pelo destruidor).

- Heurísticas de Construção: geram uma solução completa (válida) do problema a partir de uma solução parcial (incompleta ou nula). Os algoritmos desta classe constroem uma solução válida e completa elemento a elemento. Para o problema em questão, a heurística de

construção utilizada é a mesma da fase construtiva da heurística GRASP (descrita na seção 3.1.1), que será utilizada tanto para preencher a memória de soluções na fase inicial (quando ela está vazia) quanto para completar uma solução parcial/incompleta gerada pelo algoritmo de consenso.

- Heurísticas de Melhoria: partindo de uma solução válida, este tipo de heurística faz modificações nesta solução à procura de soluções melhores. Para o problema em questão, as heurísticas de melhoria utilizadas foram a fase de busca local utilizada na heurística GRASP (descrita na seção 3.1.2) e o método *path-relinking* (descrito na seção 3.2.1). O agente que executa a busca local é iniciado no momento que entra a primeira solução na memória de soluções completas e refinadas. No caso do método *path-relinking*, o seu agente começa a processar quando a memória de soluções completas e refinadas atinge pelo menos 10% do seu tamanho (*TamMem*). O agente *path-relinking* utiliza a memória de soluções completas e refinadas como o seu conjunto (*pool*) de soluções, sendo enviadas para ele duas soluções (uma como guia e outra como base) para serem trabalhadas.

- Algoritmos de Consenso: tomam como entrada as soluções da memória de soluções completas e refinadas, de onde extraem as partes comuns entre as soluções (intersecções). A idéia básica destes algoritmos é que coincidências entre soluções consistem em um bom ponto de partida para geração de uma nova solução (este conceito é bastante utilizando em algoritmos evolucionários [61]). Neste trabalho, o algoritmo de consenso proposto pega todas as soluções da memória de soluções completas e refinadas e faz um *AND*, ou seja, gera uma solução parcial onde as posições do vetor topologia de Smith que tem algum valor, os têm em comum em todas as soluções da memória de soluções completas e refinadas. Após, utiliza-se a heurística de construção citada anteriormente para completá-la e conseqüentemente gerar uma nova solução. Este algoritmo é ativado e/ou executado nas seguintes situações:

- a memória de soluções completas e refinadas fica cheia pela primeira vez; ou
- quando se passa um determinado tempo sem melhora (*TempoBest*) na melhor solução global obtida.

- O Destruidor é responsável pela eliminação de soluções da memória de soluções completas e refinadas, dado que o seu tamanho é limitado. O mesmo é ativado quando:
  - uma nova solução é aceita pelo critério de aceitação já citado, eliminado a pior solução (com maior valor da função objetivo); ou
  - se passa um determinado tempo sem melhora (*TempoBest*) na melhor solução global obtida, eliminando um percentual (*PercElim*) das piores soluções da memória de soluções completas e refinadas. Neste caso, antes da aplicação do Destruidor é executado o algoritmo de consenso. Após a execução do agente Destruidor, é executada a Heurística de Construção para preencher os espaços vazios gerados na memória pelas soluções eliminadas.

Como já citado, o time assíncrono é uma coleção de agentes (peças de software) que agem de forma cooperativa (trocando informações) e assíncronamente, criando uma sinergia (resultado coletivo igual ou geralmente melhor que as partes isoladas) para atingir o objetivo proposto. Assim sendo, o mesmo é particularmente interessante de ser implementado em máquinas com múltiplos processadores, em especial as de processamento distribuído. Na Figura 24 pode ser visto o esquema do time assíncrono proposto.

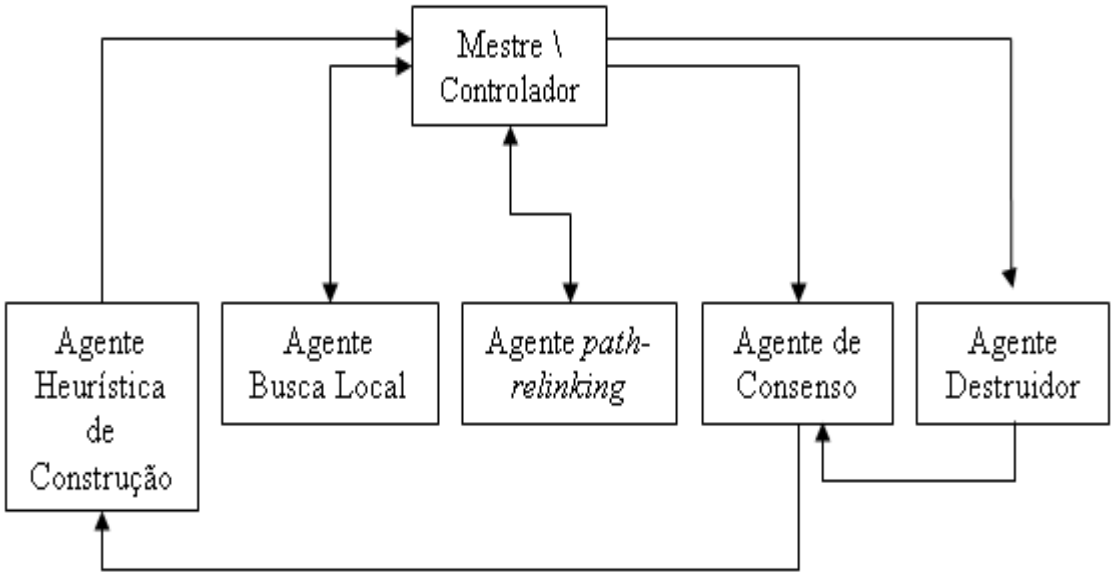


Figura 24: Esquema do Time Assíncrono Proposto.

A Figura 24 mostra o esquema do time assíncrono proposto neste trabalho especificamente para o PASE, onde cada retângulo representa um agente e as setas representam os fluxos de dados (informações) entre os agentes. Na implementação realizada, cada agente executa em uma unidade processadora diferente. Assim, tem-se:

- Agente Mestre ou Controlador: tem como tarefa principal coordenar as ações e os fluxos de informações para os agentes do time, como também gerenciar a memória de soluções completas e refinadas. O agente mestre ou controlador tem como principais funções:
  - criar novos agentes ou destruir os já existentes;
  - decidir quando o time deve parar de trabalhar, no caso deste trabalho, quando atinge o tempo máximo de processamento (*TempTot*); e
  - chamar os agentes para realizar as suas tarefas específicas e repassar os dados (informações) para os mesmos conforme necessário.
- Outros Agentes: os agentes de heurística de construção, busca local, *path-relinking,* consenso e destruidor tem suas ações realizadas como espeficiado anteriormente.

Em suma, em um time assíncrono, cada algoritmo (agente) procura apresentar suas melhores soluções (armazenadas na memória de soluções completas e refinadas) que podem ser reutilizadas como entrada para outro algoritmo (agente) do time. Um time assíncrono possui uma característica inerente de cooperação (sinergia) entre os algoritmos (agentes) aumentando as possibilidades de se encontrar melhores soluções do que aquelas encontradas por cada algoritmo isoladamente. No capítulo a seguir, mais especificamente na seção 4.4, serão apresentados os valores dos principais parâmetros particulares do time assíncrono descrito, bem como os resultados computacionais obtidos pelo mesmo.

Capítulo 4

Testes e Resultados Computacionais

Nesta seção, serão apresentados resultados da metaheurística GRASP [61] e do time assíncrono [81] obtidos sobre as instâncias de teste utilizadas em MONTENEGRO, MACULAN e TORREÃO [44], como também sobre outras instâncias de teste de maior dimensionalidade geradas neste trabalho. Estas instâncias serão utilizadas de modo a validar a eficiência dos métodos propostos para um problema tão complexo (o problema da árvore de Steiner euclidiana é NP-difícil [30]).

Aqui, os resultados computacionais obtidos pela versão proposta do GRASP seqüencial (GRASP-S) são comparados aos apresentados em MONTENEGRO, MACULAN e TORREÃO [44] e após sobre instâncias geradas de maior dimensionalidade. Posteriormente, são comparados os resultados computacionais das versões seqüenciais do GRASP, sendo uma o GRASP-S e a outra uma heurística GRASP híbrida com adição do método de *path-relinking*. Na seqüência, são feitos testes computacionais e análises dos resultados das três versões paralelas da heurística GRASP híbrida citadas nas seções 3.4.2, 3.4.3 e 3.4.4. Após, são apresentados os teste computacionais sobre o time assíncrono proposto. Finalmente, é feita uma análise geral dos métodos computacionais propostos neste trabalho para o PASE no R^n.

Em MONTENEGRO, MACULAN e TORREÃO [44] são realizados testes computacionais para um número de técnicas heurísticas e um método exato. As técnicas heurísticas utilizadas são: Soap Film [9], Algoritmos Genéticos (AG) [45] e Otimização Microcanônica (μO) [44], [86]; já o método exato utilizado é o algoritmo *branch-and-bound* (Exato) proposto em SMITH [75]. Para avaliar a qualidade da solução encontrada pelos métodos heurísticos, foi utilizada uma medida de referência conhecida como Razão de Steiner (ρ), definida como $\rho = L_{ASM}/L_{AGM}$, onde L_{ASM} é o comprimento da Árvore de Steiner Mínima (exata ou heurística) e L_{AGM} é o comprimento da Árvore Geradora Mínima (sem contar os pontos de Steiner), que é obtida em tempo polinomial [60]. Assim, um critério para que uma solução heurística com razão de Steiner ρH, possa ser considerada de boa qualidade é que atenda a $\rho H \leq 1$. Uma solução heurística

será tanto melhor quanto menor for o valor de ρH.

A metaheurística GRASP proposta neste trabalho foi implementada fazendo uso da Linguagem C [3] de programação e compilada com *gcc 3.3.2* utilizando a opção -O3 do compilador. Os testes foram executados numa máquina com a seguinte configuração: Pentium III de 1 Ghz (nome-código *coppermine* e 256 Kb de cache L2), 512 Mb de RAM e sistema operacional Linux RedHat 7.3 [9]. Já as técnicas apresentadas em MONTENEGRO, MACULAN e TORREÃO [44], também foram implementadas em C e executadas numa máquina Sun Ultra 1 de 200 Mhz.

Para cada instância, o GRASP foi executado utilizando os seguintes parâmetros:

- Alfa (α): sorteado aleatoriamente entre 0.1 e 0.3 a cada iteração do GRASP.
- Critério de parada do GRASP (*MaxIter*): 10 iterações sem melhora da melhor solução.
- Número de iterações sem melhora da busca local (*paradaBL*): 10.

A Tabela 1 reune os valores fixados para cada um dos parâmetros do GRASP-S.

Parâmetro	**Valor**
α	[0.1; 0.3]
MaxIter	10
paradaBL	10

Tabela 1: Valores dos parâmetros do algoritmo GRASP-S.

Adiante, serão feitas análises e comparações dos resultados obtidos pelo GRASP proposto com os apresentados em MONTENEGRO, MACULAN e TORREÃO [44] pelas técnicas heurísticas e o método exato. Para isto, foram realizados experimentos sobre quatro conjuntos com 1000 instâncias cada, correspondendo à distribuição aleatória de pontos obrigatórios em um cubo unitário e a quantidade dos mesmos indo de $P = 8$ à $P = 11$ para cada um dos quatro conjuntos. Ressalta-se que os experimentos realizados com GRASP, foram executados 10 vezes para cada instância de cada grupo,

cada vez com uma semente aleatória diferente, sendo apresentados os resultados médios. Isto é importante para analisar o comportamento (robustez) das técnicas, já que a metaheurística GRASP possui um componente probabilístico. Ressalta-se que em todos os experimentos, foram utilizados os mesmos valores dos parâmetros e que os mesmos foram determinados empiricamente, de modo a resultar numa boa relação entre o esforço computacional e a qualidade das soluções obtidas.

4.1 Resultados para o GRASP Seqüencial

Nesta etapa, serão apresentados os resultados computacionais da heurística GRASP seqüencial básica (GRASP-S).

Na Tabela 2, são apresentados os resultados computacionais obtidos pelo GRASP-S proposto para o PASE e os resultados extraídos de [44] para as técnicas concorrentes.

Tabela 2: Desempenho das técnicas apresentadas em [44] e do GRASP proposto para os quatro conjuntos de instâncias.

Algoritmo		*P*			
		8	9	10	11
Exato	ρ Médio	0,946761	0,946397	0,946758	0,946831
	Desvio Padrão	0,019507	0,017754	0,017531	0,015840
	Tempo Médio	1,400	7,800	43,000	240,000
Soap Film	ρ Médio	0,950232	0,950600	0,951185	0,951379
	Desvio Padrão	0,020169	0,018406	0,018003	0,016693
	Tempo Médio	0,059	0,067	0,079	0,083
	N° de Acertos	0	0	0	0
AG	ρ Médio	0,947165	0,947581	0,948360	0,948787
	Desvio Padrão	0,019643	0,018115	0,017851	0,016269
	Tempo Médio	22,000	31,000	39,000	51,000
	N° de Acertos	930	826	766	712
μO	ρ Médio	0,947507	0,947494	0,948521	0,948724
	Desvio Padrão	0,019637	0,017884	0,017828	0,016434
	Tempo Médio	22,000	28,000	34,000	37,000
	N° de Acertos	908	845	773	725
GRASP-S	ρ Médio	0,947015	0,947387	0,947571	0,948109
	Desvio Padrão	0,019563	0,018268	0,021365	0,016261
	Tempo Médio	13,480	16,755	20,823	23,873
	N° de Acertos	931	902	878	885

A Tabela 2 apresenta o valor de ρ médio, desvio padrão, tempo médio de CPU (em segundos) e o número de soluções ótimas encontradas (N° de acertos) por cada técnica para os quatro conjuntos de instâncias.

Na Tabela 2, pode-se observar que a metaheurística GRASP-S apresentou bom desempenho, encontrando a solução ótima um maior número de vezes e em baixo tempo computacional, principalmente quando comparado às outras metaheurísticas (AG e μO) propostas em [44], mesmo para as instâncias maiores. Ressalta-se também que o GRASP-S proposto apresenta um desvio padrão das soluções compatível com as técnicas concorrentes, além de encontrar a solução ótima com maior freqüência para todos os conjuntos de instâncias, comprovando sua robustez.

Vale a pena ressaltar, que mesmo pela diferença das máquinas onde as técnicas foram executadas (o processador Pentium III de 1.0 Ghz é aproximadamente cinco vezes mais rápido que o da SUN ULTRA 1 de 200 Mhz) [80], [57], os tempos apresentados pela metaheurística GRASP-S são menores que os das técnicas concorrentes diretas (AG e μO), mostrando que o bom desempenho deve-se a qualidade do GRASP-S proposto e não a diferença do hardware.

Para possibilitar esta comparação de um ponto de vista quantitativo, foi utilizada a abordagem tradicional de escalar a velocidade do *clock*. Em geral, tal escalamento é conservativo e favorece as máquinas mais lentas. Este escalamento se dá da seguinte forma: *clock* da máquina utilizada no experimento dividido pelo *clock* da máquina utilizada no experimento concorrente. Desta forma, chega-se ao valor 5 = 1000Mhz/200Mhz. Assim sendo, o tempo médio de CPU apresentado na Tabela 2, é o tempo real obtido multiplicado por 5. Desta forma, é possível realizar comparações diretas de tempo.

Nas comparações realizadas de acordo com esta metodologia, a metaheurística GRASP-S proposta apresentou bom desempenho nos resultados preliminares realizados.

4.1.1 Resultados Adicionais

Nesta seção serão apresentados resultados adicionais sobre instâncias de maior dimensionalidade entre as duas principais metaheurísticas competidoras - GRASP-S proposta neste trabalho e μO proposta em [44]. Para a realização dos testes adicionais, foram criados três conjuntos de instâncias com P=50, P=100 e P=250, onde cada conjunto possui 15 instâncias, sendo os pontos obrigatórios gerados aleatoriamente em um cubo unitário. Também foi utilizada a implementação do μO utilizada por MONTENEGRO et al. [44]; e os testes foram executados em uma mesma máquina (a mesma especificada no início do capítulo 4).

Na Tabela 3, são apresentados os resultados computacionais (ρ médio, tempo computacional médio em segundos de CPU e o número de melhores soluções encontradas - NMSE) do GRASP-S e do μO para os três conjuntos de instâncias (P=50, P=100 e P=250) utilizando os mesmos valores para os parâmetros conforme apresentados na Tabela 1, exceto para o parâmetro *MaxIter* que teve seu valor alterado para 100, dado que agora se tem instâncias de maior dimensionalidade. Nesta tabela não são apresentados os resultados referentes ao algoritmo exato de Smith, pois para as dimensionalidades das instâncias destes dois novos conjuntos, o tempo computacional é proibitivo - apresentando mais de duas horas de CPU, e não obtendo o resultado final, sendo as soluções obtidas até este ponto inferiores aos apresentados pelas metaheurísticas (ρ médio acima de 2,5%). Como não foi viável executar o algoritmo exato de Smith, não é possível computar o número de acertos de cada metaheurística em relação às soluções ótimas.

Tabela 3: Desempenho do μO e do GRASP proposto para os três novos conjuntos de instâncias de maior dimensionalidade.

Algoritmo		*P*		
		50	100	250
μO	ρ Médio	0,955351	0,966428	0,988245
	Desvio Padrão	0,022853	0,023394	0,037854
	Tempo Médio	36,853	81,953	186,751
	NMSE	0	0	0
GRASP-S	ρ Médio	**0,940796**	**0,948964**	**0,965472**
	Desvio Padrão	**0,021645**	**0,022035**	**0,029665**
	Tempo Médio	**21,374**	**41,503**	**120,581**
	NMSE	**15**	**15**	**15**

Os resultados computacionais apresentados na Tabela 3 são referentes à média de 10 execuções para todas as instâncias de cada um dos três conjuntos de testes, sendo cada uma com uma semente aleatória diferente. Para estes resultados médios, quando se obtém um resultado melhor para um determinado método, o mesmo está em negrito.

Como é possível observar na Tabela 3, a metaheurística GRASP-S proposta neste trabalho apresentou melhor desempenho computacional, tanto na qualidade da solução quanto no tempo de execução, sendo melhor do que a metaheurística μO concorrente. A diferença de desempenho é considerável, sendo de 1,5% a 3,4% para ρ médio e de 42% a 49% no tempo médio de execução.

Quando da avaliação da metaheurística GRASP-S contra a metaheurística μO quanto a sua efetividade relativa ao NMSE (número de melhores soluções encontradas), é possível observar que, considerando as 15 instâncias de cada conjunto de dados, GRASP-S encontrou melhores soluções em todos os casos.

Da análise dos resultados computacionais apresentados na Tabela 2 e Tabela 3, é aparente que o GRASP-S proposto supera μO em todos os experimentos realizados, mostrando assim ser mais efetiva para o problema em questão.

4.2 Resultados para o GRASP Seqüencial com Path-Relinking

Nesta seção, serão apresentados e analisados os resultados computacionais obtidos pelo GRASP seqüencial com *path-relinking* aqui denominado GRASP-PR, sobre as instâncias de teste especificadas na seção 4.1.1 para os quais se tem *P* variando de 50 a 250.

Dada as análises da Tabela 2 e Tabela 3, de onde se verifica que a heurística denominada GRASP-S apresenta resultados melhores do que os apresentados anteriormente na literatura, este será o método a ser utilizado para comparação com GRASP-PR.

A heurística GRASP-PR foi executada na mesma máquina e com os mesmos parâmetros do GRASP-S, e com os parâmetros próprios do *path-relinking* conforme estabelecido na seção 3.3 e os quais estão apresentados na Tabela 4.

Parâmetro	**Valor**
α	[0.1; 0.3]
MaxIter	100
paradaBL	10
MaxElite	5
p	*MaxIter*
λ	1%

Tabela 4: Valores dos parâmetros do algoritmo GRASP-PR.

Os resultados computacionais de GRASP-PR estão apresentados na Tabela 5.

Tabela 5: Desempenho do GRASP-S e do GRASP-PR proposto para os três conjuntos de instâncias de maior dimensionalidade.

Algoritmo		*P*		
		50	100	250
GRASP-S	ρ Médio	0,940796	0,948964	0,965472
	Desvio Padrão	0,021645	0,022035	0,029665
	Tempo Médio	21,374	41,503	120,581
	NMSE	**0**	0	0
GRASP-PR	ρ Médio	0,940796	0,939641	0,959331
	Desvio Padrão	0,021645	0,021885	0,024466
	Tempo Médio	23,593	45,834	130,458
	NMSE	**0**	**3**	**5**

Analisando os resultados apresentados na Tabela 5, a despeito de um pequeno aumento no tempo computacional, GRASP-PR obteve um melhor número de melhores soluções encontradas (NMSE) para as instâncias com *P* igual a 100 e igual a 250, obtendo três soluções melhores em 15 no primeiro caso e cinco soluções melhores em 15 no segundo caso que o GRASP-S. No caso de *P*=50 GRASP-PR e GRASP-S obtiveram as mesmas soluções. Nestes dois conjuntos (*P*=100 e *P*=250), a redução média das árvores de Steiner obtidas forma da ordem de 0,98% e 0,63% respectivamente.

Já para o caso de executar o GRASP-PR tendo como soluções alvo as propiciadas pelo GRASP-S, a Figura 25, Figura 26 e Figura 27 ilustram o número de instâncias em que cada algoritmo atingiu o alvo por unidade de tempo respectivamente para conjunto de instâncias (*P*=50, *P*=100 e *P*=250).

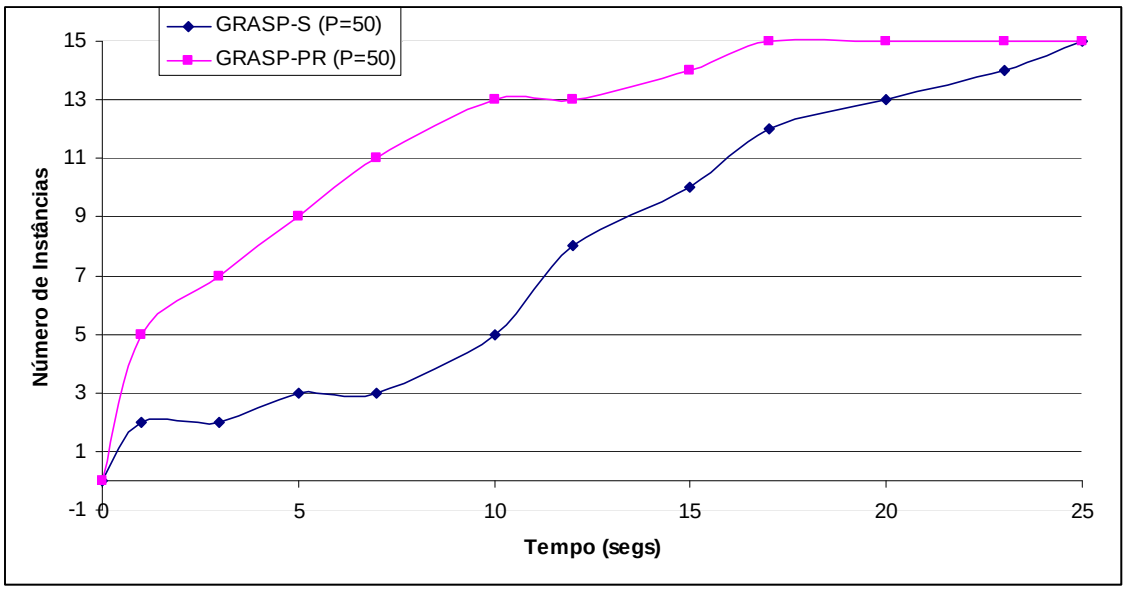


Figura 25: Número de alvos encontrados por GRASP-S e GRASP-PR para o conjunto de instâncias P=50.

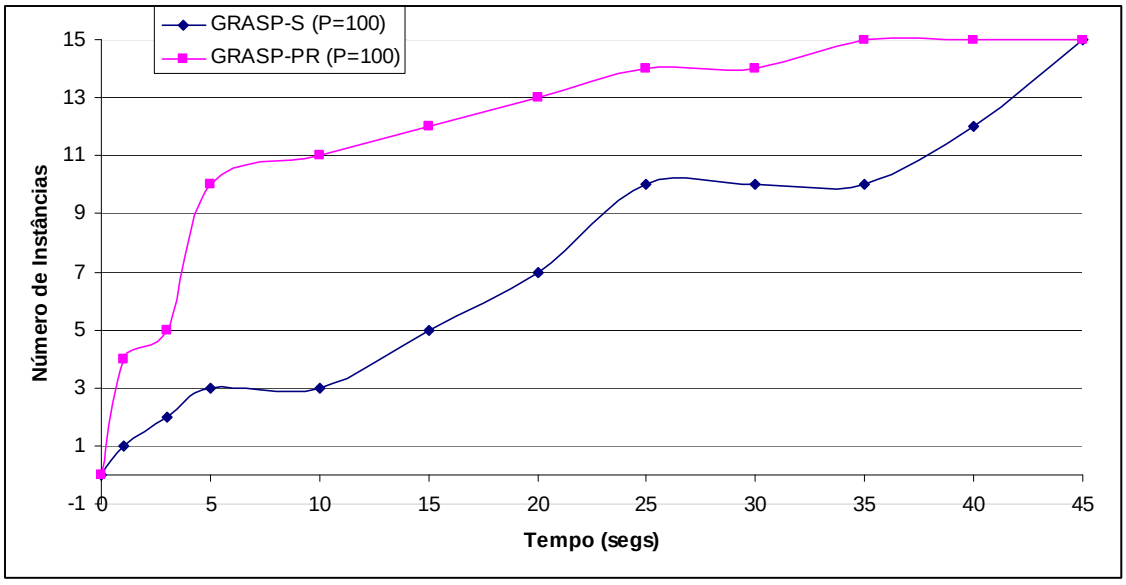


Figura 26: Número de alvos encontrados por GRASP-S e GRASP-PR para o conjunto de instâncias P=100.

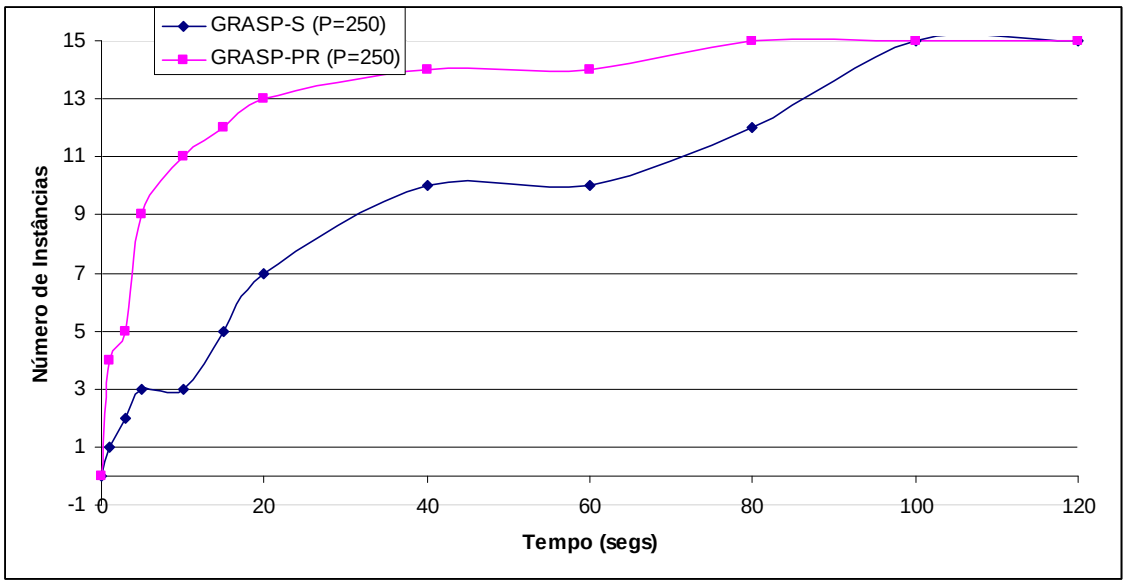


Figura 27: Número de alvos encontrados por GRASP-S e GRASP-PR para o conjunto de instâncias P=250.

Da observação da Figura 25, Figura 26 e Figura 27 fica evidente o impacto da utilização do procedimento *path-relinking* sobre a eficiência da heurística GRASP, pois se percebe que o número de alvos encontrados com GRASP-PR é sempre maior, em qualquer instante de tempo considerado, que o propiciado por GRASP-S em todos os conjuntos de instâncias. Um detalhe interessante a ser observado é que a heurística GRASP-PR consegue sempre encontrar 50% ou mais dos alvos em até 10 segundos de execução nos três conjuntos de instâncias. Para o conjunto de alvos especificados, o tempo médio do GRASP-PR para o conjunto de três instâncias foi de 40,66 segundos contra 51,33 segundos do GRASP-S.

4.3 Resultados para o GRASP Paralelo com Path-Relinking

A heurística GRASP híbrida com utilização do procedimento *path-relinking*, teve três versões paralelas desenvolvidas e implementadas conforme descrito respectivamente nas seções 3.4.2, 3.4.3 e 3.4.4, sendo cada uma delas denominada GRASP-PRP1, GRASP-PRP2 e GRASP-PRP3. Cada uma das versões paralelas da heurística GRASP-PRP foi implementada na linguagem de programação C, compilado com o gcc versão 4.1.2 utilizando a opção de compilação –O3, e fazendo uso da API OpenMP [53] no caso da versão 1 (GRASP-PRP1), da biblioteca *Message Passing*

Interface (MPI) de troca de mensagens [40], [59], [77] na versão 2 (GRASP-PRP2) e a API OpenMP e a biblioteca MPI conjuntamente no caso da versão 3 (GRASP-PRP3). Maiores detalhes a respeito das implementações paralelas do GRASP aqui utilizadas podem ser vistos na seção 3.4 deste trabalho.

Os testes computacionais foram realizados em uma máquina Altix SGI 450 disponível no Núcleo de Computação de Alto Desempenho (NACAD) [49] da Coordenação dos Programas de Pós-graduação de Engenharia (COPPE) da Universidade Federal do Rio Janeiro (UFRJ). Esta máquina possui 32 CPUs Dual Core Intel Itanium2 (totalizando 64 *cores*) e 128 Gbytes de RAM, e maiores detalhes da mesma estão apresentados na seção 3.4.1.2. Na execução das três versões do GRASP-PRP foram mantidos todos os parâmetros especificados anteriormente, conforme estabelecido na Tabela 4. A única variável adicional no processo foi o número de processadores ou processos (p) considerados em cada execução. Dado que a heurística GRASP com *path-relinking* (GRASP-PR) obteve melhor desempenho que a heurística GRASP básica (GRASP-S), os testes aqui realizados se limitaram à paralelização do GRASP-PR, denominado GRASP-PRP.

Foram realizados testes computacionais das três versões paralelas do GRASP-PRP sobre os três conjuntos de instâncias de maior dimensionalidade (P=50, P=100 e P=250) e com o número de processadores/processos (p) variando de 1 a 2 para o GRASP-PRP1 e variando de 1 a 16 em potência de dois, ou seja, p = {1, 2, 4, 8, 16}, para os GRASP-PRP2 e GRASP-PRP3. Também serão apresentados os *speed-up* das versões nas condições de teste citadas anteriormente. Um resumo das características das implementações paralelas do GRASP-PR pode ser visto na Tabela 6.

Tabela 6. Características das implementações paralelas do GRASP-PR.

Algoritmo	**Tipo de Máquina Utilizada**	**No. de Unidades Processadoras Utilizadas (p)**	**API e Biblioteca Utilizada**
GRASP-PRP1	Memória Compartilhada	1 e 2	OpenMP
GRASP-PRP2	Memória Distribuída	1, 2, 4, 8 e 16	MPI
GRASP-PRP3	Híbrida	1, 2, 4, 8 e 16	OpenMP e MPI

O *speed-up* é o fator de redução de tempo (aceleração) de execução de um programa paralelizado com *p* processadores, sendo a relação entre o tempo seqüencial e o tempo paralelo de um dado método computacional para resolver um problema particular numa máquina específica [59], [88], [27]. Ele é calculado da seguinte forma:

$$speed - up = \frac{\text{tempo para resolver um problema com o código sequencial}}{\text{tempo para resolver o mesmo problema com o código paralelo com p processadores}}$$

Em suma, o *speed-up* é uma medida de eficiência do programa paralelo implementado, onde o valor ideal é igual a *p* (*speed-up* linear), ou seja, a aceleração conseguida foi proporcional ao acréscimo de processadores.

Ressalta-se que qualquer das três versões do GRASP-PRP, considerando apenas um processador ou processo ($p = 1$), comporta-se como o GRASP-PR.

4.3.1 Resultados para a Máquina de Memória Compartilhada

Aqui, serão apresentados os resultados computacionais referentes às soluções e tempos de execução obtidos para cada conjunto de instâncias (P=50, P=100 e P=250) para a heurística GRASP paralela híbrida implementada conforme especificado na seção 3.4.2 (memória compartilhada) e denominada GRASP-PRP1. Dada a característica do processador da máquina em questão ter apenas dois núcleos, o número de processos (p) levados em consideração será até dois (2).

Os resultados obtidos estão apresentados na Tabela 7, Tabela 8 e Tabela 9. Posteriormente, são apresentados somente os valores dos *speed-up* nas condições de teste citadas anteriormente.

Tabela 7: Desempenho do GRASP-PRP1 proposto para o conjunto de instâncias *P*=50, variando o número de processos considerados.

Algoritmo		No. de processos (*p*)	
		1	2
GRASP-PRP1	ρ Médio	0,940796	0,940796
	Desvio Padrão	0,021645	0,021645
	Tempo Médio	12,364	6,244
	NMSE	0	0

Tabela 8: Desempenho do GRASP-PRP1 proposto para o conjunto de instâncias *P*=100, variando o número de processos considerados.

Algoritmo		No. de processos (*p*)	
		1	2
GRASP-PRP1	ρ Médio	0,939641	0,939641
	Desvio Padrão	0,021885	0,021885
	Tempo Médio	23,996	12,058
	NMSE	0	0

Tabela 9: Desempenho do GRASP-PRP1 proposto para o conjunto de instâncias *P*=250, variando o número de processos considerados.

Algoritmo		No. de processos (*p*)	
		1	2
GRASP-PRP1	ρ Médio	0,959331	0,959331
	Desvio Padrão	0,024466	0,024466
	Tempo Médio	66,901	33,959
	NMSE	0	0

Agora, são realizados os cálculos de *speed-up* para cada bateria de testes apresentadas na Tabela 7, Tabela 8 e Tabela 9.

Speed-up da Tabela 7 ➡ *Speed-up* = 12,364/6,244 = 1,98

Speed-up da Tabela 8 ➡ *Speed-up* = 23,996/12,058 = 1,99

Speed-up da Tabela 9 ➡ *Speed-up* = 66,901/33,959 = 1,97

Analisando os resultados apresentados na Tabela 7, Tabela 8 e Tabela 9, como também os cálculos de *speed-up*, tem-se as considerações a seguir. Como já era esperado, a heurística GRASP-PRP1 apresentou *speed-up* linear nas condições de teste na máquina utilizada, comprovando assim, que esta versão da implementação faz muito bom uso do acréscimo de unidades processadoras (*cores* ou núcleos). É esperado que este comportamento se mantenha quando o número de núcleos aumente, pois esta implementação é muito escalável, principalmente para a arquitetura da máquina (NUMA) utilizada neste trabalho. Já da observação dos NMSEs apresentados pelo GRASP-PRP1 em relação ao GRASP-PR, não se obteve nenhuma melhora. Isto se credita ao baixo número de unidades processadoras que se tem disponível. Como a tendência de mercado é que o número de núcleos só venham a aumentar, esta estratégia de implementação se mostra bem adaptada para explorar todo o recurso computacional disponível.

4.3.2 Resultados para a Máquina de Memória Distribuída

Agora, serão apresentados os resultados computacionais referentes às soluções e tempos de execução obtidos para cada conjunto de instâncias (P=50, P=100 e P=250) para a heurística GRASP paralela híbrida implementada conforme especificado na seção 3.4.3 (memória distribuída) e denominada GRASP-PRP2. Nos testes computacionais realizados para esta implementação, foram considerados os parâmetros da Tabela 4 e o número de processadores (p) levados em consideração variando de um a dezesseis (p=1,...,16), conforme citado na seção 4.3.

Os resultados obtidos estão apresentados na Tabela 10, Tabela 11 e Tabela 12. Já na Figura 28, Figura 29 e Figura 30 são apresentados os *speed-up* nas condições de teste citadas anteriormente.

Tabela 10: Desempenho do GRASP-PRP2 proposto para o conjunto de instâncias *P*=50, variando o número de processadores utilizados.

Algoritmo		No. de processadores (*p*)				
		1	2	4	8	16
GRASP-PRP2	ρ Médio	0,940796	0,940796	0,940796	0,940796	0,940796
	Desvio Padrão	0,021645	0,021645	0,021645	0,021645	0,021645
	Tempo Médio	12,384	6,893	3,839	1,783	0,969
	NMSE	0	0	0	0	0

Tabela 11: Desempenho do GRASP-PRP2 proposto para o conjunto de instâncias *P*=100, variando o número de processadores utilizados.

Algoritmo		No. de processadores (*p*)				
		1	2	4	8	16
GRASP-PRP2	ρ Médio	0,939641	0,939641	0,937547	0,937026	0,937026
	Desvio Padrão	0,021885	0,021885	0,021035	0,021008	0,021008
	Tempo Médio	23,996	12,469	6,802	3,906	1,978
	NMSE	0	0	1	2	2

Tabela 12: Desempenho do GRASP-PRP2 proposto para o conjunto de instâncias *P*=250, variando o número de processadores utilizados.

Algoritmo		No. de processadores (*p*)				
		1	2	4	8	16
GRASP-PRP2	ρ Médio	0,959331	0,959331	0,954794	0,951136	0,951085
	Desvio Padrão	0,024466	0,024466	0,024227	0,024191	0,024007
	Tempo Médio	66,901	34,996	18,823	9,634	4,873
	NMSE	0	0	2	3	4

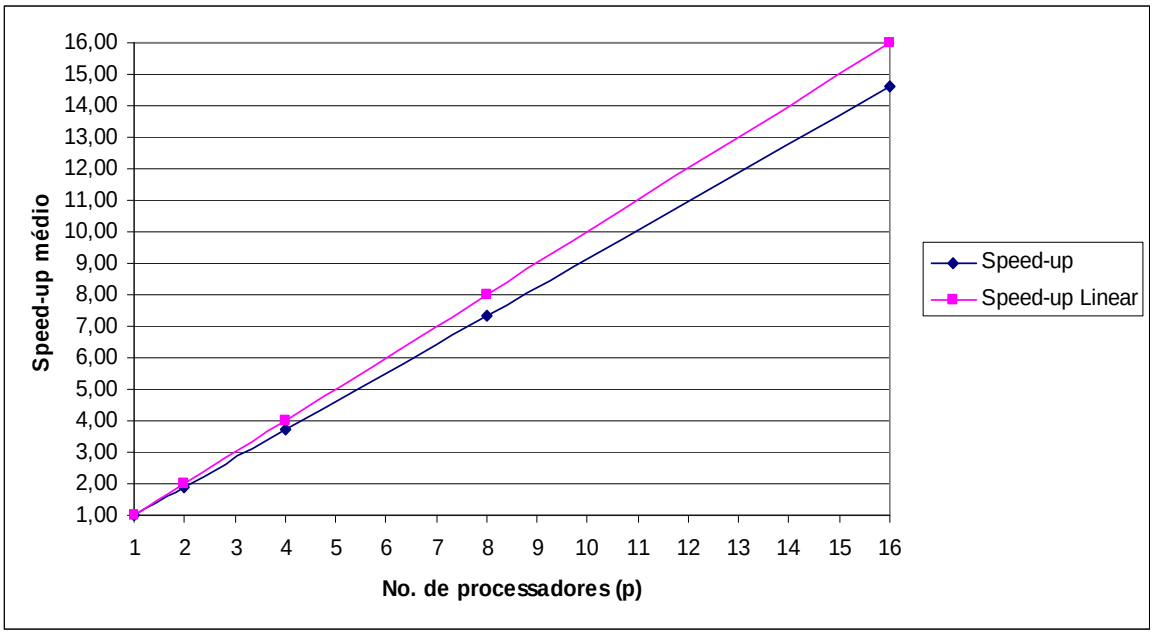


Figura 28: *Speed-up* dos testes apresentados na Tabela 10 para o conjunto de instâncias com *P*=50.

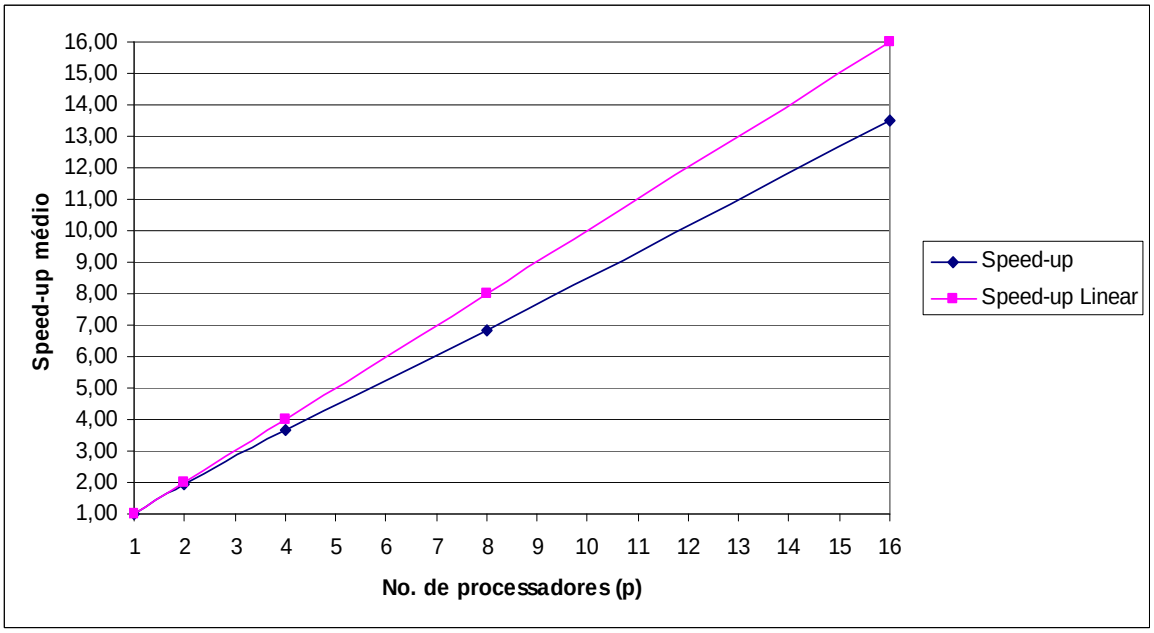


Figura 29: *Speed-up* dos testes apresentados na Tabela 11 para o conjunto de instâncias com *P*=100.

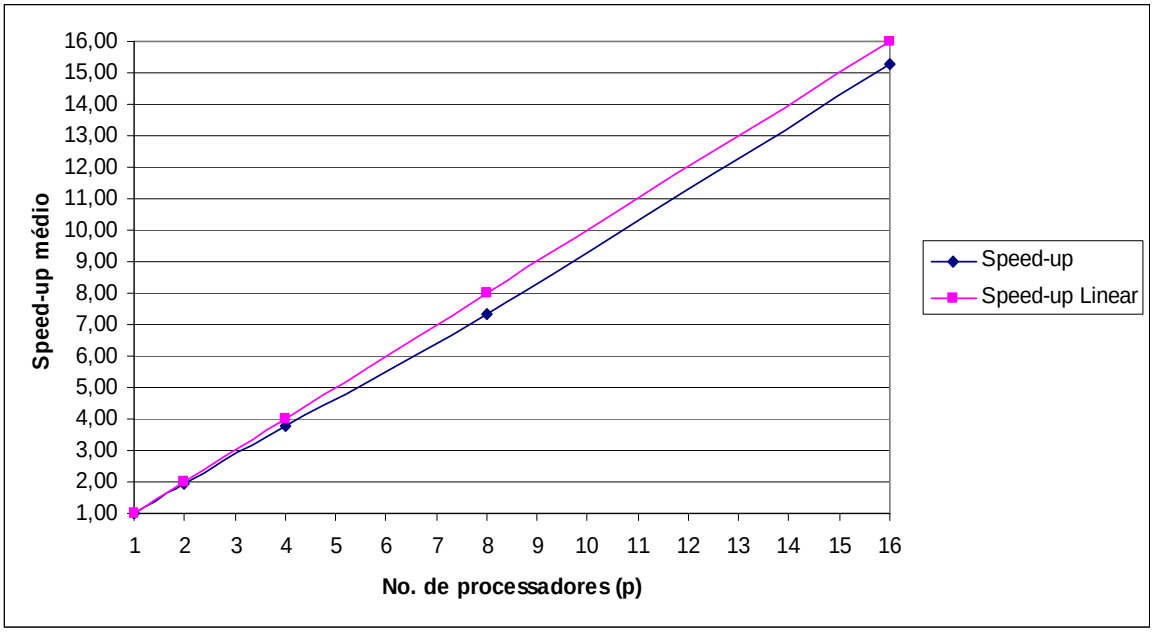


Figura 30: *Speed-up* dos testes apresentados na Tabela 12 para o conjunto de instâncias com *P*=250.

Das análises dos resultados apresentados na Tabela 10, Tabela 11 e Tabela 12, como também da Figura 28, Figura 29 e Figura 30, tem-se as considerações a seguir. Para o primeiro e menor conjunto de instâncias com *P*=50, GRASP-PRP2 encontrou sempre as mesmas soluções em relação ao GRASP-PR, independente do número de processadores (Tabela 8). Acredita-se que este comportamento se deve ao tamanho relativamente pequeno das instâncias e pelo poder computacional disponível. Contudo, apresentou uma redução no tempo computacional e um *speed-up* próximo do linear (Figura 28). Já para o segundo conjunto de instâncias com *P*=100, GRASP-PRP2 encontrou uma melhor solução em relação ao GRASP-PR com 4 processadores e duas melhores soluções em relação ao GRASP-PR com 8 e 16 processadores (Tabela 9). Como neste caso se tem instâncias de maior dimensionalidade, a utilização da computação paralela proporcionou uma maior sinergia entre os componentes do método, propiciando a obtenção de soluções melhores que no caso seqüencial. Neste caso, também se verifica redução do tempo computacional e um *speed-up* próximo do linear (Figura 29). Finalmente, para o terceiro conjunto de instâncias com *P*=250, GRASP-PRP2 encontrou duas (2) melhores soluções que GRASP-PR com 4 processadores, três (3) melhores soluções que GRASP-PR com 8 processadores e quatro (4) melhores soluções que GRASP-PR com 16 processadores (Tabela 10). Para este conjunto de instâncias de maior dimensionalidade, GRASP-PRP2 apresentou

resultados muito bons, com bom desempenho em relação ao GRASP-PR, pois devido o uso do paralelismo, aumentou-se a possibilidade de explorar melhor o espaço de busca de soluções. Mantendo o comportamento já apresentado nos conjuntos de instâncias menores (P=50 e P=100), GRASP-PRP2 apresentou redução no tempo computacional com o acréscimo de processadores e *speed-up* próximo do linear (Figura 30).

Da análise dos resultados apresentados por GRASP-PRP2, tem-se as conclusões apresentadas a seguir. A implementação paralela do GRASP-PR em máquina de memória distribuída, além de bastante escalável, permitiu a redução do tempo computacional em todos os conjuntos de instâncias, proporcionando a obtenção de soluções ainda melhores que no caso seqüencial, principalmente nas instâncias de maior dimensão (P=100 e P=250). Observou-se também, que GRASP-PRP2 obteve um *speed-up* próximo do linear para todos os conjuntos de instâncias testadas.

4.3.3 Resultados para a Máquina Híbrida

Agora, serão apresentados os resultados computacionais referentes às soluções e tempos de execução obtidos para cada conjunto de instâncias (P=50, P=100 e P=250) para a heurística GRASP paralela híbrida implementada conforme especificado na seção 3.4.4 (máquina híbrida) e denominada GRASP-PRP3. Nos testes computacionais realizados para esta implementação, foram considerados os valores dos parâmetros conforme estabelecido na Tabela 4 e com o número de elementos processadores (p) levados em consideração variando de um a dezesseis (p=1,...,16), conforme citado na seção 4.3. Neste caso, para a máquina em questão, um elemento processador é um núcleo e a cada dois núcleos se tem um processador. Logo serão utilizados oito (8) processadores.

Os resultados obtidos estão apresentados na Tabela 13, Tabela 14 e Tabela 15. Já na Figura 31, Figura 32 e Figura 33 são apresentados os *speed-up* nas condições de teste citadas anteriormente.

Tabela 13: Desempenho do GRASP-PRP3 proposto para o conjunto de instâncias *P*=50, variando o número de unidades processadoras utilizadas.

Algoritmo		No. de unidades processadoras (*p*)				
		1	2	4	8	16
GRASP-PRP3	ρ Médio	0,940796	0,940796	0,940796	0,940796	0,940796
	Desvio Padrão	0,021645	0,021645	0,021645	0,021645	0,021645
	Tempo Médio	12,384	6,393	3,174	1,579	0,783
	NMSE	0	0	0	0	0

Tabela 14: Desempenho do GRASP-PRP3 proposto para o conjunto de instâncias *P*=100, variando o número de unidades processadoras utilizadas.

Algoritmo		No. de unidades processadoras (*p*)				
		1	2	4	8	16
GRASP-PRP3	ρ Médio	0,939641	0,939641	0,937547	0,937026	0,937003
	Desvio Padrão	0,021885	0,021885	0,021035	0,021008	0,020927
	Tempo Médio	23,996	12,369	6,412	3,077	1,525
	NMSE	0	0	1	2	3

Tabela 15: Desempenho do GRASP-PRP3 proposto para o conjunto de instâncias *P*=250, variando o número de unidades processadoras utilizadas.

Algoritmo		No. de unidades processadoras (*p*)				
		1	2	4	8	16
GRASP-PRP3	ρ Médio	0,959331	0,959331	0,954794	0,951136	0,950085
	Desvio Padrão	0,024466	0,024466	0,024227	0,024191	0,023984
	Tempo Médio	66,901	33,960	17,803	8,647	4,251
	NMSE	0	0	2	3	5

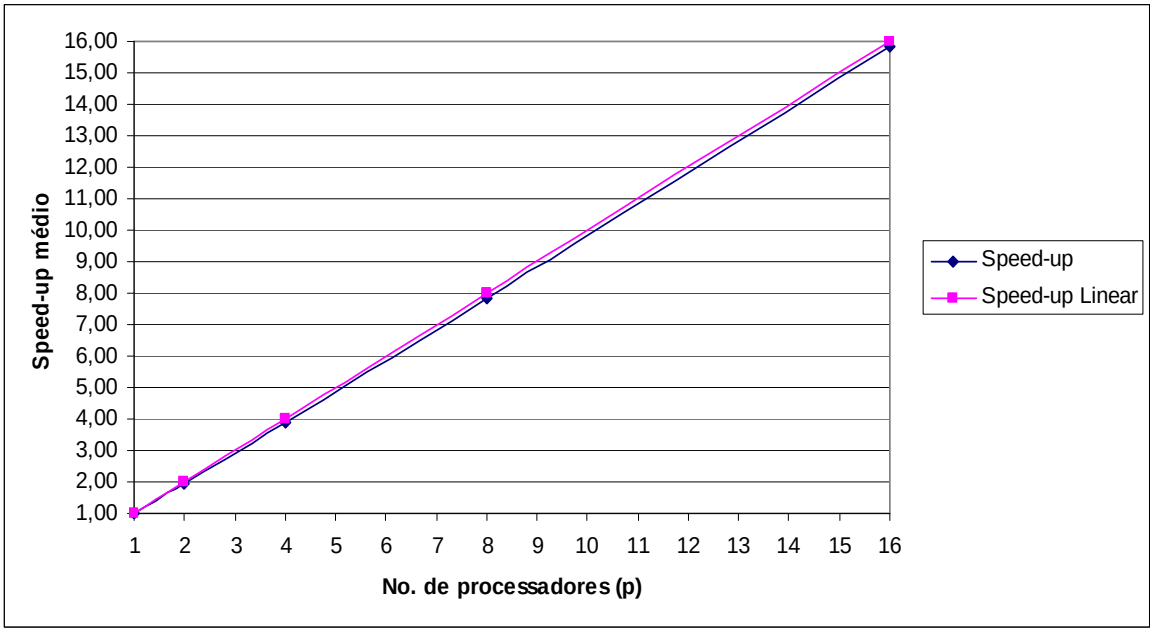


Figura 31: *Speed-up* dos testes apresentados na Tabela 13 para o conjunto de instâncias com *P*=50.

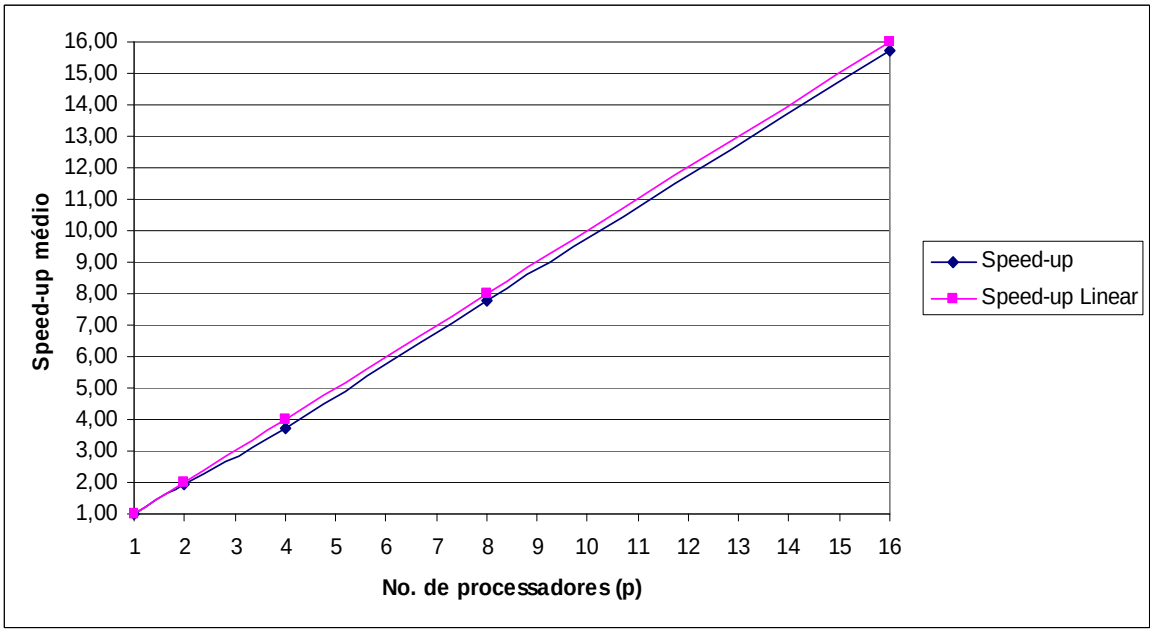


Figura 32: *Speed-up* dos testes apresentados na Tabela 14 para o conjunto de instâncias com *P*=100.

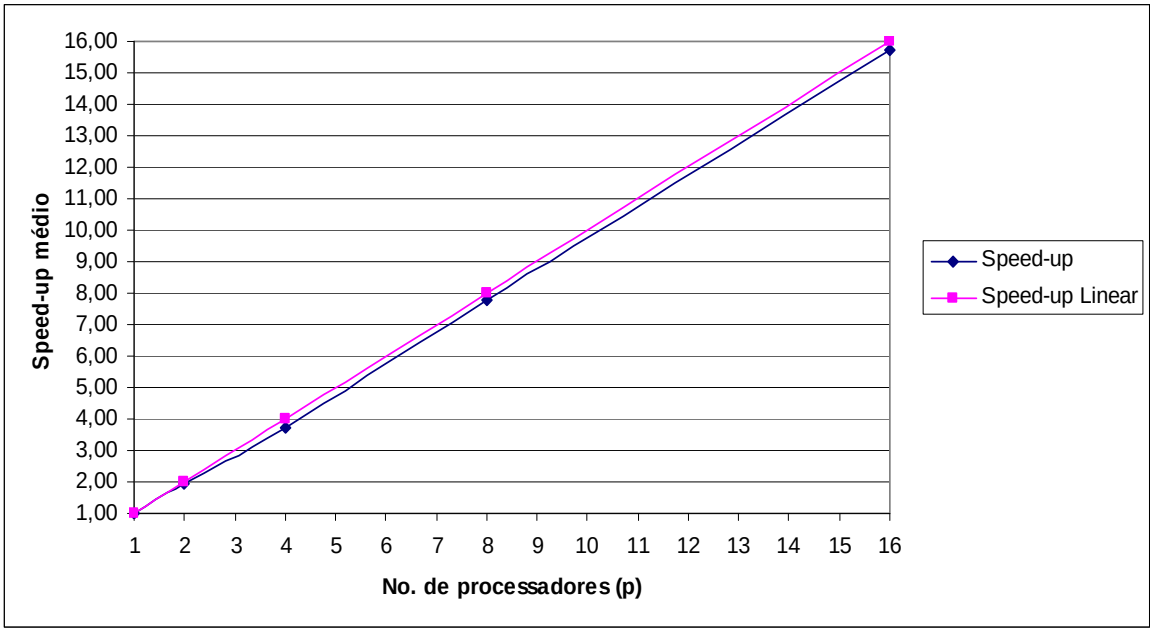


Figura 33: *Speed-up* dos testes apresentados na Tabela 15 para o conjunto de instâncias com *P*=250.

Das análises dos resultados apresentados na Tabela 13, Tabela 14 e Tabela 15, como também da Figura 31, Figura 32 e Figura 33, tem-se as considerações a seguir. Para o primeiro e menor conjunto de instâncias com *P*=50, GRASP-PRP3 também sempre encontrou as mesmas soluções em relação ao GRASP-PR, independente do número de processadores (Tabela 8). Acredita-se que este comportamento seja semelhante ao apresentado pelo GRASP-PRP2. Isto se devendo ao tamanho relativamente pequeno das instâncias. Contudo, apresentou uma redução no tempo computacional e um *speed-up* próximo do linear (Figura 31) e entre o apresentado por GRASP-PRP1 e GRASP-PRP2. Já para o segundo conjunto de instâncias com *P*=100, GRASP-PRP3 encontrou uma melhor solução em relação ao GRASP-PR com 4 processadores, duas melhores soluções com 8 processadores e 3 melhores soluções com 16 processadores em relação ao GRASP-PR (Tabela 12). Como neste caso se tem instâncias de maior dimensionalidade, a utilização da computação paralela proporcionou uma maior sinergia entre os componentes do método, propiciando a obtenção de soluções melhores que no caso seqüencial. Neste caso, também se verifica redução do tempo computacional e um *speed-up* próximo do linear e novamente apresentando *speed-up* entre o do GRASP-PRP1 e do GRASP-PRP2 (Figura 32). Finalmente, para o terceiro conjunto de instâncias com *P*=250, GRASP-PRP3 encontrou duas melhores soluções que GRASP-PR com 4 processadores, três melhores soluções que GRASP-PR

com 8 processadores e cinco melhores soluções que GRASP-PR com 16 processadores (Tabela 13). Para este conjunto de instâncias de maior dimensionalidade, GRASP-PRP3 apresentou resultados muito bons, com bom desempenho em relação ao GRASP-PR, pois devido o uso do paralelismo, aumentou-se a possibilidade de explorar melhor o espaço de busca de soluções. Mantendo o comportamento já apresentado nos conjuntos de instâncias menores (P=50 e P=100), GRASP-PRP3 apresentou redução no tempo computacional com o acréscimo de processadores e *speed-up* próximo do linear e mais uma vez entre os apresentados por GRASP-PRP1 e GRASP-PRP2 (Figura 33).

Da análise dos resultados apresentados por GRASP-PRP3, tem-se as seguintes conclusões. A utilização da computação paralela em máquina híbrida, que em pouco tempo será padrão de mercado, além de bastante escalável, permitiu a redução do tempo computacional em todos os conjuntos de instâncias, proporcionando a obtenção de soluções ainda melhores que no caso seqüencial, principalmente nas instâncias de maior dimensão (P=100 e P=250). Observou-se também, que GRASP-PRP3 obteve um *speed-up* próximo do linear para todos os conjuntos de instâncias testadas e entre os obtidos por GRASP-PRP1 e GRASP-PRP2. Isto sugere que a implementação paralela fez uso dos recursos de comunicação de alta-velocidade disponíveis, o que diminuiu a latência na comunicação e conseqüentemente aumentou o desempenho quando comparado ao GRASP-PRP2.

4.3.4 Análise Geral dos Resultados Obtidos pelas Implementações Paralelas do GRASP com *Path-Relinking*

Nesta seção será apresentada uma análise geral dos resultados computacionais obtidos pelas três implementações paralelas do GRASP com *path-relinking* denominadas GRASP-PRP1, GRASP-PRP2 e GRASP-PRP3, cujos experimentos foram realizados respectivamente nas seções 4.3.1, 4.3.2 e 4.3.3.

A Tabela 16 mostra os resultados computacionais das heurísticas GRASP, com as médias obtidas sobre os três conjuntos de instâncias de maior dimensionalidade (*P=50*, *P=100* e *P=250*). Nesta tabela é realizada uma apresentação mais densa dos dados de modo a facilitar a análise de desempenho das mesmas. Na coluna 1 da tabela,

são apresentados os nomes de cada versão da heurística GRASP; a coluna 2 possui os valores das razões de Steiner (ρ) médio; na coluna 3 são apresentadas as porcentagens de instâncias (Instâncias (%)) em que cada algoritmo encontrou a melhor razão de Steiner (ρ) em relação a GRASP-PR; já na coluna 4 são mostradas as porcentagens em que somente aquele algoritmo (Instâncias* (%)) encontrou a melhor razão de Steiner (ρ) em relação a GRASP-PR; por fim, a coluna 5 apresenta o tempo médio das execuções em segundos de CPU com melhor NMSE nos três conjuntos de instâncias. Para corroborar as análises, a Figura 34 apresenta o desempenho do GRASP-PRP2 versus o desempenho do GRASP-PRP3.

Tabela 16: Comparação entre as versões da heurística GRASP com *path-relinking*.

Versão	ρ Médio	Instâncias (%)	Instâncias* (%)	Tempo Médio (s)
GRASP-PR	0,946589	----	----	66,63
GRASP-PRP1	0,946589	70,43%	0%	17,42
GRASP-PRP3	0,942969	89,61%	13,33%	2,33
GRASP-PRP3	0,942635	93,58%	17,77%	2,19

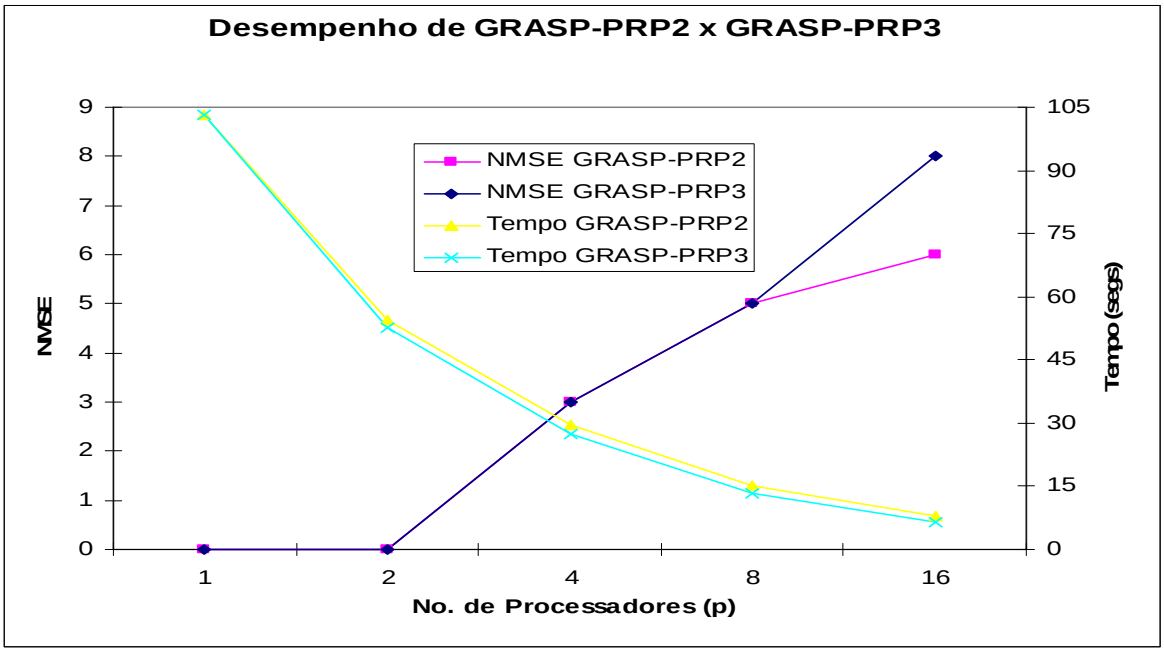


Figura 34: Desempenho de GRASP-PRP2 versus GRASP-PRP3.

Da análise geral dos resultados computacionais obtidos pelas implementações paralelas do GRASP com *path-relinking* apresentados nas seções 4.3.1, 4.3.2 e 4.3.3, como também dos dados da Tabela 16 e da Figura 34 é possível chegar às seguintes conclusões:

- a implementação paralela trouxe real ganho de desempenho em todas as versões propostas em relação à versão seqüencial, como pode ser facilmente observado na Tabela 16, tomando-se como referência o tempo médio em segundos (quinta coluna da Tabela 16) do GRASP-PR seqüencial;
- como pode ser visto na Figura 34, o acréscimo de processadores trouxe benefícios ao desempenho das versões GRASP-PRP2 e GRASP-PRP3 não só no tocante a redução do tempo computacional, como também na qualidade das soluções obtidas;
- as implementações paralelas apresentaram *speed-up* quase linear em todos os casos;
- a versão para a máquina híbrida – GRASP-PRP3 – apresentou o melhor desempenho nos testes realizados;
- máquinas paralelas híbridas são muito escaláveis e permitiu à heurística ali executada apresentar o melhor desempenho, tanto no tocante a velocidade computacional, quanto na qualidade da solução, para a heurística GRASP-PRP3. A heurística GRASP-PRP3 também encontrou o maior número de melhores soluções (NMSE) das três versões implementadas, o que pode ser corroborado pela quarta coluna da Tabela 16, indicando que obteve o maior percentual de melhores soluções sozinho; e
- não obstante, as implementações paralelas GRASP-PRP1 e GRASP-PRP2 mostraram-se com bom desempenho e apresentando-se como alternativas viáveis para quem dispõem dos respectivos tipos de máquinas utilizadas.

4.4 Resultados para o Time Assíncrono

Neste ponto do trabalho, serão apresentados os resultados computacionais obtidos pelo time assíncrono (A-Team) para o problema da árvore de Steiner euclidiana no R^n e comparados aos do GRASP-PRP3 que foi, de acordo com as análises dos resultados computacionais obtidos na seção 4.2, a melhor implementação da heurística GRASP paralela híbrida com *path-relinking*.

Aqui, os testes serão também realizados sobre as mesmas instâncias e sobre as mesmas condições que os realizados para a heurística GRASP híbrida, ou seja, a heurística de construção e as heurísticas de melhoramento (busca local e *path-relinking*) utilizam os mesmos valores dos parâmetros que os utilizados nas versões GRASP apresentadas nas seções anteriores, que podem ser vistos na Tabela 4. De modo a facilitar a comparação e a análise dos resultados, foi dado para o time assíncrono o mesmo tempo de execução que o obtido pelo GRASP-PRP3.

Agora, serão apresentados os valores dos parâmetros específicos do time assíncrono proposto, conforme citados na seção 3.5.2:

- Tamanho da memória de soluções completas e refinadas (*TamMem*):
  - Se $P \leq 50 \rightarrow TamMem = P$; e
  - Se $P > 50 \rightarrow TamMem = P*0.2$ e limitado a 100.
  - P é o número de pontos fixos.

- Tempo total de execução do time assíncrono (*TempTot*):
  - *TempTot* é determinado pelo usuário, sendo neste caso o mesmo tempo total obtido pelo GRASP-PRP3.

- Tempo sem melhora da melhor solução (*TempBest*):

- *TempBest* = *TempTot* * 0.2, ou seja, o valor de *TempBest* é 20% do tempo total de execução determinado para o time assíncrono.

- Percentual de soluções eliminadas quando *TempBest* é atingido (*PercElim*):
  - *PercElim* = 0.8, ou seja, 80% dos piores elementos da memória de soluções completas e refinadas serão eliminadas pelo agente destruidor quando o tempo *TempBest* for atingido.

Na seqüência, na Tabela 17, Tabela 18 e Tabela 19 serão apresentados os resultados obtidos com o time assíncrono para o conjunto de instâncias, comparando-os com os obtidos pelo GRASP-PRP3. A comparação será realizada pelo número de melhores soluções encontradas (NMSE) para os três conjuntos de instâncias (P=50, P=100 e P=250), considerando-se o número de processadores (p) assumindo os valores 1, 2 e 4 e utilizando como tempo total de execução do time assíncrono (*TempTot*) o tempo médio obtido pelo GRASP-PRP3 em cada um dos casos conforme apresentado na Tabela 13, Tabela 14 e Tabela 15. Neste caso, foi considerado o número máximo de quatro processadores para comparação com o GRASP-PRP3, dado que no time assíncrono foram utilizados seis processadores e na prática apenas cinco efetivamente têm agentes que processam (heurística de construção, heurística de melhoria por busca local, heurística de melhoria por *path-relinking*, algoritmo de consenso e o destruidor) e outro é um agente de controle da memória de soluções (mestre).

Tabela 17: Desempenho do A-Team proposto para o conjunto de instâncias *P*=50, variando o tempo de execução considerado.

Algoritmo		No. de processadores (*p*)		
		1	2	4
A-Team	Tempo Médio	12,384	6,393	3,174
	ρ Médio	0,940796	0,940796	0,940796
	Desvio Padrão	0,021645	0,021645	0,021645
	NMSE	0	0	0

Tabela 18: Desempenho do A-Team proposto para o conjunto de instâncias *P*=100, variando o tempo de execução considerado.

Algoritmo		No. de processadores (*p*)		
		1	2	4
A-Team	Tempo Médio	23,996	12,369	6,412
	ρ Médio	0,939641	0,939641	0,937547
	Desvio Padrão	0,021885	0,021885	0,021035
	NMSE	0	0	0

Tabela 19: Desempenho do A-Team proposto para o conjunto de instâncias *P*=250, variando o tempo de execução considerado.

Algoritmo		No. de processadores (*p*)		
		1	2	4
A-Team	Tempo Médio	66,901	33,960	17,803
	ρ Médio	0,959331	0,959331	0,954794
	Desvio Padrão	0,024466	0,024466	0,024227
	NMSE	0	0	0

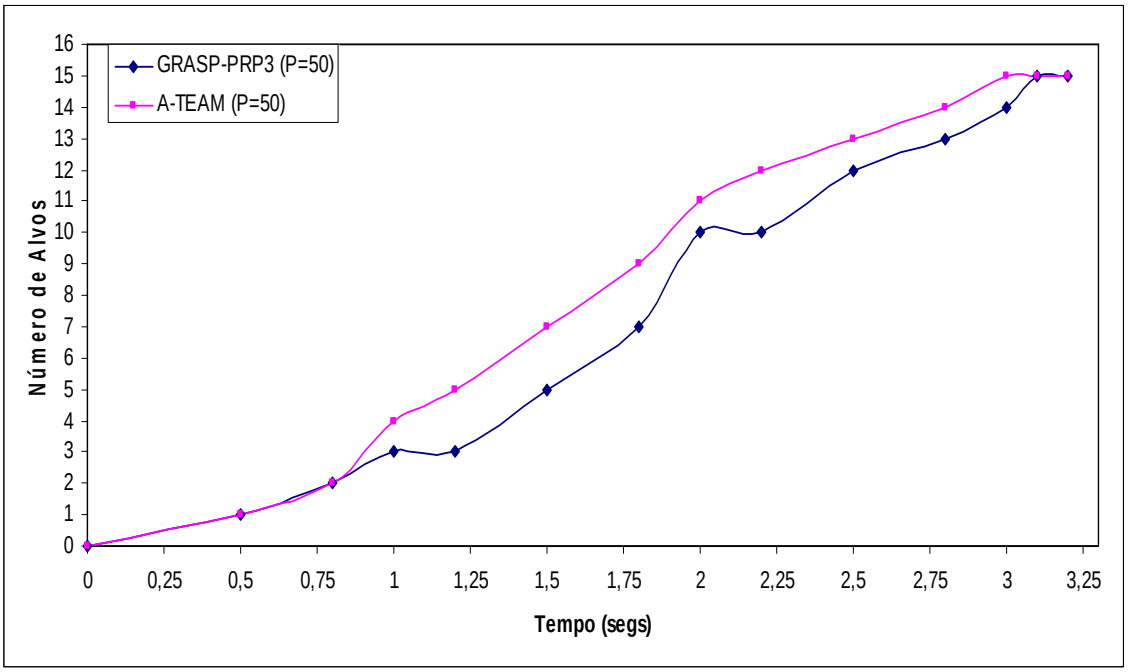


Figura 35: Número de alvos encontrados encontradas pelo A-TEAM e GRASP-PRP3 para o conjunto de instâncias P=50.

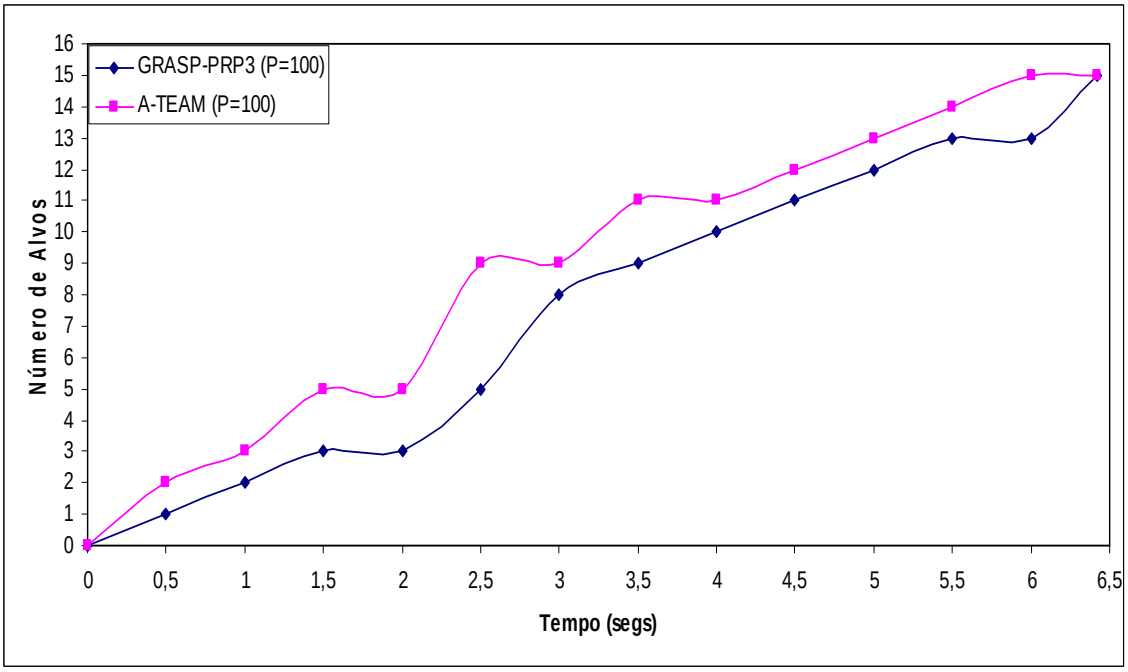


Figura 36: Número de alvos encontrados pelo A-TEAM e GRASP-PRP3 para o conjunto de instâncias P=100.

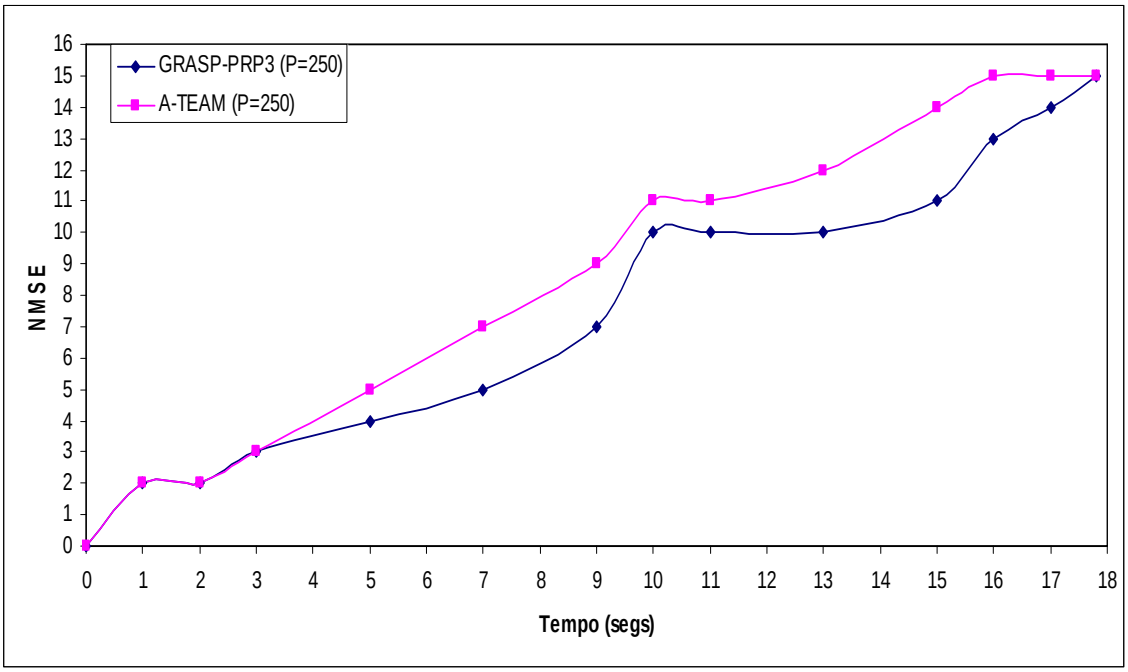


Figura 37: Número de alvos encontrados pelo A-TEAM e GRASP-PRP3 para o conjunto de instâncias P=250.

Dos resultados computacionais obtidos pelo A-Team em relação à heurística híbrida paralela GRASP-PRP3, verifica-se que:

- o time assíncrono, assim como a heurística híbrida GRASP-PRP3 apresentou bom desempenho;
- apesar de não ter encontrado nenhuma melhor solução (NMSE) que o GRASP-PRP3, neste caso, equivaleu-se a ele, pois dados os mesmos tempos médio de execução, encontraram as mesmas soluções. Contudo, o time assíncrono possui melhor comportamento, encontrando todas as 15 soluções alvo para os três conjuntos de instâncias antes (em menor tempo computacional) que o GRASP-PRP3, como pode ser visto na Figura 35, Figura 36 e Figura 37. Nestas figuras é possível ver que o comportamento do A-TEAM é sempre superior, dominando GRASP-PRP3 em todos os experimentos realizados; e
- o time assíncrono, mesmo com um baixo número de agentes (seis – sendo um de controle da memória de soluções, uma heurística de construção, duas heurísticas de melhoria, um algoritmo de consenso e um destruidor) encontrou boas soluções para o problema em questão.

Acredita-se que com o acréscimo de agentes (preferencialmente diferentes aos já desenvolvidos) o desempenho do time possa ficar ainda melhor, dado que é implementado em computador paralelo, com característica distribuída e com comunicação assíncrona.

4.5 Resumo do Capítulo

Neste capítulo foram apresentados os resultados computacionais das técnicas propostas neste trabalho, sendo estas uma heurística GRASP básica seqüencial (GRASP-S), uma heurística GRASP híbrida com *path-relinking* (GRASP-PR), três heurísticas paralelas baseadas em GRASP-PR, sendo uma para máquinas paralelas de memória compartilhada (GRASP-PRP1), outra para máquinas de memória distribuída (GRASP-PRP2) e uma terceira para máquinas híbridas (GRASP-PRP3). Também foi implementado um time assíncrono (A-TEAM) para máquinas de memória distribuída. As especificações das máquinas utilizadas estão na seção 3.4.1.2.

Dos testes realizados sobre as heurísticas GRASP propostas, pode-se verificar que a hibridização realizada com *path-relinking* (GRASP-PR) foi muito benéfica, pois melhorou bastante o desempenho da heurística GRASP. As implementações paralelas da heurística GRASP-PR, em todos os casos, levaram a um ganho considerável no tempo computacional, como também proporcionou ganhos na qualidade das soluções obtidas. Merece destaque que todas tiveram um bom comportamento no ambiente paralelo, fornecendo *speed-up* muito próximo do linear em todos os casos. Dentre as três implementações paralelas, obteve destaque o GRASP-PRP3 que encontrou um maior número de melhores soluções encontradas (NMSE) em relação ao GRASP-PR e em menor tempo de execução que as outras heurísticas híbridas GRASP paralelas propostas (GRASP-PRP1 e GRASP-PRP2). Ressalta-se que a heurística GRASP-PRP1 é muito promissora, dado que desde 2006, a grande maioria dos processadores lançados no mercado são de pelo menos dois núcleos (*multicore*) e a tendência é que o número de núcleos só aumente com o passar do tempo.

Já os testes computacionais realizados com o time assíncrono (A-TEAM) foram muito promissores. O time assíncrono obteve as mesmas soluções alvo que GRASP-PRP3 (melhor das heurística híbrida GRASP paralela propostas neste trabalho), contudo em menor tempo computacional, apresentando assim melhor desempenho e robustez para o problema em questão. Acredita-se que isto se deva a característica sinérgica e assíncrona entre os agentes do time, propiciando maior probabilidade de se encontrar melhores soluções, como pode ser comprovado pelos resultados computacionais obtidos.

Capítulo 5

Conclusões, Contribuições e Trabalhos Futuros

5.1 Conclusões

O objetivo principal do trabalho foi desenvolver e implementar técnicas que permitissem resolver problemas de grande porte e obter a solução ótima ou muito próxima dela em um tempo viável. Desta forma, neste trabalho foram apresentadas três novas abordagens híbridas e paralelas baseada na metaheurística GRASP com *path-relinking* e um time assíncrono para o problema de otimização de grande porte não polinomial conhecido como Problema de Árvore de Steiner Euclidiana (PASE) no R^n. Assim, faz-se necessário o desenvolvimento de métodos eficientes, o que foi conseguido neste trabalho.

Aqui, foram realizados experimentos computacionais e comparações com as melhores técnicas da literatura, onde os testes mostraram que a abordagem proposta inicialmente em uma heurística GRASP básica (GRASP-S) é muito eficiente para obtenção de boas soluções, utilizando muito menos tempo que o algoritmo exato aqui considerado, quando se leva em conta as instâncias de maiores dimensões (que têm interesse prático). Já quando comparando os resultados obtidos pelo GRASP-S com as outras heurísticas da literatura, o mesmo apresentou um maior número médio de acertos (obtenção da solução ótima) para as instâncias consideradas. Ressalta-se que nestes mesmos testes o GRASP-S apresentou um desvio-padrão médio das soluções obtidas compatível com os das técnicas concorrentes e um tempo computacional menor quando comparado às heurísticas da literatura que obtém melhores resultados (AG e μO).

Resultados computacionais adicionais realizados indicam que a hibridização da heurística GRASP com *path-relinking* (GRASP-S mais *path-relinking*), aqui denominado GRASP-PR, propicia uma melhora significativa no desempenho daquele. Neste caso, GRASP-PR obteve em diversas situações, soluções melhores que GRASP-S. Levando-se em consideração a obtenção das soluções alvo, GRASP-PR

também apresentou melhor desempenho no tocante ao tempo de execução. Isto corrobora o conceito de que o processo de hibridização, quando bem feito, proporciona uma melhora no desempenho global do método híbrido desenvolvido, sendo superior a cada uma das partes individualmente.

Também foi proposto neste trabalho de tese a implementação de três versões paralelas da abordagem híbrida (GRASP-PR) com intuito de melhora de desempenho. Estas versões receberam as seguintes denominações: GRASP-PRP1 quando foi utilizada uma máquina de memória compartilhada, GRASP-PRP2 quando foi utilizada uma máquina de memória distribuída e GRASP-PRP3 quando foi utilizada uma máquina híbrida. Foram realizados testes computacionais em uma máquina paralela de larga escala com a utilização de até 16 processadores. Neste caso, conforme confirmado pelos testes computacionais, GRASP-PRP3 encontrou soluções ainda melhores que os obtidos por GRASP-PR e em menor tempo computacional, sendo destacada como a melhor heurística híbrida paralela das três implementadas. Também foi verificado que a abordagem proposta se comporta bem em ambientes paralelos, pois apresentou um *speed-up* praticamente linear para os conjuntos de instâncias de testes utilizados.

Os resultados obtidos pelo time assíncrono também foram bastante promissores, sendo comparáveis ao obtidos pelo GRASP-PRP3 nas condições estabelecidas. Isto comprova que o time assíncrono implementado é bastante eficiente e robusto, pois acredita-se que com o acréscimo de agentes se possa obter resultados ainda melhores.

Os resultados computacionais obtidos comprovam a robustez e eficiência das abordagens propostas (híbrida e paralela) para um problema de otimização de grande porte, neste caso, o PASE no R^n, por ter encontrado melhores soluções para cada conjunto de problemas um maior número de vezes e em menor tempo computacional que as técnicas competidoras da literatura.

5.2 Contribuições

As principais contribuições desta tese são enumeradas a seguir:

- Desenvolvimento e implementação de cinco abordagens do método GRASP, sendo uma heurística GRASP convencional, outra heurística híbrida, fazendo uso de GRASP e *path-relinking*, e três heurísticas híbridas (GRASP e *path-relinking*) paralelas, conforme apresentado no capítulo 3, para um problema de otimização de grande porte, mais especificamente para o Problema de Árvore de Steiner Euclidiano no R^n.

- Desenvolvimento e implementação paralela de um time assíncrono para o PASE no R^n que obteve resultados equiparáveis à melhor heurística GRASP (GRASP-PRP3) também implementada neste trabalho, mostrando-se robusto e eficiente para o problema em questão.

- Comprovação de que abordagens híbridas implementadas de forma a fazer uso de computação paralela, obtêm desempenho satisfatório para problemas de otimização de grande porte.

- Avanço no tamanho dos problemas resolvidos, pela resolução de instâncias de teste bem maiores ($50 \leq P \leq 250$) do que as existentes anteriormente na literatura ($P \leq 18$).

Alguns resultados e contribuições apresentados nesta tese foram publicados nos seguintes trabalhos:

1. ROCHA, M. L., BHAYA, A., "An asynchronous team proposal for the capacitated p-median problem". In: *19Th International Symposium on Mathematical Programming*, 2006, Rio de Janeiro. 19Th International Symposium on Mathematical Programming.
2. ROCHA, M. L., BHAYA, A., "A Parallel Microcanonical Optimization Algorithm for the Euclidean Steiner Tree Problem in Rn". In: *Sixth Optimization 2007 Conference*, Porto, Portugal, 2007.
3. ROCHA, M. L., BHAYA, A., MONTENEGRO, F., MACULAN, N., "A GRASP Heuristic for the Tridimensional Euclidean Steiner Tree Problem". In: *Seventh Metaheuristic International Conference*, Montreal, Canadá, 2007.
4. ROCHA, M. L., BHAYA, A., MONTENEGRO, F., MACULAN, N., "Uma Metaheuristica GRASP para o Problema de Árvore de Steiner Euclidiana Tridimensional". In: *XL SBPO*, João Pessoa, Paraíba, Brasil, 2008.

5. ROCHA, M. L., BHAYA, A., MONTENEGRO, F., MACULAN, N., “Solving the Tridimensional Euclidean Steiner Tree Problem by an Efficient Heuristic”. In: *VI ALIO/EURO Workshop on Applied Combinatorial Optimization*, Buenos Aires, Argentina, 2008.

5.3 Trabalhos futuros

Os resultados obtidos pela heurística GRASP proposta para o PASET são muito promissores. Contudo, estudos maiores e novos desenvolvimentos podem ser feitos de modo a buscar uma eficiência da abordagem híbrida paralela proposta neste trabalho. Como principais sugestões, têm-se:

- Realizar testes computacionais com instâncias de dimensionalidade maior ainda.

- Fazer uso de outras técnicas no processo de hibridização, tais como: ILS (*Iterated Local Search*), VNS (*Variable Neighborhood Search*), entre outras; e posteriormente avaliar o impacto da utilização destas no desempenho da abordagem proposta.

- Implementar uma versão paralela do GRASP utilizando uma estratégia colaborativa de troca de informações e analisar a possibilidade de haver ganho de desempenho.

- Realizar a implementação de mais agentes a serem utilizados no time assíncrono para assim obter soluções ainda melhores que as obtidas até o momento.

- Utilizar no algoritmo de consenso do time assíncrono um percentual que indica o número mínimo de vezes que um valor deve aparecer repetido em determinada posição de todas as soluções para ser considerado comum e aparecer na solução final. Acredita-se que isto proporcione o encontro de padrões de partes similares mais facilmente e de forma menos rígida do que a original (100% de similaridade), aumentando assim as possibilidades de se

encontrar soluções ainda melhores. Deve-se realizar experimentos adicionais para determinar um valor adequado para este percentual de similaridade.

- Realizar implementações tanto das versões paralelas do GRASP quanto do time assíncrono para as grades (*grids*) computacionais, que se crê que seja a nova fronteira da computação de alto-desempenho com o intuito de se conseguir desempenho e escalabilidades ainda maiores [28], [7], [8].

- Fazer uso de técnicas de linearização sobre a formulação matemática proposta em [40] e aplicar técnicas eficientes de programação inteira mista de modo a encontrar bons *lower bounds* para o PASE.

- Realizar testes com instâncias em dimensões maiores ($n > 3$) que as utilizadas neste trabalho e analisar o comportamento das técnicas propostas para as mesmas.

Referências Bibliográficas

[1] AIEX, R. M., *Uma investigação experimental da distribuição de probabilidade do tempo de solução em heurísticas GRASP e sua aplicação na análise de implementações paralelas*. Tese de D.Sc., PUC-Rio, Rio de Janeiro, RJ, Brasil, 2002.

[2] AIEX, R.M., RESENDE, M.G.C. "Parallel strategies for GRASP with path-relinking". In: *Metaheuristics: Progress as Real Problem Solvers*, T. Ibaraki, K. Nonobe and M. Yagiura, (Eds.), Springer, pp. 301-331, 2005.

[3] ALBA, E., Parallel Metaheuristics, a New Class of Algorithms. John Wiley, New Jersey, 2005.

[4] ALFORD, C., BRAZIL, M., LEE, D. H. "Optimization in Underground Mining". In: *Handbook on Operations Research in Natural Resources*, Dordrecht/Boston/London: Kluwer Academic Publisher, 2006.

[5] ALVIM, A. C., *Parallelization strategies for the metaheuristic GRASP*. Dissertação de M.Sc., Pontifícia Univerisade Católica do Rio de Janeiro, Brazil, 1998.

[6] ALVIM, A. C., RIBEIRO, C. C., "Load balancing for the parallelization of the GRASP metaheuristic". In *Proceedings of the X Brazilian Symposium on Computer Architecture*, pages 279–282, Búzios, 1998.

[7] ARAÚJO, A.P.F., BOERES, C., REBELLO, V.E.F., RIBEIRO, C.C., URRUTIA, S., "Exploring grid implementations of parallel cooperative metaheuristics: A case study for the mirrored traveling tournament problem", *Metaheuristics: Progress in Complex Systems Optimization* (K.F. Doerner, M. Gendreau, P. Greistorfer, W. Gutjahr, R.F. Hartl e M. Reimann, editores), 297-322, Springer, 2007.

[8] ARAÚJO, A.P.F., BOERES, C., REBELLO, V.E.F., RIBEIRO, C.C., URRUTIA, S. "Towards grid implementations of metaheuristics for hard combinatorial

optimization problems", *Proceedings of the 17th International Symposium on Computer Architecture and High Performance Computing* (SBAC-PAD 2005), 19-26, 2005.

[9] BARAN, B., KASZKUREWICZ, E., BHAYA, A. "Parallel Asynchonous Team Algorithms: Convergence and performance analysis". *IEEE transactions on parallel and distributed systems*, Vol. 5, No. 7, pp. 677-688, 1996.

[10] BLUM, C., ROLI, A., "Metaheuristics in combinatorial optimization: Overview and conceptual comparision". *ACM Computing Survey,* Vol. 35, No. 3, pp. 268-308, 2003.

[11] CAVALCANTE, V. F., *Times Assíncronos para o Job Shop Scheduling Problem: heurísticas de construção*. Tese de M.Sc. UMECC – UNICAMP, Campinas, SP, Brasil, 1995.

[12] CHAPEAU-BLONDEAU, F., JANEZ, F., FERRIER, J.-L. "A Dynamic Adaptive Relaxation Scheme Applied to the Euclidean Steiner Minimal Tree Problem". *SIAM Journal of Optimization*, 7:1037-1053, 1997.

[13] CHAPMAN, B., JOST, G., VAN DER PAS, R. *Using OpenMP – Portable Shared Memory Parallel Programming*. MIT Press, 2007.

[14] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., *Algoritmo: teoria e prática*. Rio de Janeiro: Campus, 2002.

[15] COURANT, R., ROBBINS , H. *What is Mathematics?* Oxford University Press, New York, 1941.

[16] CRAINIC T.G., TOULOUSE, M., "Parallel Strategies for Meta-heuristics", *State-of-the-Art Handbook in Metaheuristics*, F. Glover, G. Kochenberger (Eds.), 475-513, Kluwer Academic Publishers, Norwell, MA, 2003.

[17] CRAINIC T.G., NOURREDINE, H., "Parallel Meta-Heuristics Applications". In: *Parallel Metaheuristics*, E. Alba (Ed.), John Wiley & Sons, 2005.

[18] CULLER, D., SINGH, J. P., GUPTA, A. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1998.

[19] CUNG, V.-D., MARTINS, S.L., RIBEIRO, C.C., ROUCAIROL, C. “Strategies for the Parallel Implemetation of Metaheuristics”, *Essays and Surveys in Metaheuristics*, C.C. Ribeiro, P. Hansen (Eds.), 263-308, Kluwer Academic Publishers, Norwell, MA, 2002.

[20] DANESH, A., *Dominando o Linux: a bíblia*. São Paulo: Makron Books, 2000.

[21] DANTZIG, G. B., THAPPA, M. N., *Linear Programming 1: Introduction.* Springer, 1997.

[22] EKISOGLU, S. D., PARDALOS, P. M., RESENDE, M. G. C. “Parallel metaheuristics for combinatorial optimization”. In: *Models for Parallel and Distributed Computation - Theory, Algorithmic Techniques and Applications*, R. Correa et al., Eds., Kluwer Academic Publishers, pp. 179-206, 2002.`

[23] FAMPA, M. H. C., ANSTRICHER, K. “An improved algorithm for computing Steiner minimal trees in Euclidean d-space”, *Discrete Optimization* 5(2), pp. 530-540, 2008.

[24] FERREIRA, A. and PARDALOS, P. M. SCOOP. “Solving Combinatorial Optimization Problems in Parallel”. In: A. Ferreira and P. Pardalos, editors, *Solving Combinatorial Optimization Problems in Parallel*, volume 1054 of LNCS State-of-the-Art Surveys, pages 1-6. Springer-Verlag, 1996.

[25] FESTA, P., RESENDE, M.G.C. “Hybrid GRASP heuristics. In "Global optimization: Theoretical foundations and applications”, A. Abraham, A.-E. Hassanien, and P. Siarry (Eds.), *Studies in Computational Intelligence*, Springer-Verlag, 2008.

[26] FLYNN, M. J., “Very high-speed computing systems”, *Proceedings of the IEEE 54,* p. 1901–1909, 1966.

[27] FOSTER, I. T. Designing and building parallel programs: concepts and tools for parallel software engineering. New York: Addison-Wesley, 1995.

[28] FUJISAWA, K., KOJIMA, M., TAKEDA, A., YAMASHITA, M., "Solving Large Scale Optimization Problems via Grid and Cluster Computing", *Journal of Operations Research Society of Japan* Vol.47 (4) 265-274, 2004.

[29] GAREY, M. R.; GRAHAM, R. L., JOHNSON, D. S. "The Complexity of Computing Steiner Minimal Tree". *SIAM Journal of Applied Mathematics*, v. 32, pp. 835-859, 1977.

[30] GAREY, M. R., JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman, San Francisco, 1979.

[31] GLOVER, F. "Tabu search and adaptive memory programing – Advances, applications and challenges". In: Barr, R.; Helgason, R.; Kennington, J., editors, *Interfaces in Computer Science and Operations Research*, p. 1-75. Kluwer, 1996.

[32] GLOVER, F.; LAGUNA, M.; MARTÍ, R. "Fundamentals of scatter-search and path-relinking". *Control and Cybernetics*, 39:635-684, 2000.

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

[34] HAGER, W.W., HEARN, D.W., PARDALOS, P.M. *Large Scale Optimization, State of the Art*. Kluwer Academic Publishers, 1994.

[35] HENNESSY, J. L., PATTERSON, D. A., *Computer Architecture: A Quantitative Approach*.Morgan Kaufmann, 4th Ed., 2006.

[36] HOPCROF J. E., ULLMAN J. D., MOTWANI R., *Introdução à Teoria de Autômatos Linguagens e Computação,* 2ª edição, Rio de Janeiro, editora Campus, 2003.

[37] LAGUNA, M., MARTI, R., "Grasp and Path Relinking for 2-Layer Straight Line Crossing Minimization". *INFORMS Journal on Computing*, v.11 n.1, p.44-52, 1999.

[38] LINHARES, A. e TORREÃO, J. R. A. "Microcanonical Optimization Applied to the Traveling Salesman Problem". *International Journal of Modern Physics*, v. 9, pp. 133-146, 1998.

[39] MACGREGOR SMITH, J. WEISS, R. e PATEL, M. "An O(N^2) heuristic for Steiner minimal trees in E^3", *Networks*, v. 25, pp. 273-289, 1995.

[40] MACULAN, N.; MICHELON, P., XAVIER, A.E. "The Euclidean Steiner Tree Problem in R^n". *Annal of Operations Research*, v. 96, pp. 209-220, 2000.

[41] MACULAN FILHO, N., FAMPA, M. H. C., *Otimização Linear*. Brasília, DF: Editora UnB, 2006.

[42] MARTINS, S.L., RESENDE, M.G.C., RIBEIRO, C.C., and PARDALOS, P.M. "A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy". J. of Global Optimization, vol. 17, pp. 267-283, 2000.

[43] MATTHEW, N., STONES, R., *Professional Linux: Programando*. São Paulo: Makron Books, 2002.

[44] MONTENEGRO, F.; MACULAN, N., TORREÃO, J. R. A. "Microcanonical Optimization Applied to the Euclidean Steiner Problem in R^n". In: *MIC'2001 - 4th Metaheuristics International Conference*, Porto, Portugal, v. 2. p. 515-519, 2001.

[45] MONTENEGRO F., MACULAN, N. "A genetic Algorithm for the Euclidean Steiner Problem in R^n". *Proc X. Cong. Ibero-Latinoamericano Inv. Oper.*, A241, 2000.

[46] MONTENEGRO, F., TORREÃO, J. R. A., MACULAN, N. "Microcanonical Optimization Algorithm for the Euclidian Steiner Problem in R(n) with application to Phylogenetic Inference". *Physical Review E - Statistical Physics, Plasmas, Fluids and Related Interdisciplinary Topics*, v. 68, n. 5, p. 56702, 2003.

[47] MONTENEGRO, F. M. T. *Heurísticas para o Problema de Steiner Euclidiano em R^n*. Tese de D.Sc., COPPE-UFRJ, Rio de Janeiro, Brasil, 2001.

[48] MONTENEGRO, F., MACULAN, N., PLATEAU, G., BOUCHER, P. "New heuristics for the Euclidean Steiner problem in R^n". In**:** *Essays and Surveys in Metaheuristics* [edited by C.C. Ribeiro and P. Hansen], 509-524, Kluwer, 2001.

[49] NACAD – *Núcleo de Computação de Alto Desempenho*. URL: www.nacad.ufrj.br.

[50] NASCIMENTO, H. A. D., LONGO, H., CARVALHO, R. P., MARTINS, W. S., "Modelos de Implementação Paralela de Times Assíncronos". In: *I Workshop em Paralelismo e Otimização Combinatória*, 1998, Búzios - RJ, 1998.

[51] NEMHAUSER, G. L., WOLSEY, L. A., *Integer and Combinatorial Optimization*. Willey, New York, 1988.

[52] OLIVEIRA, N. V., *O Problema de Steiner e a Estrutura das Biomacromoléculas*. Tese de D.Sc., COPPE-UFRJ, Rio de Janeiro, Brasil, 2005.

[53] *OpenMP: A proposed Industry Standard API for Shared Memory Programming*. White Paper, October 1997. URL: http://www.openmp.org/openmp/mp-documents/paper/paper.html.

[54] PAPADIMITRIOU, C. H. AND STEIGLITZ, K. *Combinatorial Optimization-Algorithms and Complexity*. Dover Publications, Inc., New York, 1982.

[55] PEIXOTO, H. P., Metodologia de Especificação do Time Assíncrono Para Problemas de Otimização Combinatória. Dissertação de M.Sc., UNICAMP, Campinas, SP, Brasil, 1995.

[56] PEIXOTO, H. P., SOUZA, P. S., "Times Assíncronos: Uma Nova Técnica para o Flow Shop Problem". *Seminário Integrado de Software e Hardware* (SEMISH), Caxambu –MG– Brasil, agosto de 1994.

[57] Pentium III 1Ghz - Benchmarks. Sitio: http://www.activewin.com/reviews/hardware/processors/intel/p417ghz/bench.shtm. Acessado em Agosto de 2006.

[58] PEREIRA, M. Proposta e Avaliação para o Problema de Steiner Geométrico em Duas e Três Dimensões. Dissertação de M. Sc., UFPE, 1998. O texto e os

dados de entrada estão disponibilizados em www.di.ufpe.br/posgraduacao/antigo/mestrado/mtp98/index.html

[59] PITANGA, M., *Construindo Supercomputadores com Linux*. Rio de Janeiro: Brasport, 2002.

[60] PRIM, R. C. "Shortest Connection Networks and Some Generalizations". *Bell Systems. Technical Journal*, 36:1389-1401, 1957.

[61] RAIDL, G. R., "A unified view on hybrid metaheuristics". In F. Almeida et al., editors, *Proceedings of the Hybrid Metaheuristics Workshop*, volume 4030 of *LNCS*, pages 1-12. Springer, 2006.

[62] RAVADA, S. e SHERMAN, A. T. "Experimental Evaluation of a Partitioning Algorithm for the Steiner Tree Problem in R^2 and R^3", *Networks*, v. 24, pp. 409-415, 1994.

[63] REEVES, C. R., *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publishing, Oxford, England, 1993.

[64] RESENDE, M. G. C., FEO, T. A. "Greedy Randomized Adaptive Search Procedures", *Journal of Global Optimization*, v. 6, pg. 109-133, 1995.

[65] RESENDE, M. G. C., RIBEIRO, C. C. "Greedy randomized adaptive search procedures", In: *Handbook of Metaheuristics*, Kluwer Academic Publishers, pp. 219-249, 2003.

[66] RESENDE, M.G.C., RIBEIRO, C.C. "GRASP with path-relinking: Recent advances and applications,". In *Metaheuristics: Progress as Real Problem Solvers*, T. Ibaraki, K. Nonobe and M. Yagiura, (Eds.), Springer, pp. 29-63, 2005.

[67] RESENDE, M. G. C., PARDALOS, P. M., *Handbook of Applied Optimization*. Edited by P. M. Pardalos and M. G. C. Resende, Oxford University Press, 2002.

[68] RESENDE, M.G.C., RIBEIRO, C.C., "Greedy randomized adaptive search procedures: Advances and applications". In: *Handbook of Metaheuristics*, 2nd Edition, J.-Y. Potvin and M. Gendreau (Eds.), Springer, 2008.

[69] RESENDE, M.G.C., "Metaheuristic hybridization with GRASP". *TutORials in Operations Research*, Zhi-Long Chen and S. Raghavan (Eds.), INFORMS, 2008.

[70] RESENDE, M.G.C., RIBEIRO, C. C., "Parallel Greedy Randomized Adaptive Search Procedures". In: *Parallel Metaheuristics: A new class of algorithms*, E. Alba, (Ed.), John Wiley and Sons, pp. 315-346, 2005.

[71] RIGHI, R. da R., KREUTZ, D. L., CERA, M. C., POSSANI, C. M., PASIN, M., NAVAUX, P. O. A., "Infiniband: Alta eficiência no tratamento de grandes volumes de dados e na computação numérica de alto desempenho". In: *XXIV Encontro Nacional de Engenharia de Produção* (ENEGEP), 2005.

[72] ROSSETI, I. C. M., *Estratégias seqüenciais e paralelas de GRASP com reconexão por caminhos para o problema de síntese de redes a 2-caminhos***.** Tese de D.Sc., PUC - Rio, Rio de Janeiro, Brasil, 2003.

[73] SMELYANSKIY, M., SKEDZIELEWSKI, S., DULONG, C. "Parallel computing for large-scale optimization problems: Challenges and solutions". *Intel Technology Journal*, 9(2):151-163, May 2005. ISSN 1535-766X. URL http://developer.intel.com/technology/itj/2005/volume09issue02/art06_parallel_computing/p01_abstract.htm; ftp://download.intel.com/technology/itj/2005/volume09issue02/art06_parallel_computing/vol09_art06.pdf.

[74] SMITH, J., TOPPUR, B. "Euclidean Steiner Minimal Trees, Minimum Energy Configurations, and the Embedding Problem of Weighted Graphs in E^3". *Discrete Applied Matematics*, 71:187—215, 1996.

[75] SMITH, W. D. "How to Find Steiner Minimal Trees in Euclidean d-space". *Algorithmica*, v. 7, n. 2/3, pp.137-177, 1992.

[76] SMITH, J.M. “Steiner Minimal Tree in E^3: Theory, Algorithms and Applications”. In: *Handbook of Combinatorial Optimization*, v.2, pp. 397-470. Kluwer Academic Publishers, 1998.

[77] SNIR, M., OTTO S. T., WALKER, D. W. *MPI: The Complete Reference*. The MIT Press, 1996.

[78] SOUSA, J. P., RESENDE, M. G. C., *Metaheuristics: Computer Decision-Making*. Edited by M. G. C. Resende and J. P. de Sousa with the assitance of A. Viana, Kluwer Academic Publishers, 2004.

[79] SOUZA, P. S., *Asynchronous Organizations of Multi-algorithm Problem*, Ph.D. Thesis Department of Electrical and computer engineering, Carnegie Mellon University, 1993.

[80] SUN ULTRA 30 – JUST THE FACTS. Sitio: www.sunshack.org/data/sh/2.0/infoserver.central/data/syshbk/Systems/U30/documents/u30-jtf.pdf. Acessado em Agosto de 2006.

[81] TALBI, E.-G. “A Taxonomy of Hybrid Metaheuristics”. *Journal of Heuristics* 8, 5, 541–564, 2002.

[82] TALUKDAR, S., BAERENTZEN, L., GOVE, A., de SOUZA, P., “Asynchronous Teams: Cooperation Schemes for Autonomous Agents”. *Journal of Heuristics* (4), 295-321, 1998.

[83] TALUKDAR, S., MURTY, S., AKKIRAJU, R., “Asynchronous teams”. In: *Handbook of Metaheuristics*. Volume 57. Kluwer Academic Publishers, 537–556, 2003.

[84] TALUKDAR, S., SOUZA, P. S. “Scale Efficient Organizations”, *Proceedings of the 1992 IEEE International Conference on Systems, Man and Cybernetics*, Chicago, Illinois, Oct. 18-21, 1992.

[85] THOMPSON, E. A. “The Method of Minimum Evolution”. *Annals of Human Genetics*, vol 36, pp. 333-340, 1973.

[86] TORREÃO, J. R. A., ROE, E. “Microcanonical Optimization Applied to Visual Processing”. *Physical Letters A*, 205:377-382, 1995.

[87] VIRGINIO, R. ; NEGREIROS, M. ; XAVIER, A. E. . Um Algoritmo GRASP Híbrido para o Problema de Localização Capacitada de Custo Fixo. In: XXXVII Simpósio Brasileiro de Pesquisa Operacional, 2005.

[88] WILKINSON, B., ALLEN, M. *Parallel programming: techniques and applications using networked workstations and parallel computers*. 2. ed. New Jersey: Pearson Prentice Hall, 2005.