ALGORITMO PARA LOCALIZAÇÃO EM REDES DE SENSORES SEM FIO

JÚLIO CÉSAR ALVES
Orientador: Geraldo Robson Mateus e Ricardo Martins de Abreu Silva

ALGORITMO PARA LOCALIZAÇÃO EM REDES DE SENSORES SEM FIO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Belo Horizonte
Junho de 2011

César Alves, Júlio
D1234p Algoritmo para Localização em Redes de Sensores Sem Fio / Júlio César Alves. — Belo Horizonte, 2011
xviii, 58 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Geraldo Robson Mateus e Ricardo Martins de Abreu Silva

1. Localização. 2. Redes de Sensores Sem Fio. 3. heurística. I. Título.

CDU 519.6*82.10

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Algoritmo para localização em redes de sensores sem fio

JULIO CESAR ALVES

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. GERALDO ROBSON MATEUS - Orientador
Departamento de Ciência da Computação - UFMG

PROF. RICARDO MARTINS DE ABREU SILVA - Co-orientador
Departamento de Ciência da Computação - UFLA

PROF. ALEXANDRE SALLES DA CUNHA
Departamento de Ciência da Computação - UFMG

PROF. JOSÉ MARCOS SILVA NOGUEIRA
Departamento de Ciência da Computação - UFMG

PROF. LUIZ FILIPE MENEZES VIEIRA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 27 de junho de 2011.

Dedico esse trabalho à minha esposa Fernanda, aos meus pais João Bosco e Marilene, aos meus irmãos Paulo Roberto e Ana Cláudia, e aos meus avós Euclides e Maria e Sebastião (in memorian) e Maria Conceição

Agradecimentos

Agradeço primeiro à Deus por tudo, pois é através da bênção Dele que cheguei até aqui. Agradeço à minha esposa Fernanda por todo o apoio e carinho em todos os momentos, pela compreensão pela distância, e por tudo que batalhamos juntos. Agradeço aos meus pais João Bosco e Marilene por terem sempre incentivado meus estudos, sem medir esforços para isso. Agradeço aos meus irmãos Paulo Roberto e Ana Cláudia pela amizade verdadeira e incentivo constante. Agradeço aos meus amigos Robert e Diego pela amizade e companheirismo, e por todo o aprendizado mútuo durante esse período. Agradeço aos professores Robson e Ricardo por terem me orientado nesse trabalho e por tudo que pude aprender com a experiência deles. Agradeço a todos os professores e colegas do mestrado por todo o aprendizado. Enfim, agradeço a todos aqueles que direta ou indiretamente contribuíram com este trabalho.

Resumo

O conhecimento da localização geográfica dos nós sensores nas Redes de Sensores Sem Fio é uma característica essencial, pois a informação coletada por um nó da rede só é realmente significativa se for conhecida a localização do mesmo. O problema de Localização em Redes de Sensores Sem Fio consiste em encontrar a posição para os nós sensores de uma rede sem a necessidade de instalar qualquer infraestrutura, como GPSs por exemplo, para todos eles. Dentre as diversas abordagens existentes, o presente trabalho trata do problema baseando-se na existência de âncoras (alguns nós da rede que possuem posição correta conhecida) e nas estimativas de distância entre os demais nós da rede e entre esses nós e os âncoras. O problema é NP-Difícil, o que mostra a viabilidade de se utilizar uma heurística para o mesmo.

O presente trabalho propõe uma heurística específica para o problema, utilizando-se abordagem centralizada e baseada nos métodos de trilateração e interseção entre dois círculos. A forma de controle do uso desses métodos através do processo de escolha dos nós de referência, da ordem de escolha dos nós, e do controle de tolerância de erros permitem que se encontre soluções melhores do que geralmente são encontradas por algoritmos baseados nesses métodos. Foram realizados experimentos comparando o algoritmo proposto com os algoritmos publicados por outros quatro autores. Foram analisados diversos conjuntos de instâncias, variando-se o tamanho da rede, a quantidade de nós âncoras, o alcance de rádio e o fator de ruído. Os experimentos realizados mostram que o algoritmo proposto é competitivo, alcançando melhores resultados em várias das instâncias testadas, inclusive na presença de ruído.

Abstract

The knowledge of the geographical location of sensor nodes in Wireless Sensor Networks is an essential feature, because the information collected by a node is only significant if we know the node's location. The Wireless Sensor Network Localization Problem is defined as determine the location off all sensor nodes in a network without need the use infrastructure, such as GPSs, for them. Among the various existing approaches to the problem, this work addresses the problem based on the existence of anchors (some network nodes that know their positions) and estimates distance between the other nodes of the network and among these nodes and anchors. The problem is NP-Hard, therefore we designed a heuristic for it.

This thesis proposes a specific heuristic to the problem, using a centralized approach based on trilaterations and intersections between two circles. The control of these methods through the process of choosing reference nodes, the order of choice for the reference nodes, and the fault tolerance control make it possible to find solutions better than are usually found by algorithms based on these methods. Experiments carried out compared the proposed algorithm with other four algorithms. We analyzed several sets of instances by varying the network size, the number of anchor nodes, the radio range and the noise factor. The results show that the proposed algorithm is competitive, achieving the best results in several of the instances tested, even when noise is considered.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

As Redes de Sensores Sem Fio (RSSFs) podem ser usadas em diversos cenários que necessitem de monitoramento e controle de grandezas que possam ser medidas por sensores. Segundo Estrin et al. [1999], tais grandezas poderiam ser, por exemplo, temperatura, umidade, pressão, composição do solo, movimento de veículos, níveis de ruído, condições de iluminação, presença ou ausência de certos tipos de objetos, etc. O monitoramento dessas e outras grandes permite a existência de aplicações dessa tecnologia em várias áreas, tais como: controle industrial, monitoramento de ambientes, controle de tráfego, segurança, medicina/biologia e militar [Loureiro et al., 2003].

A utilização das RSSFs tem se tornado viável graças aos diversos avanços na área de micro-eletrônica que permitiram uma redução dos custos de produção, maior miniaturização, criação de circuitos com menor consumo de energia e equipamentos de transmissão sem fio de baixa potência [Reghelin, 2007].

Segundo Wymeersch et al. [2009] em muitas aplicações de rede, sejam elas comerciais, militares ou de serviços públicos, o conhecimento da localização geográfica dos nós sensores nas RSSFs é uma característica essencial, pois a informação coletada por um nó da rede só é realmente significativa se conhecida a localização do nó. Redes de sensores usadas para detectar condições ambientais como temperatura ou poluição, por exemplo, requerem o conhecimento da posição de cada sensor. Ainda segundo Wymeersch et al. [2009] o conhecimento da posição dos nós também facilita as interações dos nós com seus vizinhos, permitindo o uso de aplicações de computação pervasiva e redes sociais.

Além disso, outras funcionalidades das RSSFs dependem da localização, tais como: roteamento geográfico, cobertura, rastreamento e detecção de eventos. Tais funcionalidades possibilitam a existência de diversos tipos de aplicações para as RSSFs, dentre as quais podemos destacar: monitoramento de serviços de saúde, rastreamento de ativos, serviços de emergência, detecção de intrusos, rastreamento de força de trabalho (*blue*

force tracking), localização de amigos ou de pontos de referência, etc. [Aspnes et al., 2004; Wymeersch et al., 2009].

Segundo Goldenberg et al. [2005], a localização dos nós sensores pode ser obtida por configuração manual ou por meios infra-estruturados como GPS, por exemplo. A configuração manual exige um esforço muito grande, o que dificulta a implantação da rede, além de tornar muito custosa a possibilidade de modificar a posição dos nós posteriormente. Já a utilização de GPS possui diversas limitações, tais como: alto custo, consumo de energia e a limitação de alcance. Essa limitação de alcance se refere à necessidade dos dispositivos estarem em linha de visada com os satélites. Isso faz com que a solução não funcione corretamente em ambientes internos, subterrâneos, subaquáticos, e em locais com obstáculos (vegetação densa, construções e montanhas, por exemplo).

O problema de localização em RSSFs consiste em encontrar a posição de cada nó da rede sem precisar colocar GPS, ou algum outro tipo de infra-estrutura, em todos eles. O presente trabalho propõe uma heurística específica para esse problema, baseada em trilateração, cuja grande contribuição é a forma de controle e uso desse e de outros métodos de forma a conseguir posicionar todos os nós alcançáveis em qualquer rede com eficiência. São apresentadas comparações com outros trabalhos que demonstram que o algoritmo é competitivo.

O próximo capítulo apresenta os detalhes do problema, com suas classificações e abordagens. O capítulo 3 apresenta os trabalhos relacionados ao problema de localização. No capítulo 4 é apresentado o algoritmo proposto neste trabalho para o problema, cujos experimentos e resultados são discutidos no capítulo 5. Por fim, as conclusões são apresentadas no capítulo 6.

Capítulo 2

O Problema

Diversos estudos têm sido realizados com o objetivo de obter a posição dos nós da rede sem a necessidade de se ter a posição de todos os dispositivos por meios infra-estruturados. Oliveira [2008] apresenta que os sistemas para a solução do problema de localização envolve três componentes distintos:

- *Estimativa de ângulo e distância*: este componente é responsável por estimar a distância e/ou o ângulo entre os pares de nós. Essa informação é usada pelos demais componentes do sistema de localização.

- *Cálculo de posição*: este componente é responsável por calcular a posição dos nós através das informações disponíveis de ângulos e distâncias entre os nós e das posições conhecidas dos nós âncoras.

- *Algoritmo de localização*: Determina como a informação disponível será manipulada de forma a permitir que muitos ou todos os nós na RSSF tenham informação sobre suas posições.

A próxima seção apresenta uma visão geral de como pode ser feita a estimativa de ângulo e de distância nas RSSFs. Em seguida, são apresentadas as técnicas mais utilizadas para o cálculo de posição e localização. Por fim, são apresentadas diferentes modelagens para o problema e sua complexidade.

2.1 Estimativa de ângulo e distância

As estimativas de ângulo e de distância podem ser obtidas através da utilização de características da própria comunicação sem fio. Boukerche et al. [2007] apresentam como essas estimativas podem ser obtidas:

- **RSSI** - *Received Signal Strength Indicator*: Em teoria, e de forma aproximada, pode-se dizer que a potência do sinal é inversamente proporcional ao quadrado da distância, dessa forma, um modelo conhecido de propagação de sinal de rádio pode ser utilizado para converter a potência de sinal em distância. Entretanto, em ambientes reais, este indicador é altamente influenciado por ruídos, obstáculos, e pelo tipo de antena, o que torna o modelo matematicamente complexo. A principal vantagem de usar essa métrica é o baixo custo e a principal desvantagem é a alta imprecisão das estimativas de distância.

- **ToA** - *Time-of-arrival*: Utiliza medidas baseadas em tempo para encontrar a distância entre dois nós, já que a distância entre eles é diretamente proporcional ao tempo que o sinal leva para propagar de um ponto ao outro. A distância pode ser encontrada através do produto da velocidade de propagação do sinal pela diferença entre o tempo de emissão e o de recepção. Uma dificuldade dessa medição é a necessidade de um preciso sincronismo de tempo entre os nós.

- **TDoA** - *Time-difference-of-Arrival*: Essa métrica também é baseada em medição de tempo e pode ter duas abordagens. Uma delas consiste em utilizar a diferença no tempo na qual um único sinal partindo de um único nó chega a três ou mais nós. Tal abordagem é comum em redes celulares e requer sincronização precisa dos nós receptores (no caso são as estações base). Já outra abordagem consiste em utilizar a diferença no tempo na qual vários sinais enviados por um único nó chegam a outro nó. Tal abordagem é mais apropriada para RSSFs, mas o nó precisa ser equipado com hardware extra para ser capaz de enviar dois tipos de sinal simultâneos (com velocidades de propagação diferentes). TDoA fornece estimativas de distância mais precisas, mas requer um maior custo e geralmente é limitada pelo baixo alcance do segundo sinal enviado para a medição.

- **AoA** - *Angle-of-Arrival*: O ângulo de chegada do sinal pode ser medido em relação ao próprio nó, em relação a um compasso eletrônico ou em relação a um segundo sinal recebido pelo nó. A estimativa pode ser feita usando antenas direcionais ou através de vários receptores uniformemente separados (a diferença de tempo de chegada do sinal permite estimar o ângulo). Essa métrica possui imprecisão e tem desvantagens de custo (devido à necessidade de hardware adicional) e de limitação na redução do tamanho do dispositivo (pois é necessária uma distância mínima entre os receptores).

Maiores detalhes sobre essas métricas podem ser encontradas em Boukerche et al. [2007], Wymeersch et al. [2009] e Mao et al. [2007]. As estimativas de distância entre os

nós não são tratadas neste trabalho, sendo consideradas portanto como um parâmetro de entrada para o algoritmo. De qualquer forma, uma característica importante que deve ser considerada, e que é comum a todas as métricas, é a imprecisão das estimativas de distância. Portanto, uma solução para o problema deve considerar este fato.

2.2 Cálculo de posição e localização

Existem diversas abordagens para se obter a localização dos nós nas RSSFs. A literatura apresenta algumas classificações para essas abordagens e elas são apresentadas a seguir.

- **Centralizada x Distribuída**: Na localização centralizada, as posições de todos os nós desconhecidos são determinadas por um processador central. Este processador recebe as posições dos âncoras e as estimativas de distância entre os pares de nós, calcula a posição de cada nó e devolve para a rede. Já na localização distribuída não há nenhum controlador central e cada nó infere a sua própria posição com base nas informações coletadas localmente (é a forma utilizada pelos GPSs, por exemplo). Algoritmos distribuídos tendem a ser mais escaláveis enquanto os centralizados, por possuírem informações globais da rede, têm condições de obter resultados mais precisos [Wymeersch et al., 2009].

- ***Anchor-free* x *Anchor-based***: Os métodos de localização *anchor-based* utilizam as posições conhecidas de alguns nós da rede, chamados âncoras, para calcular as coordenadas globais dos nós. Já na abordagem *anchor-free* não é necessária a existência de nós com posição conhecida, e cada nó cria um sistema de coordenadas local que o posiciona em relação a seus vizinhos [Biswas et al., 2008].

- ***Range-free* x *Range-based***: Os métodos *range-free* usam somente a informação de conectividade, localizando os nós através de métodos locais e/ou técnicas de contagem de *hops*, trazendo geralmente resultados muito imprecisos. Exemplos de trabalhos que utilizam essa abordagem são: [Doherty e El Ghaoui, 2001; Niculescu e Nath, 2001; Shang et al., 2004]. Já os métodos *range-based* utilizam estimativas de distâncias ou de ângulos entre os nós para localizá-los na rede. Geralmente trazem melhores resultados mas exigem mais componentes de hardware, o que aumenta o custo. Existem diversos trabalhos na literatura utilizando essa abordagem e mais informações podem ser encontradas nas próximas seções.

- **Móvel x estacionária**: Existem aplicações para redes de sensores com nós móveis e/ou com nós estacionários e, portanto, existem métodos de localização apropriados para cada cenário [Hu e Evans, 2004].

Já em relação aos métodos que fazem parte de um algoritmo de localização, Boukerche et al. [2007] apresenta vários deles, como: a trilateração, a multilateração, a triangulação, abordagens probabilísticas, *bounding box* e técnicas de otimização matemática. O desempenho do algoritmo dependerá do método escolhido e cada método depende da disponibilidade de algum tipo de informação e do poder de processamento disponível. Abaixo está uma descrição geral sobre cada um dos métodos a partir do que é apresentado por Boukerche et al. [2007].

- **Trilateração**: é o método mais intuitivo e consiste em encontrar a posição de um nó utilizando as posições conhecidas de três nós de referência e a distância estimada para cada um deles (Figura 2.1). A trilateração é utilizada pelos sistemas de GPS, e em várias outras áreas, como localização de robôs, *kinematics*, aeronáutica, cristalografia e computação gráfica [Thomas e Ros, 2005]. O método pode ser entendido como o processo de encontrar a interseção entre três círculos, ou seja, encontrar a solução para o sistema de equações quadráticas abaixo, onde $p_i = (x_i, y_i), i = 1, 2$ são as coordenadas da referência i e d_i a medida de distância para a referência.

$$(x - x_1)^2 + (y - y_1)^2 = d_1^2 \tag{2.1}$$
$$(x - x_2)^2 + (y - y_2)^2 = d_2^2 \tag{2.2}$$
$$(x - x_3)^2 + (y - y_3)^2 = d_3^2 \tag{2.3}$$

  Em aplicações do mundo real como as estimativas de distância possuem imprecisões, a trilateração pode encontrar resultados imprecisos ou ainda não encontrar resultado. Dessa forma, algoritmos baseados em trilateração precisam utilizar técnicas para contornar esses problemas.

- **Multilateração**: Segue o mesmo princípio da trilateração mas faz uso de um maior número de pontos de referência (Figura 2.2). Neste caso o sistema de equações passa a ser sobredeterminado e o que se faz geralmente é linearizar o sistema para resolvê-lo utilizando algum tipo de cálculo comum para esse fim (através do método de mínimos quadrados, por exemplo). Mais detalhes podem ser encontrados em Boukerche et al. [2007].

- ***Bounding Box***: Este método utiliza quadrados ao invés de círculos (como na trilateração) para limitar as possíveis posições de um nó (Figura 2.3). Para cada nó de referência i é definido um *bounding box* como um quadrado cujo centro é a posição do nó i e com lados de tamanho $2d_i$ (onde d_i é a distância estimada). A interseção entre todos os quadrados pode ser facilmente calculada sem a necessidade de operações de ponto flutuante, e a posição do nó é definida como o centro dessa interseção. A vantagem do método é ser mais simples e bem mais rápido de se calcular e a desvantagem é a grande imprecisão do resultado (que pode ser ainda pior ao considerarmos os erros das estimativas das distância).

- **Triangulação**: utiliza as informações de ângulos, podendo também usar informações de distâncias ou não (Figura 2.4). Geralmente o nó estima os ângulos para três nós de referência com posições conhecidas (que formam um triângulo), ou então a estimativa de ângulos para dois nós de referência e de distância para outro. Com essas informações calcula sua posição utilizando relações trigonométricas.

- **Abordagens probabilísticas**: a existência de erros nas estimativas de distâncias levou à utilização de abordagens probabilísticas para o cálculo de posição de um nó. A ideia é que quando um nó desconhecido recebe um pacote de um nó de referência, ele pode estar em qualquer posição ao redor do nó de referência com igual probabilidade. Ao receber a informação de outro nó, as probabilidades são atualizadas. Quando uma nova informação de posição é recebida a partir de outros nós, é possível que o nó desconhecido calcule sua posição. As principais desvantagens dessa abordagem são o alto custo computacional e o espaço necessário para armazenamento das informações.

- **Técnicas de otimização matemática**: Boukerche et al. [2007] apresentam mais detalhes sobre cada um dos métodos anteriores e citam a existência de vários outros métodos encontrados na literatura. Dentre eles, é importante destacar a formulação do problema de localização em RSSFs como um problema de otimização convexa a partir do qual podem ser utilizadas diversas técnicas de otimização matemática, tais como *semidefinite program* (SDP) [Doherty e El Ghaoui, 2001] e *multidimensional scale* (MDS) [Shang et al., 2004].

O presente trabalho propõe um algoritmo baseado em trilateração para o cálculo de posição e localização utilizando uma abordagem centralizada, *anchor-based*, *range-based*, considerando redes estacionárias. Tal algoritmo é apresentado no capítulo 4. Mas antes disso, as próximas seções mostram algumas modelagens aplicadas ao problema e a complexidade do mesmo, além disso, o capítulo 3 apresenta os trabalhos relacionados na literatura.

2.3 Modelagem do problema

A seguir são apresentadas algumas formas de modelar o problema de acordo com o tipo de abordagem utilizada neste trabalho.

2.3.1 Realização de um Grafo

A modelagem do problema de localização em RSSFs tem se baseado em problemas anteriores bem conhecidos na literatura. Um das primeiras modelagens propostas se baseou no problema de realização de um grafo (*Graph Realization*). Alfakih [2000] define a realização de um grafo da seguinte forma: seja G = (V, E, ω) um grafo simples, não-dirigido, incompleto e com peso nas arestas, com um conjunto de vértices V = v_1, v_2, ..., v_n, um conjunto de arestas E $\subset$ V x V com peso não-negativo ω_{ij} para cada aresta (v_i, v_j) $\in$ E. Cada aresta (v_i, v_j) do grafo G possui um tamanho fixo ω_{ij}, e pode ser livremente rotacionada em torno de seus vértices ligantes. A realização de um grafo G no espaço euclidiano $\mathbf{R}^d$ é então um mapeamento dos vértices de G em pontos de $\mathbf{R}^d$ tal que todo par de vértices adjacentes v_i e v_j de G é mapeado em pontos p^i, p^j $\in \mathbf{R}^d$, cuja distância euclidiana é igual ao peso ω_{ij}.

Segundo So e Ye [2007], a realização de um grafo e suas variantes aparecem em aplicações de diversas áreas, tais como: estruturas de proteína, *dimensionality reduction*, *Euclidean ball packing* e, também, em localização em redes de sensores sem fio [Hendrickson, 1995; Alfakih, 2001; Doherty e El Ghaoui, 2001; Savvides et al., 2001; Biswas e Ye, 2004; Biswas et al., 2008].

Para o caso de estruturas de proteína, por exemplo, More e Wu [1997] mostram que através de ressonância magnética é possível obter os limites mínimos e máximos de distância entre alguns pares de átomos das moléculas. Esse problema pode ser entendido

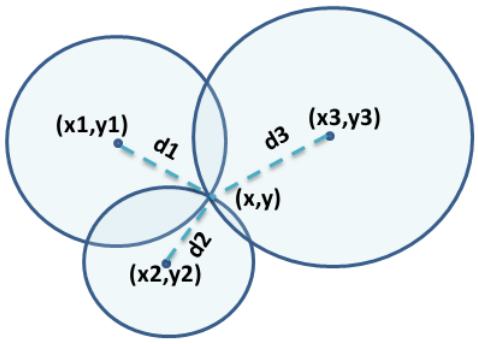

Figura 2.1. Trilateração: consiste em encontrar o ponto de interseção entre três círculos cujos raios são dados pela estimativa de distância do nó para outros três nós de referência com posição conhecida.

então como uma variação da realização de um grafo, na qual os átomos são mapeados como vértices e os limites das distâncias como os limites de pesos das arestas. Esse problema é conhecido na literatura como *distance geometry* e se refere à obtenção das conformações possíveis de uma molécula com base nesses limites de distância entre os átomos e também em características físico/químicas (como quiralidade, por exemplo). Tal estudo é muito importante no processo de descoberta de novas drogas terapêuticas.

Quanto ao problema da localização em RSSFs, So e Ye [2007] o mapeiam como uma variação da realização de um grafo, na qual os vértices do grafo correspondem aos nós sensores, as arestas correspondem aos enlaces de comunicação e os pesos às distâncias entre os nós. Além disso, os vértices são divididos em dois conjuntos, um conjunto representa os nós que têm posições exatas conhecidas (via GPS, por exemplo) e o outro contém os nós cujas posições são desconhecidas. Dessa forma, os autores enxergam que o problema da localização é a realização de um grafo, mas com algumas posições de vértices já pré-determinadas.

Contudo, nas aplicações reais de RSSFs existem outras questões importantes que não estão mapeadas nessa abordagem. Principalmente o fato de que, geralmente, não se tem a distância entre todos os pares de nós e as distâncias conhecidas possuem um erro de medição. A próxima sub-seção apresenta uma modelagem que leva em conta todas essas características.

2.3.2 Otimização Global

Nie [2007] modela o problema de localização em RSSFs como um problema de otimização global cujo objetivo é encontrar a posição de cada nó com o menor erro possível. A modelagem detalhada apresentada pelo autor é dada a seguir.

Para uma sequência de posições (de nós de sensores) $x_1, x_2, ..., x_n$ no espaço euclidiano $\mathbf{R}^d$ $(d = 1, 2, 3)$, o objetivo é encontrar o valor dessas coordenadas, sendo conhecidas: as distâncias (não necessariamente todas) entre estes nós, as distâncias (não necessariamente todas) para outros sensores âncoras $a_1, a_2, ..., a_m$ e as posições

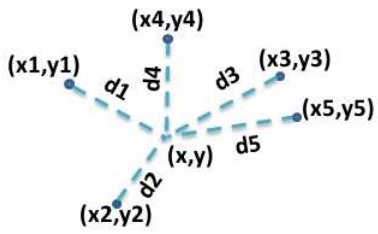

Figura 2.2. Multilateração: possui o mesmo princípio da trilateração, mas são utilizados mais de três nós de referência.

desses sensores âncoras. Seja $A = \{(i,j) \in [n] \times [n] : \|x_i - x_j\|_2 = d_{ij}\}$, e $B = \{(i,k) \in [n] \times [m] : \|x_i - a_k\|_2 = e_{ik}\}$, onde d_{ij}, e e_{ij} são distâncias dadas, $[n] = \{1, 2, ..., n\}$ e $[m] = \{1, 2, ..., m\}$. O problema da localização em RSSFs é então encontrar as posições $\{x_1, x_2, ..., x_n\}$ de acordo com o seguinte sistema de equações:

$$\|x_i - x_j\|_2^2 = d_{ij}^2, (i,j) \in A \tag{2.4}$$

$$\|x_i - a_k\|_2^2 = e_{ik}^2, (i,k) \in B \tag{2.5}$$

Ainda segundo Nie [2007], para uma pequena quantidade de nós, pode ser possível encontrar a localização deles através da solução dessas equações. Contudo, para uma grande quantidade de sensores o custo é muito alto, isso ainda sem contar que as distâncias d_{ij} e e_{ij} possuem erros de medição.

Nie [2007] formula então a localização de sensores como um problema de otimização global, no qual, os valores de $\{x_1, x_2, ..., x_n\}$ seriam coordenadas reais de sensores se, e somente se, o valor ótimo da fórmula abaixo é zero.

$$\min_{x,...,x^n \in \mathbf{R}^d} \sum_{(i,j)\in A} |\|x_i - x_j\|_2^2 - d_{ij}^2| + \sum_{(i,k)\in B} |\|x_i - a_k\|_2^2 - e_{ik}^2| \tag{2.6}$$

Mas na verdade essa formulação é incompleta, pois é possível que se encontre as posições dos nós sensores de forma que a fórmula dê zero e essas posições não correspondam às coordenadas reais dos nós. Um exemplo pode ser visto na Figura 2.5. O nó n só possui estimativa de distância d para o âncora a. Dessa forma, o nó pode ser posicionado em qualquer ponto do círculo de raio d ao redor do âncora a que a contribuição

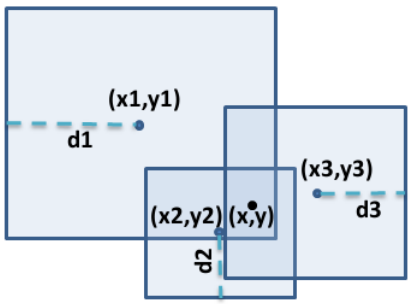

Figura 2.3. *Bounding box*: Consiste em encontrar o ponto médio do retângulo de interseção formado pelos retângulos com centro em cada nó de referência e lado igual a duas vezes a distância entre o nó e a referência. O cálculo é trivial em relação à trilateração, mas a qualidade do resultado é pior.

dessa informação para a formulação acima será zero. O que faltou na modelagem foi considerar que se um nó não tem estimativa de distância para outros nós, significa que eles não podem estar próximos, pois caso contrário eles deveriam ter essa estimativa de distância. Logo, o que se faz nestes casos é considerar que eles devem estar distantes a um valor maior que o alcance do rádio dos nós [Cassioli, 2009]. Na figura mostrada, por exemplo, isso significa que o nó não pode estar nas regiões de interseção entre o círculo de raio d ao redor do âncora a e os círculos que representam o alcance do rádio dos nós para os quais não se tem estimativa de distância. A questão é que não é possível colocar essa restrição na modelagem enquanto não se tem o posicionamento inicial dos nós vizinhos, dessa forma, uma abordagem exata para o problema, por exemplo, precisaria considerar a inclusão de restrições *on-line* no algoritmo, à medida que os nós fossem posicionados.

De qualquer forma, pode-se considerar que o problema de localização é equivalente a encontrar um mínimo global. Diversas propostas de soluções aproximadas foram então publicadas na literatura, como por exemplo, através da modelagem da equação não convexa como *semidefinite programming* (SDP) e *second-order cone programming* (SOCP) através de relaxações do problema original [Nie, 2007].

2.4 Complexidade do problema

Aspnes et al. [2004] apresentam a prova de que o problema geral de localização em RSSFs é NP-difícil. Eles utilizam o conceito de realização de um grafo e também a definição de *Unit Disk Graphs*. Este tipo de grafo é formado a partir de discos de mesmo tamanho posicionados no espaço bidimensional. Os vértices do grafo são dados pelo ponto central desses discos; já as arestas ligam os vértices cujos discos correspondentes tenham interseção.

Eles propõem uma versão de decisão para o problema de localização em RSSFs que consiste em perguntar se um grafo particular com arestas de um determinado

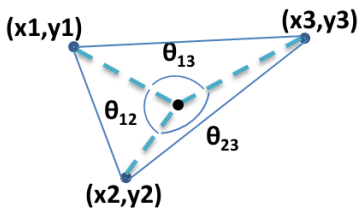

Figura 2.4. *Triangulação*: Consiste em encontrar a posição do nó através de relações trigonométricas tiradas das estimativas de ângulos (e talvez também de distâncias) para os nós vizinhos.

tamanho pode ser realizado em duas dimensões como um *Unit Disk Graph* com discos de determinado tamanho de rádio. A entrada é um grafo G onde cada aresta (u, v) de G é rotulada com um inteiro l_{uv}^2, o quadrado de seu tamanho, junto com um inteiro r^2 que é o quadrado do raio de um *unit disk*. A saída é "sim"ou "não"dependendo se existe um conjunto de pontos em R^2 tal que a distância entre u e v é l_{uv}^2 se (u, v) é uma aresta em G e é maior que r se (u, v) não é uma aresta de G. Aspnes et al. [2004] mostram então que é possível reduzir o problema de satisfabilidade de circuitos a esse problema de decisão, o que prova que o problema de localização em RSSFs é NP-Difícil e mostra a viabilidade de se utilizar uma heurística para o problema.

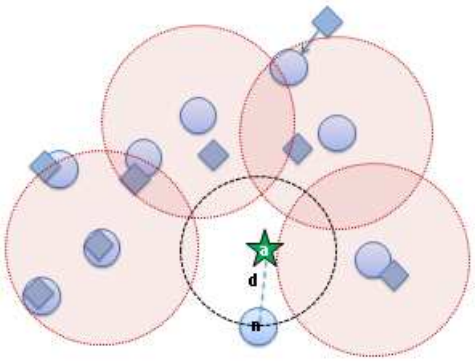

Figura 2.5. Exemplo mostrando que a formulação proposta por Nie [2007] é incompleta: o nó n só possui distância estimada d para o âncora a, logo ele poderia ser posicionado em qualquer ponto da circunferência de centro no âncora e raio igual a d que a parcela de contribuição na formulação seria zero, mas a posição do nó poderá não ser a posição correta.

Capítulo 3

Trabalhos Relacionados

Este capítulo apresenta os principais trabalhos publicados na literatura para o problema de Localização em RSSFs utilizando abordagem centralizada.

Doherty e El Ghaoui [2001] publicaram um trabalho, considerado seminal por Cassioli [2009], no qual eles apresentam uma contextualização completa do problema de localização em RSSFs. Eles propõem um método para estimar a posição dos nós desconhecidos em uma RSSF baseando-se em restrições deduzidas da conectividade entre os nós, através de um modelo conhecido de comunicação par-a-par. Segundo os autores tais restrições são fortes o suficiente para estimar a posição dos nós e simulações mostram que tais posições estimadas são próximas às posições reais dos nós. Eles utilizam *Second-order Cone Problem* (um caso mais geral de SDP) tanto para resolver o problema como também para encontrar limites mínimos e máximos para cada coordenada (fazendo uso também de programação linear para isso). Eles também detalham diversas aproximações do mundo real que os algoritmos de localização em RSSFs geralmente trazem, como por exemplo, o uso de forma geométrica circular para representar o alcance do rádio e a consideração de que dois nós que não possuam distância estimada entre si estão mais distantes que o alcance de rádio (na prática os nós poderiam não ter comunicado por ter alguma barreira física que impediu a comunicação, por exemplo).

Biswas e Ye [2004] introduzem um método de relaxação SDP e encontram melhores resultados que os apresentados por Doherty e El Ghaoui [2001]. A solução apresentada por eles consegue resolver problemas com poucos nós de forma muito eficiente, mas como o número de restrições no modelo SDP é da ordem de $O(n^2)$, onde n é o número de nós na rede, o método não funciona adequadamente para redes com uma grande quantidade de nós.

Para resolver o problema de escalabilidade Biswas e Ye [2006] propõem um esquema de decomposição, no qual os âncoras são particionados em grupos de acordo com suas posições e os sensores que comunicam diretamente com um dos âncoras são atribuídos

aos grupos correspondentes; já os nós mais distantes são colocados em mais de um grupo. O problema é então resolvido para cada grupo separadamente utilizando a relaxação SDP apresentada em Biswas e Ye [2004] e os nós presentes em mais de um grupo assumem a posição encontrada no grupo que obteve o menor erro estimado. Dessa forma, eles conseguiram resolver o problema em redes de até poucos milhares de nós em pouco tempo utilizando um único processador. Além disso, é óbvio que o tempo de execução poderia ser reduzido ainda mais resolvendo o problema de cada grupo paralelamente em diferentes processadores.

Tseng [2008] utiliza relaxação *second order cone programming* (SOCP) para resolver o problema. Ele apresenta relaxações mais fracas do que as utilizadas até então, mas que por outro lado permitem que se encontre soluções mais rapidamente. Ele utiliza *solvers* de SOCP disponíveis através dos quais consegue resolver o problema para alguns milhares de nós em pouco tempo (mil nós em 330 segundos e dois mil nós em três horas). Mas, por outro lado, o algoritmo possui menor qualidade em relação as taxas de erro encontradas.

Niewiadomska-Szynkiewicz e Marks [2009] apresentam um método baseado em trilateração e meta-heurísticas (*Simulated Annealing* e Algoritmo Genético). Um primeiro posicionamento da rede é feito através de trilateração e em seguida é aplicada uma das meta-heurísticas para refinamento da posição. Uma das formas de minimizar o erro encontrado consiste em substituir os nós de referência da trilateração caso o erro fique acima de um limiar aceitável. Eles comparam os resultados encontrados pelo algoritmo com técnicas de SDP e com outro trabalho que também aplica Simulated Annealing e mostram que alcançam melhores resultados tanto para redes com âncoras bem distribuídos quanto para redes com âncoras concentrados em determinadas regiões da rede.

O trabalho de Carter et al. [2007] é considerado por Cassioli [2009] como o estado da arte para o problema de localização com abordagem centralizada. O algoritmo proposto por eles, o SPASELOC, trabalha com a decomposição do problema em subproblemas que são tratados utilizando-se a formulação de SDP definida por Biswas e Ye [2004] e são eficientemente resolvidos por um *solver* de SDP, o DSDP5 [Benson e Ye, 2008]. Tal abordagem alcança melhores resultados especialmente para redes com poucos âncoras, e permite a implementação paralela dos algoritmos, já que cada subproblema poderia ser resolvido independentemente.

Para a separação dos subproblemas uma quantidade máxima de nós por subproblema é definida empiricamente. Durante a execução do algoritmo ao posicionar um nó desconhecido se sua estimativa de erro for menor que um limiar ele passa a ser tratado também como um âncora. Cada âncora possui um nível de confiabilidade atribuído a

ele, sendo que os âncoras reais da rede possuem nível 1 e quanto menor o nível, mais confiável é o âncora. Dessa forma, os nós ajustados que passaram a ser considerados como âncoras possuem níveis de confiabilidade que dependem de quais âncoras foram utilizados para localizá-los. Nesse contexto, a escolha dos nós para formar os subproblemas é feita considerando-se primeiramente o número de âncoras que eles estão conectados e depois o nível desses âncoras.

Para cada subproblema os nós conectados a pelo menos três âncoras são posicionados utilizando-se o *solver* de SDP. Para os nós conectados a menos de três âncoras, Carter et al. [2007] apresentam experimentos que mostram que o *solver* não resolve bem esses casos. Para posicionar esses nós são utilizadas então rotinas geométricas específicas para cada caso que levam em conta a distância estimada e fazem uso da informação dos âncoras não vizinhos para eliminar posições onde o nó não pode estar.

O algoritmo SPASELOC têm um desempenho melhor do que a abordagem baseada unicamente em SDP tanto em qualidade da solução quanto em tempo (possui comportamento linear em relação ao tamanho da rede, enquanto a SDP possui crescimento exponencial), resolvendo problemas de 4 mil nós em 20 segundos e de 10 mil nós em 1 minuto. Além disso, experimentos mostram que não é necessário mais que 10% de âncoras para que o algoritmo encontre seus melhores resultados. Por fim, na tese que originou o algoritmo [Jin, 2005] os autores apresentam uma versão dinâmica para estimar a posição dos nós em tempo-real, uma versão estendida para considerar o problema em três dimensões, o uso de ambientes distribuídos e um pré-processamento para aplicar o algoritmo em redes que não possuam nós âncoras.

Cassioli [2009] apresenta uma heurística multi-estágio focada no problema de localização para redes com baixo ruído (o trabalho é resultado da tese de Cassioli [2008]). O algoritmo proposto usa um esquema de posicionamento sequencial baseado em trilateração para obter uma primeira configuração da rede. Em seguida, é aplicada uma busca local utilizando o método de gradientes chamado NM-BB [Grippo e Sciandrone, 2002]. O terceiro passo consiste em refinar a solução utilizando o método L-BFGS-B [Zhu et al., 1997]. Por fim são aplicadas técnicas de decomposição *ad hoc*. O autor apresenta experimentos comparando seus resultados com outros trabalhos como Carter et al. [2007] e Tseng [2008].

Várias outras abordagens têm sido aplicadas para resolver o problema de localização em RSSFs. Como exemplo podem ser citados os trabalhos de Nie [2007], que usa relaxações *Sum-of-Squares* (SOS) aplicadas à abordagem SDP, Dang et al. [2010], que modela a diferença entre as distâncias estimadas e atuais dos nós como forças que atraem ou repelem os nós uns dos outros, e Vecchio et al. [2011], que usa um algoritmo evolucionário multi-objetivo que trata concorrentemente a precisão da localização e

certas restrições topológicas deduzidas a partir da conectividade dos nós.

Outros problemas relacionados também têm sido propostos na literatura, como o *minimum cost localization problem* (MCLP), cujo objetivo é localizar todos os nós com o menor número possível de nós âncoras. A justificativa é diminuir os custos da rede com a utilização do número de âncoras. Esse problema é proposto por Huang et al. [2011], mas o algoritmo apresentado por ele ainda não é muito prático pois não considera erros nas medidas de distância e assume que qualquer nó da rede pode ser transformado em um âncora.

O próximo capítulo apresenta o algoritmo proposto neste trabalho para o problema de Localização em RSSFs.

Capítulo 4

Algoritmo proposto

Este capítulo apresenta o algoritmo proposto neste trabalho para o problema de localização em RSSFs. Antes disso, a primeira seção contextualiza qual é o cenário no qual o algoritmo poderia ser aplicado. Em seguida o algoritmo é apresentado.

4.1 Cenário para aplicação do algoritmo ao problema

Nesse trabalho será considerada a utilização de RSSFs homogêneas, ou seja, os nós possuem as mesmas especificações de *hardware* (processador, memória e capacidade de energia), com exceção dos nós âncoras que são equipados com um receptor de GPS (obs: apesar do trabalho considerar nós homogêneos, a abordagem utilizada poderia ser facilmente estendida para considerar nós heterogêneos). Considera-se ainda a existência de um nó *sink*, com maior capacidade de processamento, memória e energia (talvez energizado) e capaz de executar o algoritmo proposto.

Oliveira [2008] descreve a seguinte classificação para os nós no contexto de localização em RSSF:

- *Nós desconhecidos*: São nós da rede que não têm informação sobre sua posição. O objetivo do algoritmo de localização é permitir que esses nós tenham essa informação.

- *Nós ajustados*: São nós que a princípio eram desconhecidos, mas que já têm sua posição obtida pelo algoritmo de localização. A quantidade de nós ajustados e a estimativa de erro de suas posições são, geralmente, os principais parâmetros para determinar a qualidade de um sistema de localização.

- *Nós âncoras*: Também conhecidos como *beacons*, são os nós que não precisam de um sistema de localização para estimar suas posições. Eles têm sua posição através de configuração manual ou por meios externos, tais como GPS. Estes nós formam a base de muitos sistemas de localização para RSSFs.

De forma geral, o problema da localização em RSSFs consiste então em transformar o máximo possível de nós desconhecidos em nós ajustados, dadas as posições dos nós âncoras.

Para a presente proposta, não será considerada a estimativa de ângulos entre os nós. Já a estimativa de distância entre os pares de nós (não necessariamente todos) será considerada como entrada do algoritmo. Portanto, não faz parte do escopo desse trabalho definir como essa informação foi obtida da rede (poderia ser através de RSSI, TDoA, etc.).

O principal objetivo do trabalho consiste em calcular as posições dos nós desconhecidos. E, para isso, será utilizado o nó âncora central que receberá toda a informação disponível sobre as distâncias estimadas entre os nós e as posições dos nós âncoras, e executará o algoritmo proposto para estimar as posições dos nós desconhecidos, transformando-os em nós ajustados.

Será considerado ainda que qualquer nó consegue enviar para os nós âncoras as estimativas de distância para seus vizinhos, seja através de comunicação direta, ou através de comunicação entre nós sensores comuns até alcançar os nós âncoras. Além disso, os nós âncoras conseguem comunicar entre si e enviam todas as informações para o nó âncora central. Após o cálculo das posições, estas são enviadas de volta pela rede até que todos os nós recebam sua posição.

A próxima seção apresenta o algoritmo proposto neste trabalho para a Localização em RSSFs.

4.2 Descrição do Algoritmo

4.2.1 Ideia geral

O presente trabalho propõe uma heurística específica para o problema. Em um primeiro momento, tentou-se a utilização do C-GRASP (Hirsch et al. [2010]), uma variação do GRASP (Feo e Resende [1995]) para problemas de otimização global, como método para resolver o problema, mas tal meta-heurística não se mostrou adequada da forma como o problema foi modelado (veja anexo A). Nessa tentativa havia sido criado um método heurístico de geração de solução inicial e tal método acabou se tornando a base para o algoritmo final proposto nesse trabalho.

O algoritmo consiste em um procedimento iterativo que posiciona um nó da rede por vez, sempre tentando fazê-lo da forma mais confiável possível. Primeiro tenta-se posicionar algum nó por trilateração, se não for possível posicionar nenhum nó por trilateração tenta-se por interseção de dois círculos. Cada nó guarda o total de tentativas frustradas e, caso esse valor chegue a um limite pré-determinado como parâmetro, o nó deixa de ser analisado temporariamente. Por fim, se não for possível posicionar mais nenhum nó por nenhum dos dois métodos, utiliza-se a informação de apenas um vizinho para se posicionar um nó. (Obs: antes da execução do algoritmo, é realizado um pré-processamento que retira os nós, ou conjunto de nós, que não são alcançáveis a partir de nenhum âncora, já que não seria possível localizá-los).

A cada vez que um nó é posicionado, independente do método utilizado, seus vizinhos desconhecidos passam a ter mais uma referência que pode ser utilizada para localizá-los. Com isso, o número de tentativas frustradas dos vizinhos é zerado e aqueles que estavam temporariamente fora da análise voltam a ser analisados. Para aumentar as chances de se encontrar a posição mais confiável possível para um nó, o algoritmo dá preferência para os nós que tenham o maior número de vizinhos com posição definida (sejam eles âncoras ou nós ajustados).

Para instâncias sem ruído (cujas distâncias dadas entre os nós são as distâncias exatas) a trilateração consegue encontrar a posição exata de um nó sempre que tivermos distância para pelo menos três nós de referência com posições conhecidas, e desde que elas tenham um posicionamento favorável ao redor do nó. Mas, além dos nós de referência poderem estar em posições desfavoráveis (serem colineares, por exemplo), as medidas de distância entre os nós possuem erros de medida e, nesse caso, a trilateração retorna um resultado aproximado sendo portanto necessário um procedimento que trate um limite de tolerância de erro para se aceitar o resultado de uma trilateração ou ignorá-lo. O algoritmo faz esse tipo de tratamento como pode ser visto nas próximas seções.

Para os casos nos quais os nós não conseguem se comunicar com três vizinhos ou que, durante o processo de execução do algoritmo, não têm três vizinhos com posições já conhecidas, se eles tiverem dois vizinhos com posições conhecidas podemos utilizar a interseção entre dois círculos. Um detalhe importante é que esse tipo de cálculo pode retornar duas soluções, logo é necessário algum tipo de tratamento para discernir qual das duas soluções é a mais correta no contexto do problema; e isso é descrito em detalhes mais a frente.

As próximas seções detalham o algoritmo proposto.

4.2.2 Algoritmo principal

O algoritmo 1 apresenta o algoritmo geral proposto. Ele espera como parâmetros:

- N: conjunto de nós desconhecidos
- A: conjunto de nós âncoras da rede
- low: limite inferior da área da rede
- up: limite superior da área da rede
- DN: distâncias estimadas entre os pares de nós desconhecidos
- DA: distâncias estimadas entre nós desconhecidos e âncoras
- $f()$: função objetivo que retorna a média de erro das distâncias dos nós
- rr: alcance do rádio
- nf: fator de ruído
- ec: tamanho do conjunto elite
- nc: número de vizinhos a serem avaliados
- t: limiar de erro para as soluções (valor abaixo do qual se considera um erro como zero)
- ft: fator de tolerância de erro em relação ao ruído
- $ftmin$: fator de tolerância de erro em relação à área da rede
- $maxI$: número máximo de tentativas com a mesma tolerância de erro
- ma: número máximo de tentativas frustradas de posicionamento por nó

O algoritmo é executado até que uma condição de parada seja alcançada (número máximo de iterações, tempo de execução, qualidade da solução, etc.), na linha 2. Para cada solução a ser gerada, primeiro define-se a tolerância de erro por nó (ϵ). Ela será sempre o maior valor entre a largura da região da rede divida por $ftmin$ ou o ruído divido por ft (linha 3). Na linha 4, o contador de tentativas com a mesma tolerância i é inicializado com zero, os conjuntos de nós ajustados T e de soluções S são iniciados vazios e conjunto de nós pendentes P é construído. Cada elemento do conjunto S representará uma solução para o problema, ou seja, guardará uma possível posição para cada nó desconhecido. Já o conjunto P representa o conjunto de nós

Algoritmo 1 WSN Localization Algorithm

```
1: procedure LocRSSF(N, A, low, up, DN, DA, f(), rr, nf, ec, t, nc, ft, ftmin, maxI, ma)
2:     while Ending condition not reached do
3:         ε ← FractionOfNoiseFactor(nf, ft, up, low, ftmin)
4:         i ← 0; T ← ∅; S ← ∅; P ← BuildPending(s, ma);
5:         while |P| > 0 do
6:             sensor ← TryTrilateration(s, P, t, ε, nc, ma, rr, nf)
7:             if sensor is not valid then
8:                 sensor ← Try2CircleInt(s, P, rr, nf, t, ε, nc, ma)
9:             end if
10:            if sensor is not valid and (|P| = 0 or JustSensorWithOneRef(P)) then
11:                sensor ← TryNeighborRadius(fpn, s, t, ε, rr, nf); i ← i + 1;
12:                if i ≥ maxI then
13:                    i ← 0; Increase error tolerance ε
14:                    attemps ← 0, for each not located node in N
15:                end if
16:            end if
17:            if sensor is valid then
18:                i ← 0; T ← T ∪ {sensor};
19:                attemps ← 0, for each not located sensor's neighboor
20:                P ← BuildPending(s, ma)
21:                if |P| = 0 then
22:                    attemps ← 0, for each not located node in N
23:                    P ← BuildPending(s, ma)
24:                end if
25:                fpn ← P[1]
26:            end if
27:        end while
28:        UpdateOrdered(S, sol)
29:        BestSol ← CalculateBestSolFrom(S)
30:    end while
31:    return BestSol
32: end procedure
```

desconhecidos, ou seja, aqueles que ainda precisam ser posicionados. Tal conjunto é ordenado pela quantidade de vizinhos com posição definida (âncoras e nós ajustados).

O procedimento de posicionamento é então repetido até que não exista nenhum nó pendente. Em um primeiro momento tenta-se posicionar um nó por trilateração (linha 6). Se for possível posicionar um nó, ele é passado para o conjunto de nós ajustados (linha 18b), o número de tentativas de posicionamento dos seus vizinhos ainda não posicionados é zerado (linha 19) e a lista de nós pendentes é construída novamente (linhas 20 a 24). O primeiro nó da lista é guardado na variável fpn para ser utilizado caso não seja possível posicionar mais nenhum nó pendente (será melhor detalhado mais adiante). Se não for possível posicionar nenhum nó por trilateração, tenta-se por interseção de dois círculos (linha 11). Esse processo é então repetido posicionando-se

os nós um a um. Ao terminar de posicionar todos os nós a solução é inserida em um conjunto ordenado de soluções (linha 18).

A melhor solução encontrada até o momento é calculada e guardada na variável *BestSol* (linha 29) e o cálculo é feito como descrito a seguir. Enquanto o conjunto elite não estiver cheio (ou seja, $|S| < ec$) a melhor solução será sempre a primeira solução do conjunto (uma vez que ele é ordenado). Já se ele estiver cheio e a solução recém-inserida fizer parte do conjunto, são encontradas quatro soluções "médias"considerando todas as soluções do conjunto elite. A posição de cada nó é calculada independentemente, utilizando as posições encontradas para o nó em cada uma das soluções do conjunto.

A figura 4.1 representa como os cálculos são feitos. Na primeira solução média, S_{avg}, a posição de um nó é dada pelo seu ponto médio. Já a segunda, S_{wavg}, é similar à anterior, mas é usada uma média ponderada pelo inverso do erro estimado para cada posição; o objetivo é deixar o nó mais próximo das posições com menor erro. A solução S_{lavg} é similar à primeira mas desconsidera a maior e a menor coordenada de cada eixo, procurando evitar soluções extremas que prejudiquem a média. Por fim, a solução S_{lwavg} é similar à segunda, mas ignora as posições com os 20% piores valores de erro; também é uma forma evitar soluções extremas que prejudiquem a média. É retornada então a melhor solução entre essas quatro e a primeira solução do conjunto elite. À princípio pode-se supor que é possível eleger uma dessas médias como a melhor estratégia de cálculo e adotá-la no algoritmo. Mas os experimentos mostraram que nenhuma delas encontra sempre a melhor solução para o problema. Dessa forma, preferiu-se adotar todas elas escolhendo a solução que encontra a solução com a menor estimativa de erro.

Durante a execução do algoritmo, a cada tentativa frustrada de posicionamento de um nó, seja por trilateração, seja por interseção de 2 círculos, o número de tentativas de posicionamento do nó é incrementado. Se esse número de tentativas atingir um máximo configurado, o nó é retirado da lista de nós pendentes. Isso é feito porque os métodos de trilateração e interseção de dois círculos, possuem características aleatórias, logo duas execuções diferentes dos métodos pode obter resultados diferentes; mas, depois de algumas tentativas, é preferível parar de tentar posicioná-lo até que alguma coisa mude no cenário (ex: um vizinho do nó for posicionado).

Com o processo descrito acima, pode ser que a lista de nós pendentes fique vazia sem que todos os nós tenham sido posicionados. Nesse caso, tenta-se posicionar o nó fpn utilizando a informação da distância para um único vizinho (tal procedimento também é realizado se a lista de nós pendentes não estiver vazia, mas tiver apenas nós que tenham um único vizinho posicionado, ou seja, uma única referência para o posicionamento). Toda vez que é necessário recorrer a esse procedimento, o número de

tentativas com a mesma tolerância i é incrementado (linha 11b). Se esse número atingir o valor $maxI$, a tolerância de erro é então aumentada (linhas 12 e 13). Enquanto a tolerância for menor que o ruído ela aumenta em r/ft, caso contrário ela é aumentada em dez vezes. Por fim, o número de tentativas de cada nó é zerado (linha 14). Dessa forma, o algoritmo voltará a tentar posicionar os nós por trilateração ou interseção de dois círculos, mas agora considerando uma tolerância de erro maior.

Por fim, ao atingir a condição de parada, o algoritmo retorna então a melhor solução encontrada durante todo o processo.

As próximas seções apresentam os métodos internos do algoritmo.

4.2.3 Método para construir a lista de nós pendentes

O algoritmo 2 apresenta o procedimento para construir a lista de nós desconhecidos, ou seja, ainda não posicionados. A ideia é construir uma lista ordenada pelo número de referências (vizinhos posicionados, sejam eles âncoras ou ajustados) de cada nó. Para melhorar o desempenho, ao invés de se fazer uma inserção ordenada, cria-se um vetor onde cada posição tem uma lista de nós cujo número de referências é igual à posição da lista no vetor (nós com dez referências ou mais são colocados em uma única lista).

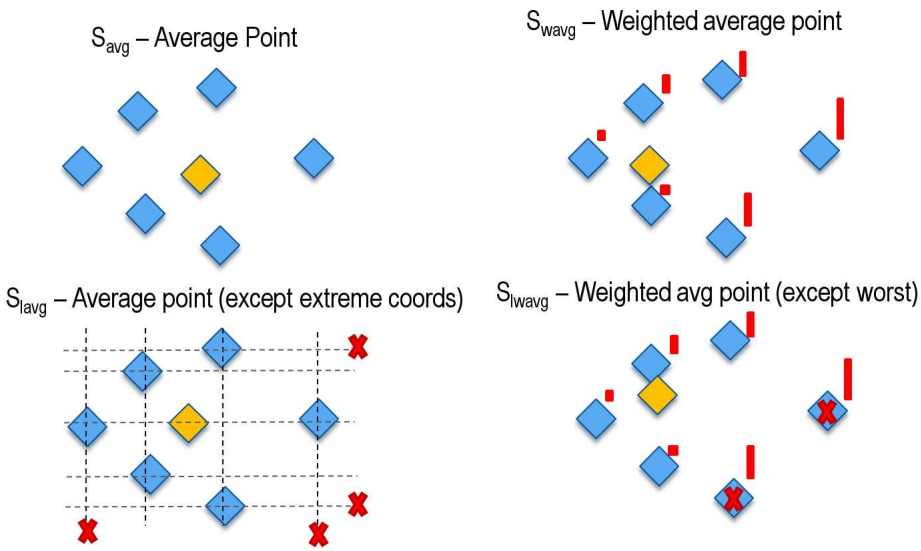

Figura 4.1. Representação dos cálculos de solução "média"utilizados pelo algoritmo. S_{avg}: a posição de um nó é dada pelo ponto médio calculado a partir da posição dada em cada uma das soluções do conjunto elite. S_{wavg}: similar à primeira, mas é usada uma média ponderada pelo inverso do erro estimado para cada posição. S_{lavg} : Similar à primeira mas desconsidera a maior e a menor coordenada de cada eixo. S_{lwavg}: similar à segunda, mas ignora as posições com os 20% piores valores de erro.

Ao inserir um nó na lista a posição de inserção é definida aleatoriamente (linhas 3 a 7).

Por fim, é montada a lista completa concatenando-se as listas do vetor, começando pelos nós com maior número de referências (linhas 8 a 10) e essa lista é retornada pelo método (linha 11).

Algoritmo 2 Procedure to build the list of not positioned nodes

```
 1: procedure BuildPending(N, ma)
 2:     P ← ∅
 3:     for each node n not positioned in N do
 4:         if n is reachable and n.attempts ≤ ma then
 5:             P[n.|RN|].Insert(n, random)
 6:         end if
 7:     end for
 8:     for i ← P.size to 1 do
 9:         P'.Concatenate(P[i])
10:     end for
11:     return P'
12: end procedure
```

4.2.4 Método que tenta posicionar por trilateração

O algoritmo 3 mostra o procedimento para tentar posicionar algum nó por trilateração. O que ele faz é: para cada nó da lista de pendentes, que tenha pelo menos três nós de referência (linha 5), tenta aplicar o método de trilateração (linha 6). Se ele retornar um resultado e o erro for menor que a tolerância (linha 7), posiciona-se o nó (linha 8) e encerra-se o método.

Para cada nó que não seja possível posicionar, seu número de tentativas é incrementado (linha 10) e, caso ele tenha atingido o número máximo permitido, ele é retirado da lista de nós pendentes (linhas 11 a 13). Se não for possível encontrar resultado para nenhum nó, o método retorna que nenhum nó foi posicionado.

O algoritmo 4 apresenta o procedimento utilizado para posicionar os nós por trilateração. O ideal seria escolher o melhor posicionamento utilizando a trilateração com todas as combinações possíveis de vizinhos posicionados do nó combinados três a três; mas isso seria inviável em relação ao tempo de execução do algoritmo. Dessa forma, o que o método faz é escolher aleatoriamente um grupo três dos vizinhos para tentar a primeira trilateração (linhas 2 e 6). Depois, para cada vizinho ainda não utilizado, escolhe-se um deles de forma aleatória (linha 23) para substituir um dos vizinhos do grupo atual (escolhido também de forma aleatória - linha 22) e tenta-se a trilateração novamente (linha 10). Entre todas as trilaterações que encontrarem resultado (linha 11) e que tenham um erro acumulado menor do que a tolerância ϵ (linhas 15 e 16), retorna-se a que tenha o menor valor (linha 27).

Algoritmo 3 Try to position by trilateration

```
1: procedure TryTrilateration(s, P, t, ε, nc, ma, rr, nf)
2:     pn ← null
3:     i ← 1
4:     while pn = null and i ≤ |P| do
5:         if P[i].|RN| ≥ 3 then
6:             [tOK, x, y] ← PosTrilateration(P[i], s, t, ε, nc, rr, nf)
7:             if tOK and CalcNodeError(P[i], s, x, y, rr, nf) < ε then
8:                 pn ← P[i]
9:             else
10:                P[i].attempts ← P[i].attempts + 1
11:                if P[i].attempts > ma then
12:                    P ← P \ P[i]
13:                end if
14:            end if
15:        end if
16:        i ← i + 1
17:    end while
18:    return pn
19: end procedure
```

O método de trilateração utilizado é um procedimento aproximado que utiliza apenas operações aritméticas básicas e raiz quadrada, sem fazer uso de relações trigonométricas. Ele retorna a solução correta se forem passados os raios exatos (sem erro de estimativa) das circunferências; já para os casos com erro de estimativa ele pode conseguir retornar uma solução ou não. Mas, mesmo que retorne, é necessário verificar se o valor encontrado é uma solução razoável ou não e isso é feito na linha 13 verificando-se se as distâncias passadas para o algoritmo batem com as distâncias encontradas a partir da posição retornada. Dessa forma, não se garante que a solução encontrada está correta mas evita-se soluções sabidamente ruins.

4.2.5 Método que tenta posicionar pela interseção entre dois círculos

O algoritmo 5 mostra o procedimento para tentar posicionar algum nó pela interseção entre dois círculos. O que ele faz é: para cada nó da lista de pendentes, na ordem do número de referências, tentar aplicar o método de interseção (linhas 5 a 19). Se ele retornar um resultado e o erro calculado for menor que a tolerância (linha 8), posiciona-se o nó e encerra-se o método.

Assim como na trilateração, para cada nó que não é possível posicionar, seu número de tentativas é incrementado (linha 11) e, caso ele tenha atingido o número máximo permitido, ele é retirado da lista de nós pendentes (linhas 12 a 14). Se não for possível

Algoritmo 4 Position a node by trilateration

```
 1: procedure PosTrilateration(n, s, t, ε, nc, rr, nf)
 2:     PN ← anchor and common neighbors of n already positioned at random order
 3:     PN ← PN[1..nc]
 4:     pnc ← |PN|
 5:     tOK ← false
 6:     minE ← ∞
 7:     TN ← PN[1..3]
 8:     PN ← PN \ {TN}
 9:     u ← 3
10:     while u < pnc do
11:         [tOK, p] ← Trilateration(TN[1].p, TN[1].d, TN[2].p, TN[2].d, TN[3].p, TN[3].d, t)
12:         if tOK then
13:             a ← |||p, TN[1]|| − TN[1].d| + |||p, TN[2]|| − TN[2].d| + |||p, TN[3]|| − TN[3].d|
14:             if a ≤ ε then
15:                 p ← FitBounds(p)
16:                 e ← CalcNodeError(n, s, p.x, p.y, rr, nf)
17:                 if e < minE then
18:                     x ← p.x; y ← p.y
19:                     minE ← e; tOK ← true
20:                 end if
21:             end if
22:         end if
23:         TN ← TN \ {ChooseRandom(TN)}
24:         v ← ChooseRandom(PN)
25:         TN ← TN ∪ {v}; PN ← PN \ {v}
26:         u ← u + 1
27:     end while
28:     return tOK, x, y
29: end procedure
```

encontrar resultado para nenhum nó, o método retorna que nenhum nó foi posicionado.

O posicionamento pela interseção entre dois círculos (algoritmo 6) segue os mesmos princípios do procedimento de trilateração. Como não é viável fazer a interseção entre todos os pares de vizinhos do nó, forma-se um grupo escolhendo-se aleatoriamente dois vizinhos (linha 4), calcula-se a interseção (linha 6) e depois um dos dois é sorteado pra sair do conjunto (linha 48) e outro ainda não usado é sorteado para ser usado no lugar (linha 49). Entre todas as interseções que encontrarem resultado (linha 7) e que tenham um erro acumulado menor do que a tolerância ϵ (linhas 11 a 16), retorna-se a que tenha o melhor valor.

Mas há um tratamento adicional na interseção de dois círculos devido ao fato dela retornar duas posições possíveis, sendo assim necessário escolher entre uma delas. O que se faz é tentar descobrir qual das duas posições não é a correta para eliminá-la. Para isso, primeiro o algoritmo verifica se a distância que o nó ficou para os seus vizinhos

Algoritmo 5 Try to position by 2 circle intersection

```
 1: procedure Try2CircleInt(s, P, rr, nf, t, ε, nc, ma)
 2:     pn ← null
 3:     i ← 1
 4:     n ← P[1]
 5:     while pn = null and i ≤ |P| do
 6:         if n.|RN| ≥ 2 then
 7:             [iOK, x, y] ← Pos2CircleInt(n, s, rr, nf, t, ε, nc)
 8:             if iOK and CalcNodeError(P[i], s, x, y, rr, nf) < ε then
 9:                 pn ← n
10:             else
11:                 P[i].attempts ← P[i].attempts + 1
12:                 if P[i].attempts > ma then
13:                     P ← P \ P[i]
14:                 end if
15:                 i ← i + 1
16:                 n ← P[i]
17:             end if
18:         end if
19:     end while
20:     return pn
21: end procedure
```

tem uma diferença menor que a tolerância; se uma das posições não atende a isso ela é eliminada (linhas 11 a 16). Se as duas atendem a esse quesito, verifica-se então entre os nós não vizinhos, se tem algum que ficou a uma distância menor que o alcance do rádio (considerando o fator de ruído); se isso acontecer para alguma posição, ela é eliminada (linhas 19 a 24). Por fim, se as duas posições ainda são válidas, escolhe-se aquela que retorna o menor erro (linhas 33 a 41).

4.2.6 Método que tenta posicionar pela informação de um único vizinho

O algoritmo 7 apresenta o procedimento para posicionar um nó a partir do raio de um único vizinho. Ele consiste em verificar, para um dado vizinho já posicionado do nó, qual é o ponto na circunferência centrada no vizinho e de raio igual à distância entre o vizinho e o nó, que tem o menor valor de erro. Isso é feito tentando-se posicionar o nó na circunferência de um em um grau. Retorna-se então o posicionamento de menor erro.

Se o nó tiver vizinhos âncoras, um deles é escolhido aleatoriamente para o procedimento; caso contrário, algum dos nós ajustados é escolhido.

Algoritmo 6 Posiciona um nó usando interseção de 2 círculos

```
1: procedure Pos2CircleInt(n, s, rr, nf, t, ϵ, nc)
2:     PN ← anchor and common neighbors of n already positioned at random order
3:     PN ← PN[1..nc]
4:     pnc ← |PN|; iOK ← false; minE ← ∞
5:     u ← 2; TN ← PN[1..2]; PN ← PN \ {TN}
6:     while u < pnc do
7:         [iOK, p1, p2] ← 2CircleInt(TN[1].p, TN[1].d, TN[2].p, TN[2].d)
8:         if iOK then
9:             p1 ← FitBounds(p1); p2 ← FitBounds(p2)
10:            p1Inv ← false; p2Inv ← false
11:            for each positioned neighbor t of n do
12:                if |‖p1, t‖−t.d| > ϵ then
13:                    p1Inv ← true
14:                end if
15:                if |‖p2, t‖−t.d| > ϵ then
16:                    p2Inv ← true
17:                end if
18:            end for
19:            for each node t not neighbor of n do
20:                if ‖p1, t‖< rr ∗ (1 − nf) then
21:                    p1Inv ← true
22:                end if
23:                if ‖p2, t‖< rr ∗ (1 − nf) then
24:                    p2Inv ← true
25:                end if
26:            end for
27:            p ← null
28:            if not p1Inv and p2Inv then
29:                p ← p1
30:            else
31:                if p1Inv and not p2Inv then
32:                    p ← p2
33:                else
34:                    if notp1Inv and not p2Inv then
35:                        e1 ← CalcErrorNode(n, s, p1.x, p1.y, rr, nf)
36:                        e2 ← CalcErrorNode(n, s, p2.x, p2.y, rr, nf)
37:                        if e1 < e2 then
38:                            p ← p1; e ← e1
39:                        else
40:                            p ← p2; e ← e2
41:                        end if
42:                    end if
43:                end if
44:            end if
45:            if p ≠ null and e < minE then
46:                x ← p.x; y ← p.y; minE ← e; iOK ← true
47:            end if
48:        end if
49:        TN ← TN \ {ChooseRandom(TN)}
50:        v ← ChooseRandom(PN); TN ← TN ∪ {v}; PN ← PN \ {v}
51:        u ← u + 1
52:    end while
53:    return iOK, x, y
54: end procedure
```

Algoritmo 7 Try to position by neighbor radius

```
 1: procedure TryNeighborRadius(n, s, t, ϵ, rr, nf)
 2:     r ← a random anchor neighbor of n
 3:     if r = null then
 4:         r ← a random common neighbor of n
 5:     end if
 6:     minE ← ∞
 7:     for i ← 1 to 360 do
 8:         [x, y] ← r.p+ Rotate(r.d, i)
 9:         e ← CalcNodeError(n, s, x, y, rr, nf)
10:         if e < minE then
11:             minE ← e; minX ← x; minY ← y;
12:         end if
13:     end for
14:     n.p ← [minX, minY]
15:     return n
16: end procedure
```

4.2.7 Método de cálculo de erro por nó

A função de cálculo de erro de cada nó é apresentada no algoritmo 8. Basicamente o que se faz é somar: (a) as diferenças entre as distâncias estimadas e as encontradas na rede do nó para todos os seus vizinhos com posição definida (linha 6), (b) os erros encontrados para os nós não vizinhos (linhas 9 a 11). Esse erro é dado pelo tanto que um nó não vizinho está mais próximo do nó que o mínimo para considerá-lo como não vizinho, ou seja, do alcance do rádio considerando o ruído. Por fim, retorna-se esse valor em média, dividindo-se pelo número de vizinhos já posicionados (linha 15).

Algoritmo 8 Calculate the error of a node

```
 1: procedure CalcNodeError(n, s, x, y, rr, nf)
 2:     VS ← {s.A} ∪ {s.N}
 3:     i ← 0; e ← 0
 4:     for each node k in VS do
 5:         if k is neighbor of n then
 6:             e ← |‖k, n‖−D(k, n)|
 7:             i ← i + 1
 8:         else
 9:             d ← rr ∗ (1 + nf)−‖k, n‖
10:             if d > 0 then
11:                 e ← e + d
12:             end if
13:         end if
14:     end for
15:     return (e/i)
16: end procedure
```

Capítulo 5

Experimentos e Análise dos Resultados

Este capítulo descreve os experimentos realizados para verificar a qualidade do algoritmo proposto para localização em RSSFs. Foram utilizadas diversas variações de instâncias do problema e os resultados apresentados são comparados com outros trabalhos publicados sobre o problema (Niewiadomska-Szynkiewicz e Marks [2009], Carter et al. [2007], Cassioli [2009], Tseng [2008]). O trabalho de Niewiadomska-Szynkiewicz e Marks [2009] foi escolhido por ter disponibilizado suas instâncias para comparação (algo incomum nessa área). Já o trabalho do Carter et al. [2007] foi comparado por ser considerado o estado da arte por Cassioli [2009]. E os trabalhos do Cassioli [2009] e Tseng [2008] foram escolhidos por estarem publicadas comparações entre eles na literatura

Com exceção de Niewiadomska-Szynkiewicz e Marks [2009], os principais trabalhos publicados sobre o problema de localização em RSSFs não publicaram um *benchmark* de instâncias para o problema. Logo foi implementado um gerador de instâncias capaz de gerar instâncias com as mesmas características daquelas apresentadas nestes trabalhos. Para quase todas as instâncias a rede foi gerada posicionando-se os nós aleatoriamente dentro de uma região de tamanho R x R, sendo R = 1; apenas uma instância teve os nós posicionados em uma topologia regular (triângulo equilátero). Os nós âncoras eram escolhidos aleatoriamente dentre os nós da rede.

Ao gerar as instâncias foi calculada a distância d entre todos pares de nós. Para os pares de nós cuja distância era menor que o alcance do rádio, o valor de distância considerado como entrada para o problema era dado por: $d(1 + nf * \alpha)$, onde nf é o parâmetro de fator de ruído da instância, e α um valor aleatório seguindo a distribuição normal com média 0 e variância 1 (valores fora do intervalo $[-1, 1]$ foram descartados). Tal abordagem é muito comum na literatura [Mao et al., 2007; Carter et al., 2007;

Niewiadomska-Szynkiewicz e Marks, 2009; Cassioli, 2009]. Já para os pares de nós mais distantes que o alcance do rádio, as distâncias não foram dadas como entrada e o algoritmo calcula um erro caso encontre um posicionamento com distância menor que $rr * 1,001$, onde rr é o alcance do rádio.

Já para a comparação com Niewiadomska-Szynkiewicz e Marks [2009], os autores forneceram suas instâncias permitindo uma comparação mais precisa com o algoritmo proposto neste trabalho. As quatro instâncias fornecidas por eles seguem os mesmos princípios gerais apresentados acima, com relação à região, aplicação de fator de ruído, etc. Já em relação ao posicionamento dos nós, a primeira instância tem os nós posicionados de forma mais uniformemente espalhada na região. As outras três possuem os nós âncoras mal distribuídos (concentrados em partes da região), o que dificulta a localização. Além disso, em uma dessas três instâncias os nós desconhecidos também são colocados dessa forma.

Foram utilizados vários casos de teste variando-se a quantidade total de nós, o número de nós âncoras, o alcance do rádio e o fator de ruído. Espera-se que quanto maior o número de nós âncoras na rede melhor será o resultado do algoritmo. Da mesma forma, um alcance de rádio maior permite que os nós tenham mais referências para o posicionamento levando a um melhor resultado. Quanto ao fator de ruído é claro que quanto menor ele for, melhores serão os resultados.

O algoritmo foi implementado utilizando-se a linguagem C++, com gerador de números aleatórios Mersenne Twister [Matsumoto e Nishimura, 1998] e compilador GCC versão 4.3.3. Os testes foram executados utilizando-se um notebook Dell Inspiron 1525 com processador Core 2 Duo de 2 GHz e 2 GB de memória RAM, com o sistema operacional Linux Ubuntu 9.04.

As próximas seções apresentam os resultados encontrados, onde são apresentados os parâmetros de cada instância. Para as instâncias geradas, os resultados se referem a uma média de dez execuções do algoritmo sobre cada uma de dez instâncias geradas com as mesmas características; já para as instâncias fornecidas, os resultados se referem a uma média de dez execuções. Quanto aos parâmetros específicos do algoritmo, foram realizados testes variando-se os seus valores, e os melhores resultados encontrados em média foram aqueles que utilizaram a seguinte configuração: $ec = 10$, $t = 0.001$, $ft = 10$, $ftmin = 10$, $maxI = 10$ e tendo como condição de parada 20 iterações. Já a última seção apresenta a análise do tempo de execução do algoritmo proposto.

As tabelas de resultados apresentadas contêm os seguintes campos:

- n: número total de nós da rede
- na: número de nós âncoras

- ar: alcance do rádio
- fr: fator de ruído
- ϵ <método/autor>: erro encontrado pelo algoritmo de acordo com a métrica apresentada
- σ <método/autor>: desvio padrão do erro encontrado pelo algoritmo
- t <método/autor>: tempo de execução do algoritmo em segundos
- Para o algoritmo proposto é usada a notação σ Alg e t Alg

Em relação às métricas de erro não há uma padronização dos métodos de cálculo utilizados. Dessa forma, como cada autor propõe uma métrica específica, as comparações são feitas utilizando a métrica utilizada pelo autor comparado. A cada comparação é apresentada a métrica em questão. Além disso, essas métricas geralmente utilizam a posição real do nó para comparação, algo que não é possível obter em instâncias reais. Mas, de qualquer forma, são métricas interessantes para validarmos a qualidade do algoritmo. O método de cálculo de erro do próprio algoritmo, apresentado no capítulo anterior, poderia ser utilizada como estimativa de erro para instâncias reais.

A tabela 5.1 apresenta uma visão geral de todas as instâncias, apresentando o número de nós desconhecidos, a quantidade de nós inalcançáveis (que são desconsiderados no pré-processamento) e o número de nós com apenas 1 e com 2 vizinhos (a existência desses nós geralmente piora a qualidade encontrada pelos algoritmos de localização). São apresentadas médias nos casos de várias instâncias geradas com as mesmas características.

As próximas seções apresentam os resultados encontrados na comparação com os trabalhos dos outros autores.

5.1 Comparações com Niewiadomska-Szynkiewicz e Marks [2009]

Os testes realizados nessa seção utilizaram as instâncias fornecidas por Niewiadomska-Szynkiewicz e Marks [2009] para comparação dos resultados. Eles apresentam em seu trabalho os resultados de quatro métodos: SDP, *Simulated Annealing* (SA) e dois métodos híbridos que combinam técnicas geométricas (trilateração) com *Simulated Annealing* (TSA) e Algoritmo Genético (TGA).

Instância	Nro. Nós	Inalcançáveis	1 vizinho	2 vizinhos
evenly	180	0	0	0
unevenly A	180	0	0	0
unevenly B	180	0	0	0
unevenly C	180	0	0	2
4.1	90	0	0	0.1
4.2	90	0	0	0
4.3-01	42	0	0	0.1
4.3-03	210	0	0.1	0.2
4.3-04	506	0	0.2	0.6
4.4-01	3906	2.6	1.2	7.3
4.4-09	3906	0.9	1.3	4.7
4.4-16	3906	0	0.7	3.1
4.5-01	3919	0.1	0.6	2.1
4.5-09	3769	0.8	0.2	2.7
4.5-16	3569	0.6	0.5	2.7
4.6-01	3906	0.6	0.7	2.3
4.6-03	3906	0.3	0.7	2.4
4.6-07	3906	0.4	0.6	2.7
ts39a-01	900	0.8	0.8	3.1
ts39a-02	900	0.7	0.9	3.1
ts39a-03	900	0.2	0.4	3.6
ts39b-01	60	0	0.1	0
ts39b-02	60	0	0	0.1

Tabela 5.1. Detalhamento das instâncias

As quatro instâncias utilizadas possuem 200 nós, sendo 20 âncoras, com alcance de rádio igual a 0.18 e fator de ruído 0.1. A primeira instância (*evenly*) possui os nós espalhados aleatoriamente ao redor de toda região. Já as demais instâncias (*unevenlyA*, *unevenlyB* e *unevenlyC*) têm os nós âncoras mal-distribuídos concentrados em determinadas partes da região. Além disso, a instância *unevenlyC* possui também os nós desconhecidos mal-distribuídos.

A métrica de erro utilizada pelos autores é dada em relação ao alcance do rádio, através da fórmula: $\frac{1}{n}\sum_{i=1}^{n}\frac{(||\hat{x}_i - x_i||)^2}{ar^2}100\%$, onde x_i é a posição real do nó i, $\hat{x}_i$ é a posição encontrada pelo algoritmo para o nó, ar é o alcance de rádio e n é o número de nós. O erro é dado então em porcentagem e, segundo os autores, ele é normalizado em relação ao alcance do rádio para ser possível a comparação de resultados para redes de diferentes tamanhos e diferentes alcances de rádio.

A Tabela 5.2 apresenta os resultados das quatro instâncias comparando os quatro métodos de Niewiadomska-Szynkiewicz e Marks [2009] com o algoritmo proposto neste trabalho. Para a primeira instância, o algoritmo proposto encontra melhores resultados que o SA e o TGA, ficando próximo das outras duas técnicas. Já para as outras três instâncias, que são casos mais complexos, o algoritmo proposto alcança melhores resultados que todas as quatro técnicas comparadas. Destaca-se o resultado da última instância para a qual Niewiadomska-Szynkiewicz e Marks [2009] chegam a dizer em seu trabalho que métodos baseados em distância não são adequados para localização em redes com nós mal-distribuídos, mas o algoritmo proposto consegue resolver bem tam-

bém este caso (o melhor erro encontrado por eles foi de 133.78%, enquanto o algoritmo proposto alcançou 3.4%). A Figura 5.1 mostra as soluções encontradas por todos os métodos nessa última instância.

Instância	ϵ SDP	t SDP	ϵ SA	t SA	ϵ TSA	t TSA	ϵ TGA	t TGA	ϵ Alg	σ Alg	t Alg
evenly	0.18	6.95	2.76	3.04	**0.13**	0.46	3.80	2.85	0.27	0.060	13.13
unevenlyA	174.91	5.51	233.89	2.85	1.78	0.44	20.61	2.34	**1.21**	0.548	24.41
unevenlyB	330.56	6.25	293.01	3.06	1.81	0.47	56.06	2.90	**0.65**	0.209	18.47
unevenlyC	434.83	8.95	446.13	3.84	433.09	0.61	133.78	3.46	**3.74**	2.934	27.97

Tabela 5.2. Comparação com Niewiadomska-Szynkiewicz e Marks [2009] em redes de 200 nós, sendo 20 âncoras, alcance de rádio 0.18 e fator de ruído 0.1. A instância *evenly* possui os nós espalhados em todas as regiões, as demais possuem os nós âncoras concentrados em determinadas partes, enquanto a último possui também os nós desconhecidos concentrados.

5.2 Comparações com Carter et al. [2007] e Cassioli [2009]

Os testes realizados nessa seção foram propostos por Carter et al. [2007] e utilizados por Cassioli [2009] para comparação de seus resultados. Foram utilizados então os resultados publicados neste último trabalho para a comparação com o algoritmo proposto nesse trabalho e as instâncias foram geradas utilizando as mesmas características descritas pelos autores. Foram escolhidas 14 das 47 instâncias publicadas por Cassioli [2009]: para cada tipo de comparação (tamanho de rede, fator de ruído, etc.) foram escolhidas três instâncias representativas do grupo (geralmente a de tamanho menor, a de tamanho médio e a maior).

Para esse grupo de testes utilizou-se como estimativa de erro a média das distâncias entre a posição real de cada nó na instância e a posição encontrada para o nó pelo algoritmo.

5.2.1 Primeiros testes

O primeiro teste foi executado com uma rede contendo um total de 100 nós, sendo 10 âncoras, alcance de rádio 0.2275 e sem ruído. A Tabela 5.3 apresenta os resultados que mostram que o algoritmo proposto encontrou um resultado melhor do que os algoritmos dos demais autores. A Figura 5.2 mostra um exemplo de resultado do algoritmo.

O segundo teste teve como objetivo avaliar o comportamento de algoritmo sob condição de ruído e com um posicionamento dos âncoras desfavorável. A rede gerada possui 100 nós posicionados nos vértices de triângulos equiláteros, sendo que os 10

Inst.	n	na	ar	fr	ϵ Carter	t Carter	ϵ Cassioli	t Cassioli	ϵ Alg	σ Alg	t Alg
4.1	100	10	0.2275	0	$1.47e^{-7}$	0.35	$9.2e^{-5}$	0.011	$\mathbf{9.2e^{-8}}$	$1.04e^{-7}$	1.3

Tabela 5.3. Comparação com Carter et al. [2007] e Cassioli [2009] em redes de 100 nós, sendo 10 âncoras, alcance de rádio 0.2275 e sem fator de ruído

âncoras foram escolhidos como sendo o nó central de cada linha de nós. Foi utilizado alcance de rádio 0.25 e fator de ruído 0.1.

A Tabela 5.4 apresenta os resultados que mostram que o valor obtido pelo algoritmo proposto é melhor do que o encontrado no trabalho do Carter et al. [2007] (Cassioli [2009] não publicou resultado para essa instância). A Figura 5.3 mostra como é a rede e a solução encontrada pelo algoritmo.

Inst.	n	na	ar	fr	ϵ Carter	t Carter	ϵ Alg	σ Alg	t Alg
4.2	100	10	0.25	0.1	0.0203	0.51	**0.01282**	$1.10e^{-3}$	5.0

Tabela 5.4. Comparação com Carter et al. [2007] em redes de 100 nós, sendo 10 âncoras posicionados desfavoravelmente, alcance de rádio 0.2275 e com fator de ruído 0.1

5.2.2 Impacto do tamanho da rede

Esse conjunto de testes teve como objetivo avaliar a qualidade das soluções do algoritmo para diferentes tamanhos de rede. Percebe-se que quanto menor a rede melhores são os resultados. Isso se explica pelo fato de que há uma certa propagação de erros no posicionamento sucessivo por trilateração ou interseção de dois círculos. Quanto mais distante um nó está de um âncora (em relação ao número de nós que são necessários para levar uma informação para o âncora) maior será esse erro propagado. De qualquer forma os resultados encontrados foram muito bons, sendo sempre bem melhores que os publicados por Cassioli [2009] e bem similares aos publicados por Carter et al. [2007], conforme demonstrado na Tabela 5.5.

Inst.	n	na	ar	fr	ϵ Carter	t Carter	ϵ Cassioli	t Cassioli	ϵ Alg	σ Alg	t Alg
4.3-01	49	7	0.3412	0	$\mathbf{4.58e^{-8}}$	0.18	$2.5e^{-5}$	0.003	$5.12e^{-8}$	$7.30e^{-8}$	0.4
4.3-03	225	15	0.1462	0	$\mathbf{4.49e^{-7}}$	0.82	0.00583	0.062	$1.56e^{-6}$	$1.72e^{-6}$	5.9
4.3-04	529	23	0.0931	0	$\mathbf{8.99e^{-7}}$	2.02	0.00488	0.286	$3.33e^{-5}$	$5.15e^{-5}$	33.6

Tabela 5.5. Comparação com Carter et al. [2007] e Cassioli [2009] em redes de diferentes tamanhos e sem fator de ruído

5.2.3 Impacto do alcance de rádio

O próximo teste consistiu na avaliação de qual é o impacto da variação do alcance do rádio. Como era de se esperar, quanto maior o alcance do rádio melhores são os resultados. Isso se explica pelo fato de que, quanto maior esse valor, maior será a quantidade de nós vizinhos e, portanto, maior o número de referências para a trilateração ou interseção de dois círculos. Dessa forma, além do nó ser melhor posicionado, a propagação de erros descrita na seção anterior é menor.

Os resultados encontrados foram melhores que os publicados por Cassioli [2009], mas um pouco piores que os publicados por Carter et al. [2007], conforme demonstrado na Tabela 5.6.

Inst.	n	na	ar	fr	ϵ Carter	t Carter	ϵ Cassioli	t Cassioli	ϵ Alg	σ Alg	t Alg
4.4-01	3969	63	0.0304	0	$\mathbf{2.44e^{-3}}$	18.03	0.0982	15.23	0.01057	$8.02e{-}3$	2640.9
4.4-09	3969	63	0.0320	0	$\mathbf{4.21e^{-4}}$	18.91	0.03493	11.41	$4.70e^{-3}$	$1.45e{-}3$	2237.2
4.4-16	3969	63	0.0334	0	$\mathbf{1.24e^{-4}}$	18.79	0.03469	9.28	$4.89e^{-3}$	$1.45e{-}3$	2096.8

Tabela 5.6. Comparação com Carter et al. [2007] e Cassioli [2009] em redes de 3969 nós, sendo 63 âncoras, sem fator de ruído e variando-se o alcance de rádio

5.2.4 Impacto do número de âncoras

Esse grupo de testes teve como objetivo verificar o impacto da variação da quantidade de âncoras na rede. Este teste demonstra que a qualidade do algoritmo proposto está diretamente relacionada à quantidade de âncoras presentes na rede. Em uma rede bem grande, com 3969 nós, o algoritmo alcança melhores resultados que os demais quando se tem 200 e 400 âncoras. Já para a instância de 50 âncoras, o algoritmo é melhor que o Cassioli [2009] e bem próximo ao resultado do Carter et al. [2007] (como mostrado na Tabela 5.7).

Inst.	n	na	ar	fr	ϵ Carter	t Carter	ϵ Cassioli	t Cassioli	ϵ Alg	σ Alg	t Alg
4.5-01	3969	50	0.0334	0	**0.00111**	19.38	0.03028	11.31	$2.99e^{-3}$	$1.90e^{-3}$	2072.8
4.5-09	3969	200	0.0334	0	$4.90e^{-5}$	18.77	0.01624	5.91	$\mathbf{3.51e^{-5}}$	$1.78e^{-5}$	1731.3
4.5-16	3969	400	0.0334	0	$6.40e^{-6}$	18.16	0.01381	4.24	$\mathbf{3.42e^{-6}}$	$4.22e^{-6}$	1692.5

Tabela 5.7. Comparação com Carter et al. [2007] e Cassioli [2009] em redes de 3969 nós, alcance de rádio 0.0334, sem fator de ruído e variando-se a quantidade de âncoras

5.2.5 Impacto do fator de ruído

Esse grupo de testes teve como objetivo verificar o impacto da variação do fator de ruído. Quanto maior o fator de ruído pior será a qualidade da solução encontrada e foi realmente isso que aconteceu. Para essas instâncias os resultados do algoritmo proposto não foram melhores que os demais trabalhos comparados. Nas conclusões são apresentadas sugestões de trabalhos futuros visando melhorar os resultados do algoritmo para essas circunstâncias. Os resultados são apresentados na Tabela 5.8.

Inst.	n	na	ar	fr	ϵ Carter	t Carter	ϵ Cassioli	t Cassioli	ϵ Alg	σ Alg	t Alg
4.6-01	3969	63	0.0334	0.01	**0.00096**	20.38	0.0068	9.52	0.01644	$9.34e^{-3}$	4766.9
4.6-03	3969	63	0.0334	0.1	**0.00687**	21.46	0.00707	10.88	0.10282	0.011	4406.3
4.6-07	3969	63	0.0334	0.5	0.0305	22.07	**0.01658**	11.33	0.22086	0.033	3098.9

Tabela 5.8. Comparação com Carter et al. [2007] e Cassioli [2009] em redes de 3969 nós, sendo 63 âncoras, alcance de rádio 0.0334 e variando-se o fator de ruído

5.3 Comparações com Tseng [2008] e Cassioli [2009]

O próximo conjunto de instâncias também foi avaliado a partir de resultados apresentados por Cassioli [2009], mas dessa vez comparados com o trabalho de Tseng [2008] (as instâncias foram geradas utilizando as mesmas características descritas pelos autores).

O primeiro grupo de testes verificou o impacto da variação do fator de ruído. Como medida de erro utilizou-se o maior desvio, ou seja, a maior distância entre a posição real de um nó e a posição encontrada para ele pelo algoritmo. Foram encontrados melhores resultados nas três instâncias conforme demonstrado na Tabela 5.9.

Inst.	n	na	ar	fr	ϵ Tseng	t Tseng	ϵ Cassioli	t Cassioli	ϵ Alg	σ Alg	t Alg
ts39a-01	1000	100	0.06	0	0.11	0.2	0.10	0.92	**0.01723**	$7.30e^{-3}$	118.6
ts39a-02	1000	100	0.06	0.001	0.17	0.4	0.14	0.79	**0.01577**	0.014	165.3
ts39a-03	1000	100	0.06	0.01	0.17	1.6	0.0964	0.93	**0.02707**	0.013	306.7

Tabela 5.9. Comparação com Tseng [2008] e Cassioli [2009] em redes de 1000 nós, sendo 100 âncoras, alcance de rádio 0.66 e variando-se o fator de ruído

O último conjunto de testes consistiu em uma rede contendo apenas 4 nós âncoras posicionados nos pontos (0.05, 0.05), (0.05, 0.95), (0.95, 0.05) e (0.95, 0.95). A medida de erro utilizada foi o desvio quadrado ($e = \sum_{n_i \in N} ||n_i - \hat{n}_i||^2$, onde n_i é a posição encontrada pelo algoritmo e $\hat{n}_i$ a posição real do nó). Para essas instâncias o algoritmo proposto obtém o melhor resultado para a instância com maior ruído e o segundo melhor resultado para a outra instância (veja Tabela 5.10).

Inst.	n	na	ar	fr	ϵ Tseng	t Tseng	ϵ Cassioli	t Cassioli	ϵ Alg	σ Alg	t Alg
ts39b-01	64	4	0.3	0.1	**0.24**	–	0.781	0.269	0.419	0.373	6.9
ts39b-02	64	4	0.3	0.2	0.48	–	0.41	0.139	**0.31**	0.231	7.6

Tabela 5.10. Comparação com Tseng [2008] e Cassioli [2009] em redes de 64 nós, sendo 4 âncoras posicionados nos cantos da rede, alcance de rádio 0.3 e fator de ruído 0.1

5.4 Análise do tempo de execução do algoritmo proposto

Nesta seção é apresentada a análise do tempo de execução do algoritmo proposto. Para isso foi utilizada a abordagem *time-to-target* [Aiex et al., 2006], procurando mostrar através de gráficos a probabilidade de se obter uma solução com determinada qualidade em determinado tempo. Com isso é possível verificar a convergência do algoritmo e também a variabilidade dos tempos de execução.

Para essa análise foram utilizadas as quatro instâncias fornecidas por Niewiadomska-Szynkiewicz e Marks [2009]. Para cada uma delas, foram realizadas cem execuções independentes do algoritmo utilizando como alvo o valor médio encontrado para a função objetivo nos resultados apresentados na Seção 5.1. A Tabela 5.11 apresenta os valores da função objetivo utilizados como alvo, e os tempos mínimo, médio e máximo encontrados para cada instância.

Instância	$F(x)$ alvo	t min	t med	t max
evenly	0.002558	4.19	16.82	58.00
unevenlyA	0.003399	9.40	30.81	81.59
unevenlyB	0.002962	6.90	24.88	77.39
unevenlyC	0.004009	7.82	24.50	52.12

Tabela 5.11. Valores alvo da função objetivo do algoritmo proposto para as instâncias fornecidas por Niewiadomska-Szynkiewicz e Marks [2009] e valores de tempos de execução mínimo, médio e máximo para cem execuções independentes do algoritmo

Para cada instância, os gráficos foram gerados a partir dos valores de tempo de execução das cem execuções utilizando o aplicativo tttplots [Aiex et al., 2006] e eles são apresentados na Figura 5.4. Para a primeira instância, *evenly*, nota-se no gráfico (a) que com 15 segundos tem-se 60% de probabilidade de se encontrar uma solução com $F(x) = 0.002558$, e com cerca de 24 segundos a probabilidade sobe para 80%. A mesma análise pode ser feita para as demais instâncias: para a instância *unevenlyA* com cerca de 45 segundos têm-se 80% de probabilidade de se encontrar a solução alvo; para a instância *unevenlyB* essa mesma probabilidade é encontrada com cerca de 38

segundos e para a instância *unevenlyC* com 34 segundos.

Já o gráfico (b) permite uma análise da variabilidade dos tempos de execução. Os tempos de cada uma das cem execuções são plotados no gráfico e são comparados a uma distribuição teórica; e quanto mais próximos os dados reais estão dessa linha teórica, menor é a variabilidade dos tempos de execução. Nota-se, em todas as instâncias, que a grande maioria dos dados estão dentro da faixa de tolerância de um desvio padrão pra cima e um pra baixo dessa curva. Apenas uma minoria dos dados, que são os maiores tempos de execução, fogem um pouco da curva. Dessa forma, pode-se afirmar que a variabilidade dos tempos de execução do algoritmo é relativamente baixa.

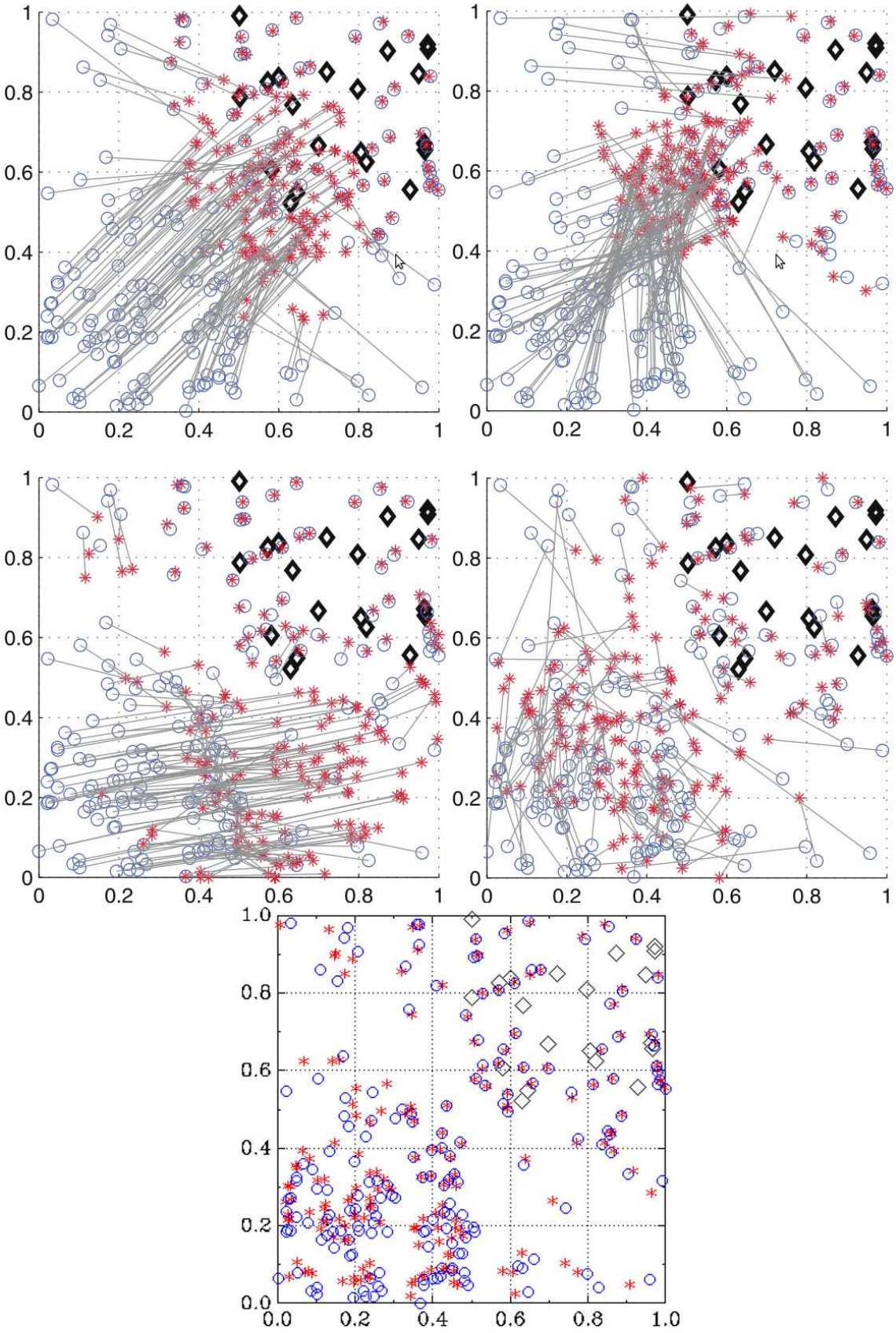

Figura 5.1. Comparação com os métodos de Niewiadomska-Szynkiewicz e Marks [2009] na instância *unevenlyC* (200 nós, sendo 20 âncoras, alcance de rádio de 0.18 e fator de ruído igual a 0.1). O gráfico no alto à esquerda se refere ao método SDP, à direita SA e depois TSA, TGA e o algoritmo proposto neste trabalho, respectivamente. Os losangos representam os âncoras, os círculos as posições reais dos nós, os asteriscos as posições encontradas pelo algoritmo e os traços a distância entre a posição real e a posição encontrada de cada nó.

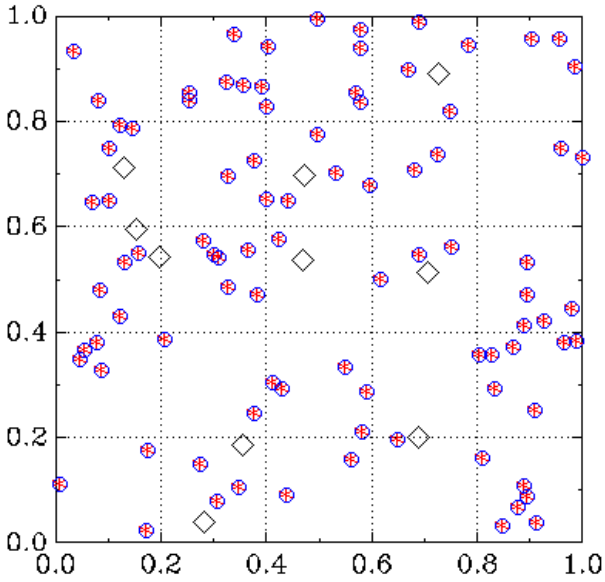

Figura 5.2. Exemplo de execução do algoritmo para rede com 100 nós, sendo 10 âncoras, alcance de rádio 0.2275 e sem ruído. Os losangos representam os âncoras, os círculos as posições reais dos nós e os asteriscos as posições encontradas pelo algoritmo.

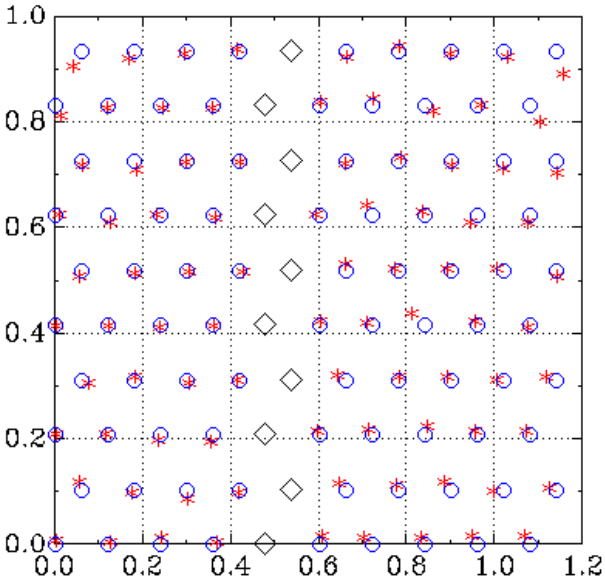

Figura 5.3. Exemplo de execução do algoritmo para rede com 100 nós, sendo 10 âncoras com posição desfavorável, alcance de rádio 0.25 e fator de ruído 0.1. Os losangos representam os âncoras, os círculos as posições reais dos nós e os asteriscos as posições encontradas pelo algoritmo.

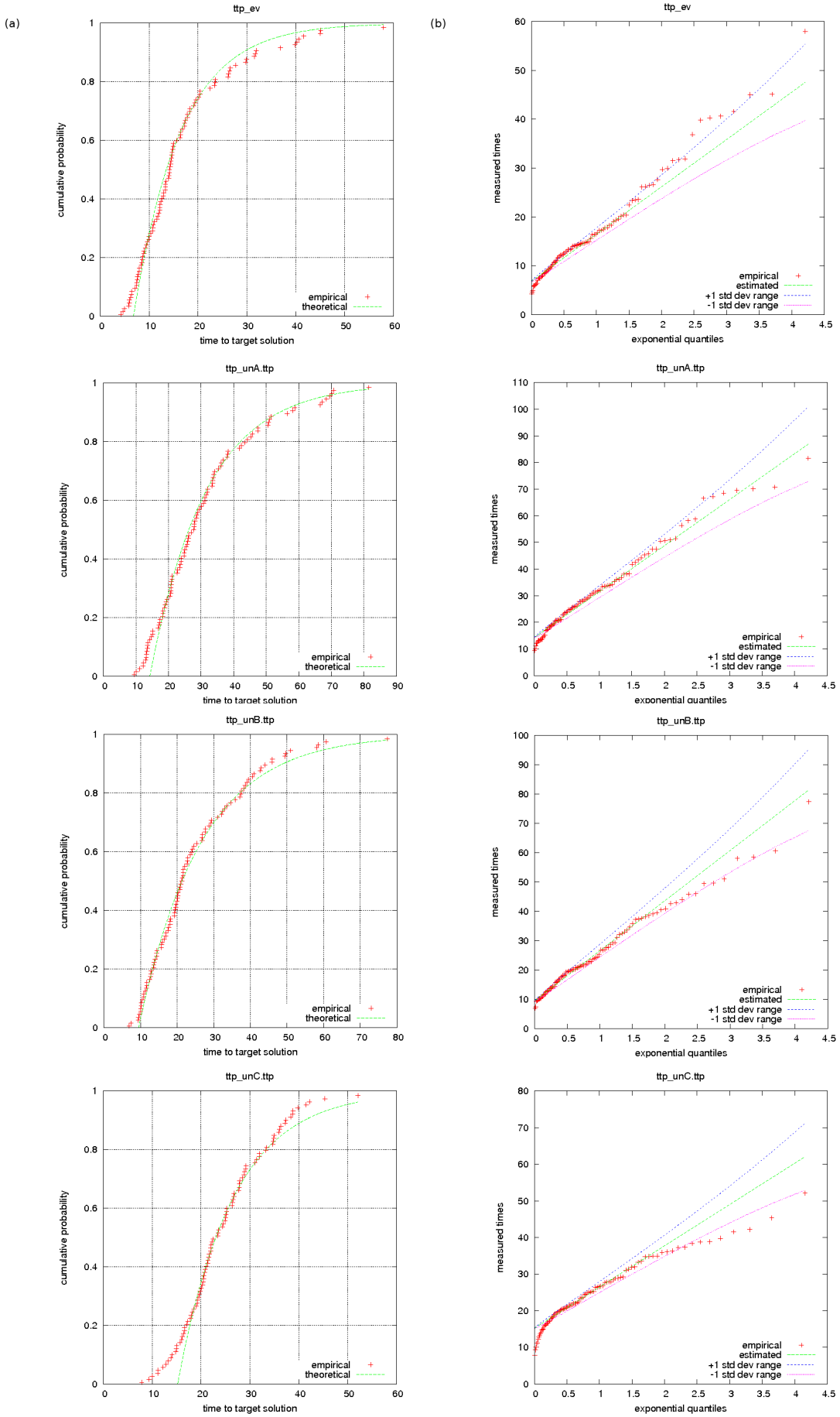

Figura 5.4. Gráficos *time-to-target* gerados utilizando tttplots [Aiex et al., 2006] para as instâncias fornecidas por Niewiadomska-Szynkiewicz e Marks [2009]. As probabilidades são em porcentagem e os tempos em segundos.

Capítulo 6

Conclusões

O presente trabalho aborda o problema de Localização em Redes de Sensores Sem Fio, apresentando o problema, suas classificações, modelagens e complexidade e propõe uma heurística específica para o problema, baseada em trilaterações, com abordagem centralizada, baseada em distância (*range based*) e *anchor-based*. São apresentados também trabalhos relacionados que tratam o problema também de forma centralizada.

O algoritmo proposto consiste em um procedimento iterativo que posiciona um nó da rede por vez, sempre tentando fazê-lo da forma mais confiável possível. Primeiro tenta-se posicionar algum nó por trilateração, se não for possível posicionar nenhum nó por trilateração tenta-se por interseção de dois círculos. Algoritmos baseados em trilateração geralmente não encontram boas soluções em instâncias com ruído e não conseguem posicionar todos os nós da rede. A maior contribuição deste trabalho então é a forma como essas ferramentas são utilizadas: a escolha dos nós de referência, a ordem de preferência e o controle de tolerância de erros, permitem que se encontre soluções para qualquer rede com qualidade similar ou superior aos outros trabalhos publicados na literatura.

Foram comparados o algoritmo proposto com outros algoritmos para o problema de Localização em Redes de Sensores Sem Fio com abordagem centralizada. Em relação aos resultados publicados por Niewiadomska-Szynkiewicz e Marks [2009], foram utilizadas as mesmas instâncias e o algoritmo proposto encontra melhores resultados em três das quatro instâncias. Para as demais comparações foram geradas instâncias com as mesmas características que as apresentadas pelos autores. Em relação ao Cassioli [2009] foram encontrados melhores resultados em 14 das 18 instâncias apresentadas. Já em comparação com Tseng [2008] o algoritmo proposto se comporta melhor em quatro das cinco instâncias analisadas. Por fim, comparando-se os resultados com Carter et al. [2007], em 14 instâncias foram encontrados resultados melhores em quatro e bem similares em outras seis. E a análise dos desvios padrões dos erros alcançados pelo algoritmo

proposto mostra que a variabilidade dos resultados encontrados é relativamente baixa.

Em relação ao tempo de execução foi feita análise que mostra a convergência e a variabilidade do algoritmo. Os tempos apresentados pelo algoritmo proposto são maiores que os demais trabalhos comparados na literatura. Mas considera-se que eles são aceitáveis para aplicações reais de localização em RSSFs com sensores estáticos, pois valeria a pena gastar um pouco mais tempo para se obter soluções mais confiáveis.

Os resultados mostram que o algoritmo se comporta melhor com uma maior quantidade de âncoras. Para trabalhos futuros deve-se alterar o algoritmo para permitir que sejam alcançados melhores resultados para instâncias com menor quantidade de âncoras. Outra melhoria futura seria a criação de uma estratégia para dividir as redes das grandes instâncias em redes menores permitindo a execução do algoritmo em paralelo. Um outro ponto importante a se fazer em trabalhos futuros é a utilização do mesmo em instâncias reais de redes de sensores sem fio. Por fim, sugere-se a aplicação do algoritmo em problemas relacionados como realização de grafos e estrutura de proteínas, por exemplo.

Apêndice A

C-GRASP aplicado à Localização em RSSFs

A.1 A Meta-heurística C-GRASP

Segundo Feo e Resende [1995], GRASP (*greedy randomized adaptive search procedure*) é uma meta-heurística para problemas de otimização combinatória, na qual cada iteração consiste basicamente de duas fases: construção e busca local. A fase de construção constrói uma solução viável cuja vizinhança é investigada até que um mínimo local é encontrado durante a fase de busca local. Esse processo é repetido até que uma condição de parada seja atingida; e a melhor solução geral encontrada é dada como resultado.

Uma vez que o GRASP foi criado para ser aplicado a problemas otimização discreta, Hirsch et al. [2006] propõe uma extensão do método para trabalhar com problemas de otimização contínua global, a qual foi chamada de *Continuous-GRASP* (C-GRASP). Os autores tiveram como objetivo a criação de um método estocástico de busca local que fosse simples de implementar e que, ao mesmo tempo, pudesse ser aplicado a uma grande variedade de problemas, sem a necessidade de usar derivadas e relaxações. Segundo os autores, tais características tornam o método bem apropriado aos diversos problemas de otimização global.

No trabalho original [Hirsch et al., 2006] e em [Hirsch et al., 2009], os autores apresentam resultados que mostram a validade e a aplicabilidade do método proposto em diversos cenários. Seria interessante então um estudo que utilizasse essa meta-heurística com o intuito de encontrar soluções aproximadas para o problema da localização em RSSFs.

A próxima seção apresenta maiores detalhes da meta-heurística C-GRASP. O algoritmo geral e os algoritmos das fases construtiva e de busca local são explicados a

partir de seus pseudo-códigos.

A.1.1 O algoritmo

Essa seção descreve em detalhes o algoritmo da meta-heurística C-GRASP, tomando-se por base a versão melhorada do C-GRASP, proposta por Hirsch et al. [2010]. De maneira geral, o problema de otimização contínua global pode ser definido como encontrar $s^* = \text{argmin}\{f(x) \mid inf \leq x \leq sup\}$, onde $f : \mathbf{R}^n \rightarrow \mathbf{R}$, e inf, s e $sup \in \mathbf{R}^n$. Sendo o domínio S um hiper-retângulo $S = \{s = (s_1, ..., s_n) \in \mathbf{R}^n : inf \leq x \leq sup\}$, onde $sup_i \geq inf_i$, para todo $i = 1, ..., n$.

Segundo Hirsch et al. [2006], o C-GRASP trabalha discretizando o espaço de solução em uma grade uniforme. As fases de construção e busca local movem a solução através dos pontos da grade. À medida que o algoritmo progride, a grade adaptativamente se torna mais densa. A principal diferença do C-GRASP em relação ao GRASP original é que o primeiro possui uma série de ciclos de construção/busca local, nos quais o resultado da busca local realimenta a próxima fase construtiva. O algoritmo 9 apresenta o pseudo-código da meta-heurística. Ele possui sete parâmetros de entrada, são eles:

- n: dimensão do problema.
- inf e sup: limites inferior e superior do espaço de soluções.
- $f()$: função objetivo.
- h_i e h_f: densidades de discretização inicial e final.
- ρ_{lo}: porção da vizinhança da solução atual que é pesquisada na busca local.

O valor da melhor solução encontrada para a função objetivo é inicializado com infinito (linha 2). Como o C-GRASP é um procedimento de multi-partida, o método é repetido até que um critério de parada seja satisfeito (o critério poderia ser o número de iterações ou o tempo de execução, por exemplo). A cada iteração, a solução inicial s é ajustada (linha 4) como um ponto aleatório distribuído uniformemente sobre o hiper-retângulo em $\mathbf{R}^n$ definido por inf e sup. O parâmetro h que controla a discretização do espaço de busca é inicializado como h_i (linha 5). A construção e a busca local são então chamados enquanto a discretização for maior ou igual ao mínimo especificado (linhas 7 e 8).

A nova solução encontrada após a fase de busca local é comparada com a melhor solução encontrada até então (linha 9) e se ela é a melhor até agora, ela é guardada como

Algoritmo 9 Pseudo-código do C-GRASP

```
 1: procedure C-GRASP(n, inf, sup, f(), h_i, h_f, ρ_lo)
 2:     f* ← ∞
 3:     while Critério de parada não é satisfeito do
 4:         s ← AleatorioUniforme(inf, sup)
 5:         h ← h_i
 6:         while h ≥ h_f do
 7:             [s, Melh_c] ← ContrucaoGulosaAleatoria(s, f(), n, h, inf, sup, Melh_c)
 8:             [s, Melh_l] ← BuscaLocal(s, f(), n, h, inf, sup, ρ_lo, Melh_l)
 9:             if f(s) < f* then
10:                 s* ← s
11:                 f* ← f(s)
12:             end if
13:             if Melh_c = false and Melh_l = false then
14:                 h ← h/2
15:             end if
16:         end while
17:     end while
18:     return s*
19: end procedure
```

a nova solução (linhas 10 e 11). Se não houver melhora na solução nem na construção, nem na busca local, a variável h é dividida por uma constante que geralmente é igual a 2 (linha 14), o que significa que a densidade do espaço de busca será aumentada. A intenção disso é adaptar dinamicamente a densidade da busca de acordo com o resultado, permitindo um ajuste mais fino quando uma boa solução é encontrada.

Quando o critério de parada é satisfeito, a melhor solução encontrada é retornada como resposta.

O algoritmo 10 apresenta o pseudo-código da fase construtiva do C-GRASP. Ele recebe como entrada uma solução s e começa permitindo que todas as coordenadas de s sejam alteradas (elas não estão fixadas). Realiza-se uma busca em linha para a coordenada não-fixada i de s (se não for para reutilizar os valores) mantendo-se as outras $n-1$ coordenadas com seus valores atuais. O valor z_i da i^{a} coordenada que minimiza a função objetivo é guardado (linha 11), bem como o valor da função objetivo para g_i (linha 12).

Depois a busca em linha é realizada para cada coordenada não-fixada, e nas linhas 22 a 28 é formada uma lista de candidatos restrita (LCR) que contém as coordenadas não-fixadas i cujos valores de g_i são menores ou iguais a um limiar $min+\alpha(max-min)$, onde max e min são, respectivamente, os valores máximo e mínimo de g_i sobre todas as coordenadas não-fixadas de s, e $\alpha \in [0,1]$ é um valor aleatório ajustado no início do procedimento (linha 3).

Uma coordenada é escolhida aleatoriamente a partir da lista de candidatos restrita

Algoritmo 10 Fase Construtiva do C-GRASP

```
1: procedure ConstrucaoGulosaAleatoria(s, f(), n, h, inf, sup, Melh_c)
2:     NaoFixadas ← {1, 2, ..., n}
3:     α ← AleatorioUniforme(0,1)
4:     Reutilizar ← false
5:     while NaoFixadas ≠ ∅ do
6:         min ← +∞
7:         max ← −∞
8:         for i = 1 to n do
9:             if i ∈ NaoFixadas then
10:                if Reutilizar = false then
11:                    z_i ← BuscaEmLinha(s, h, i, n, f(), inf, sup)
12:                    g_i ← f(s_(z_i))
13:                end if
14:                if min > g_i then
15:                    min ← g_i
16:                end if
17:                if max < g_i then
18:                    max ← g_i
19:                end if
20:            end if
21:        end for
22:        LCR ← ∅
23:        Limiar ← min + α(max − min)
24:        for i = 1 to n do
25:            if i ∈ NaoFixadas and g_i ≤ Limiar then
26:                LCR ← LCR ∪ {i}
27:            end if
28:        end for
29:        j ← SelecionaElemAleatoriamente(LCR)
30:        if s_j = z_j then
31:            Reutilizar ← true
32:        else
33:            s_j ← z_j
34:            Reutilizar ← false
35:            Melh_c ← true
36:        end if
37:        NaoFixadas ← NaoFixadas − {j}
38:    end while
39:    return (s, Melh_c)
40: end procedure
```

(linha 29). Se s_j for igual a z_j, a variável *Reutilizar* é ajustada para *true*. Caso contrário, nas linhas 33 a 35, *Reutilizar* torna-se $false$, $Melh_c$ é ajustado para *true*, e s_j recebe z_j. Finalmente, na linha 37, a coordenada j de s é fixada, ao ser removida do conjunto $NaoFixadas$.

A escolha aleatória de uma coordenada a partir da lista de candidatos restrita garante o caráter guloso e aleatório da fase construtiva. O procedimento é executado até que todas as n coordenadas de s sejam fixadas. Quando isso acontece, s e $Melh_c$ são retornados pelo método.

O pseudo-código da fase de busca local é apresentado no algoritmo 11. Para um dado ponto de entrada $s \in \mathbf{R}^n$, o algoritmo de busca local gera uma vizinhança e determina quais pontos da vizinhança melhoram a função objetivo (se existirem). Se um ponto de melhora é encontrado, ele torna-se o ponto atual e a busca local continua a partir dessa nova solução.

Algoritmo 11 Fase de Busca Local do C-GRASP

```
 1: procedure BuscaLocal(s, f(), n, h, inf, sup, ρ_lo, Melh_l)
 2:     s* ← s
 3:     f* ← f(s)
 4:     NumPontosGrid ← ∏_{i=1}^{n} ⌈(sup_i − inf_i)/h⌉
 5:     MaxPontosExaminar ← ⌈ρ_lo.NumPontosGrid⌉
 6:     NroPontosExaminados ← 0
 7:     while NroPontosExaminados ≤ MaxPontosExaminar do
 8:         NroPontosExaminados ← NroPontosExaminados + 1
 9:         s ← SelecionaElemAleatoriamente(B_h(s*))
10:         if inf ≤ s ≤ sup and f(s) < f* then
11:             s* ← s
12:             f* ← f(s)
13:             Melh_l ← true
14:             NroPontosExaminados ← 0
15:         end if
16:     end while
17:     return (s*, Melh_l)
18: end procedure
```

Seja $s \in \mathbf{R}^n$ a solução atual e h o parâmetro atual de discretização da grade. Seja $S_h(\bar{s}) = \{s \in S \mid inf \leq s \leq sup, s = \bar{s} + \tau.h, \tau \in \mathbf{R}^n\}$ o conjunto de pontos que são passos inteiros (de tamanho h) a partir de s e $B_h(\bar{s}) = \{s \in S \mid s = s + h.(s' - \bar{s})/\|s' - \bar{s}\|, s' \in S_h(\bar{s}) \backslash \bar{s}\}$ a projeção dos pontos em $S_h(\bar{s}) \backslash \bar{s}$ sobre a hiper-esfera centrada em s de raio h. A vizinhança-h do ponto s é então definida como o conjunto de pontos em $B_h(\bar{s})$.

O método de busca local começa guardando a solução recebida como a melhor encontrada até agora (linhas 2 e 3). Nas linhas 4 e 5 são calculados o número de pontos na grade (de acordo como parâmetro de discretização h) e o número máximo

de pontos que serão examinados em $B_h(s^*)$ (utilizando o parâmetro ρ_{lo} que é porção da vizinhança que será examinada). Se todos esses pontos forem examinados e não for encontrado nenhum ponto melhor, a solução atual s^* é considerada como um mínimo local.

No laço das linhas 7 a 16, o método seleciona aleatoriamente uma quantidade $MaxPontosExaminar$ de pontos em $B_h(s^*)$, um de cada vez. Se a solução atual s é a melhor até agora, ela é guardada como a melhor solução e a variável $Melh_l$ é ajustada para *true*. O processo é então reiniciado a partir dessa nova solução inicial. A variável $Melh_l$ é usada para indicar se a busca local conseguiu melhorar a solução. O procedimento termina quando encontra a melhor solução s^* considerando a discretização h. A solução e a variável $Melh_l$ são então retornadas como resposta.

A próxima seção descreve como a meta-heurística foi utilizada para tentar resolver o problema de localização em Redes de Sensores Sem Fio

A.2 Implementação do C-GRASP para o problema de localização em RSSF

A.2.1 Aplicação do C-GRASP ao problema

A meta-heurística C-GRASP possui procedimentos simples de serem aplicados aos diversos problemas de otimização global. Os pontos chave a serem considerados são os mapeamentos das coordenadas dos pontos a serem trabalhados (que definem o espaço de soluções) e da função objetivo.

Para o problema de localização em RSSFs a forma mais simples de aplicar o C-GRASP é considerar cada coordenada de cada nó sensor x_i como uma coordenada no espaço de soluções. Dessa forma, sendo i o número de nós sensores e d o número de dimensões do espaço euclidiano considerado, o domínio do problema seria dado por um hiper-retângulo $S = \{s = (s_1, ..., s_n) \in \mathbf{R}^n : inf \leq x \leq sup, n = i.d\}$. Já a função objetivo poderia ser utilizada diretamente como definida no capítulo 2.

Para uma instância do problema que envolva 10 nós sensores, por exemplo, considerando coordenadas no espaço bidimensional (latitude e longitude), o domínio do problema seria dado por um conjunto $S \subseteq \mathbf{R}^{20}$.

Quanto aos limites inferiores e superiores do espaço de soluções, eles devem ser dados como entrada para o algoritmo, uma vez que é razoável considerar que se sejam conhecidos os limites da região onde os nós sensores foram colocados.

É importante notar que essa proposta exige uma grande quantidade de chamadas de avaliação da função objetivo quando a quantidade de nós sensores for grande. Dessa

forma, ao implementar essa proposta, foram ser feitos testes para verificar o desempenho do algoritmo. Foram então implementadas formas de melhorar o desempenho, como estruturas de dados auxiliares e modificações nos métodos construtivo e de busca em linha. Tais modificações são melhor descritas em outras seções à frente.

A.2.2 Alterações no algoritmo C-GRASP

Durante os testes de implementação para aplicação ao problema de localização de sensores em RSSF, foram feitas alterações em relação ao algoritmo proposto por Hirsch et al. [2010]. Elas são detalhadas a seguir.

A.2.2.1 Limitação no número de vizinhos

O algoritmo original propõe que o número de vizinhos verificados na busca local seja dado por uma porção ρ_{lo} de todos os pontos formados pela grade dentro dos limites inferiores e superiores de cada dimensão considerando a densidade h. Mas, pela modelagem descrita na sub-seção anterior, vemos que precisamos de duas dimensões para cada sensor. Isso leva a uma quantidade enorme de pontos possíveis na grade.

Ao analisar o artigo que propõe o algoritmo Hirsch et al. [2010], vê-se que o próprio autor utiliza uma limitação de 1000 vizinhos para os problemas com mais dimensões. Dessa forma, optou-se por parametrizar esse valor (número absoluto de vizinhos) ao invés de se utilizar uma porção da grade de pontos.

A.2.2.2 Fator de redução da densidade

Quanto ao fator de redução da densidade, o algoritmo original sempre divide a densidade por 2 quando precisa diminuí-la. A implementação foi feita parametrizando a porcentagem de diminuição da densidade de forma a dar mais opções para serem testadas. Para utilizar o algoritmo original basta utilizar o valor $0,5$ como fator de redução.

A.3 Conclusões da tentativa de utilização do C-GRASP

Várias outras alterações foram realizadas na tentativa de melhorar os resultados encontrados, tais como: utilização de estruturas de dados otimizadas para recalcular apenas as distâncias modificadas, mudanças nos métodos de busca em linha e de geração de vizinhos, etc. Algumas dessas mudanças propiciaram melhoras em instâncias de determinadas características, mas, no geral, não se conseguiu adaptar a meta-heurística de

forma a obter os resultados esperados. Uma das mudanças foi criação de um método de geração de solução inicial que acabou se tornando o método proposto neste trabalho.

Não descarta-se a possibilidade de ainda ser possível utilizar o C-GRASP para resolver o problema de localização em RSSFs, mas esse apêndice contribui com a literatura ao apresentar uma forma de modelagem na qual isso não é possível.

Referências Bibliográficas

Aiex, R. M.; Resende, M. G. C. e Ribeiro, C. C. (2006). Tttplots: A perl program to create time-to-target plots. *Optimization Letters*, 1(4):355–366.

Alfakih, A. Y. (2000). Graph rigidity via euclidean distance matrices. *Linear Algebra and its Applications*, 310(1-3):149 – 165.

Alfakih, A. Y. (2001). On rigidity and realizability of weighted graphs. *Linear Algebra and its Applications*, 325(1-3):57–70.

Aspnes, J.; Goldenberg, D. e Yang, Y. (2004). On the computational complexity of sensor network localization. *Lecture Notes in Computer Science*, 3121(Algorithmic Aspects of Wireless Sensor Networks):32–44.

Benson, S. e Ye, Y. (2008). Algorithm 875: Dsdp5—software for semidefinite programming. *ACM Transactions on Mathematical Software (TOMS)*, 34(3):16.

Biswas, P.; Toh, K. e Ye, Y. (2008). A distributed sdp approach for large-scale noisy anchor-free graph realization with applications to molecular conformation. *SIAM Journal on Scientific Computing*, 30(3):1251–1277.

Biswas, P. e Ye, Y. (2004). Semidefinite programming for ad hoc wireless sensor network localization. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, pp. 46–54. ACM.

Biswas, P. e Ye, Y. (2006). A distributed method for solving semidefinite programs arising from ad hoc wireless sensor network localization. *Nonconvex Optimization and Its Applications*, 82(Multiscale Optimization Methods and Applications):69–84.

Boukerche, A.; Oliveira, H. A.; Nakamura, E. e Loureiro, A. (2007). Localization systems for wireless sensor networks. *IEEE Wireless Communications*, 14(6):6–12.

Carter, M. W.; Jin, H. H.; Saunders, M. A. e Ye, Y. (2007). Spaseloc: An adaptive subproblem algorithm for scalable wireless sensor network localization. *SIAM Journal on Optimization*, 17(4):1102.

Cassioli, A. (2008). *Global optimization of highly multimodal problems*. PhD thesis, Universita degli Studi di Firenze.

Cassioli, A. (2009). Solving the sensor network localization problem using an heuristic multistage approach. *Computational Optimization and Applications*.

Dang, V.; Le, V. e Lee, Y. (2010). Distributed push-pull estimation for node localization in wireless sensor networks. *Journal of Parallel and Distributed*, pp. 1–14.

Doherty, L. e El Ghaoui, L. (2001). Convex position estimation in wireless sensor networks. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pp. 1655–1663. IEEE.

Estrin, D.; Govindan, R. e Heidemann, J. (1999). Next century challenges: Scalable coordination in sensor networks. *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, (Section 4):263—-270.

Feo, T. e Resende, M. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133.

Goldenberg, D.; Krishnamurthy, A.; Maness, W.; Yang, Y.; Young, A.; Morse, A. e Savvides, A. (2005). Network localization in partially localizable networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 1, pp. 313–326. IEEE.

Grippo, L. e Sciandrone, M. (2002). Nonmonotone globalization techniques for the barzilai-borwein gradient method. *Computational Optimization and Applications*, 23(2):143–169.

Hendrickson, B. (1995). The molecule problem: Exploiting structure in global optimization. *SIAM Journal on Optimization*, 5:835–857.

Hirsch, M.; Pardalos, P. e Resende, M. (2010). Speeding up continuous grasp. Technical Report 3.

Hirsch, M. J.; Meneses, C. N.; Pardalos, P. M. e Resende, M. G. C. (2006). Global optimization by continuous grasp. *Optimization Letters*, 1(2):201–212.

Hirsch, M. J.; Pardalos, P. M. e Resende, M. G. (2009). Solving systems of nonlinear equations with continuous grasp. *Nonlinear Analysis: Real World Applications*, 10(4):2000–2006.

Hu, L. e Evans, D. (2004). Localization for mobile sensor networks. In *Proceedings of the 10th annual international conference on Mobile computing and networking*, pp. 45–57. ACM.

Huang, M.; Chen, S. e Wang, Y. (2011). Minimum cost localization problem in wireless sensor networks. *Ad Hoc Networks*, 9(3):387–399.

Jin, H. (2005). *Scalable sensor localization algorithms for wireless sensor networks.* PhD thesis, University of Toronto.

Loureiro, A. A.; Nogueira, J. M. S.; Ruiz, L. B.; de Freitas Mini, R. A.; Nakamura, E. F. e Figueiredo, C. M. S. (2003). Redes de sensores sem fio. In *XXI Simpósio Brasileiro de Redes de Computadores*, pp. 179–226.

Mao, G.; Fidan, B. e Anderson, B. (2007). Wireless sensor network localization techniques. *Computer Networks*, 51(10):2529–2553.

Matsumoto, M. e Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30.

More, J. J. e Wu, Z. (1997). Global continuation for distance geometry problems. *SIAM Journal on Optimization*, 7(3):814–836.

Niculescu, D. e Nath, B. (2001). Ad hoc positioning system (aps). In *Global Telecommunications Conference, 2001. GLOBECOM'01. IEEE*, volume 5, pp. 2926–2931. Ieee.

Nie, J. (2007). Sum of squares method for sensor network localization. *Computational Optimization and Applications*, 43(2):151–179.

Niewiadomska-Szynkiewicz, E. e Marks, M. (2009). Optimization schemes for wireless sensor network localization. *International Journal of Applied Mathematics and Computer Science*, 19(2):291–302.

Oliveira, H. A. B. F. (2008). *Localização no tempo e no espaço em redes de sensores sem fio.* PhD thesis, UFMG.

Reghelin, R. (2007). Um algoritmo descentralizado de localização para rede de sensores sem fio usando calibragem cooperativa e heurísticas. Master's thesis, Universidade Federal de Santa Catarina.

Savvides, A.; Han, C. e Strivastava, M. (2001). Dynamic fine-grained localization in ad-hoc networks of sensors. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, pp. 166–179, New York, New York, USA. ACM.

Shang, Y.; Ruml, W.; Zhang, Y. e Fromherz, M. (2004). Localization from connectivity in sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 15(11):961–974.

So, A. M.-C. e Ye, Y. (2007). Theory of semidefinite programming for sensor network localization. *Mathematical Programming*, 109(2-3):367–384.

Thomas, F. e Ros, L. (2005). Revisiting trilateration for robot localization. *IEEE Transactions on Robotics*, 21(1):93–101.

Tseng, P. (2008). Second-order cone programming relaxation of sensor network localization. *SIAM Journal on Optimization*, 18(1):156–185.

Vecchio, M.; Lopez-Valcarce, R. e Marcelloni, F. (2011). A two-objective evolutionary approach based on topological constraints for node localization in wireless sensor networks. *Applied Soft Computing*, (In press).

Wymeersch, H.; Lien, J. e Win, M. Z. (2009). Cooperative localization in wireless networks. *Proceedings of the IEEE*, 97(2):427–450.

Zhu, C.; Byrd, R.; Lu, P. e Nocedal, J. (1997). L-bfgs-b: Fortran subroutines for large-scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 23(4):550–560.