Caminhos mais curtos em localização em espaços fechados

Cátia Alexandra Alves Santos

Caminhos mais curtos em localização em espaços fechados

Cátia Alexandra Alves Santos

Dissertação para a obtenção do Grau de **Mestre em Matemática**

Área de Especialização em **Computação**

Júri

Presidente: Pedro Quaresma

Orientador: Marta Pascoal

Vogal: João Soares

Data: junho de 2014

Resumo

Os sistemas de posicionamento global, basedos na navegação por satélites, têm sido amplamente utilizados para localização no exterior. Em espaços fechados estes sistemas não são operacionais, no entanto, a localização no interior de edifícios é um problema com aplicações em áreas diversas, tais como a saúde ou o turismo, e em desenvolvimento.

Este trabalho integra um projeto para implementação de um sistema de navegação em edifícios, baseado na rede sem fios pré-existente. Em geral os algoritmos que determinam a localização nestas condições são afetados pelo ruído que resulta da oscilação do sinal emitido pelos dispositivos eletrónicos. Pretende-se apresentar uma ferramenta complementar a estes algoritmos, que permita reduzir possíveis erros produzidos na localização. O objetivo da ferramenta é calcular limites inferiores para o tempo que um indivíduo precisa para se deslocar entre dois pontos de um edifício, como forma de eliminar localizações erradas. Este cálculo baseia-se na representação do edifício por um grafo, e no desenvolvimento de um método para determinar o caminho mais curto entre quaisquer dois pontos do edifício. A escolha do método a utilizar é baseada em testes computacionais realizados em problemas gerados aleatoriamente.

Palavras Chave: localização dentro de edifícios, caminho mais curto

Abstract

The global positioning systems, based on satellite navigation, have been widely used for outdoor location. Such systems are not reliable when operating indoors, however, the location inside buildings is a timely problem with applications in many fields, such as health and tourism, which has been attracting the attention of researchers and practitioners.

This work is part of a project to implement a navigation system within buildings, based on a pre-existing wireless network. The algorithms which determine the position under these conditions are often affected by noise, resulting from the oscillation of the signal emitted by the electronic devices. It is intended to present a complementary tool to these algorithms, which can reduce possible errors produced by the location. The purpose of the tool is to compute lower bounds for the time that a person needs to move between two points of a building, in order to eliminate wrong locations. This calculation is based on the representation of the building by a graph, and the development of a method for finding the shortest path between any two points in the building. The choice of the method to use is supported by tests on randomly generated problems.

Keywords: indoor location, shortest path

Agradecimentos

Agradeço à Professora Marta Pascoal e ao Engenheiro André Lemos pela orientação e por toda a disponibilidade e incentivo durante a elaboração desta dissertação.

Ao Departamento de Matemática por todos os recursos oferecidos, bem como a todos os Professores que me acompanharam durante todo o percurso académico.

A todos os colegas que, de alguma forma, ajudaram à realização desta dissertação.

Por fim, não podiam deixar de agradecer a toda a minha família, em especial aos meus pais, por todo o apoio e incentivo.

Esta dissertação foi desenvolvida no âmbito dos trabalhos de uma bolsa de investigação para licenciados, financiada pelo Projeto ADINLOC: Advanced Dynamic Indoor Localization System, QREN, Agência de Inovação.

União Europeia

Fundo Europeu de
Desenvolvimento Regional

Conteúdo

Capítulo 1

Introdução

O presente trabalho é desenvolvido no âmbito do projeto ADINLOC (abreviatura de *Advanced Dynamic Indoor Localization System*) em co-promoção com a companhia onCaring. Este projeto pretende desenvolver um produto para localização de indivíduos em espaços fechados, que se pretende implementar em unidades de cuidados continuados, tais como hospitais e residências assistidas.

Os sistemas de localização exterior, tais como o Sistema de Posicionamento Global (*Global Positioning System*), conhecido habitualmente por GPS, não permitem a localização dentro de edifícios com a exatidão pretendida. Desta forma, a localização *indoor*, ou seja, a localização dentro de edifícios, recorre a outras abordagens para atingir o mesmo objetivo. A estratégia de localização em espaços fechados proposta consiste na localização de pessoas usando dispositivos de transmissão/receção sem fios (ditos *wireless*). O meio *wireless* usado foi o *Wi-Fi*, dado ser uma infraestrutura já presente em muitos edifícios.

O produto resultante, chamado *onAll*, consiste num sistema composto por uma plataforma de *software* e dispositivos destinados a idosos e prestadores de cuidados em hospitais e residências assistidas. Existem dois tipos de dispositivos, um para o idoso e outro para o prestador de cuidados. Ambos permitem determinar a sua localização dentro do edifício. O dispositivo do idoso permite ainda a deteção de quedas e o dispositivo do prestador de cuidados permite receber notificações, caso seja necessário auxiliar algum idoso.

É importante que a determinação da localização seja precisa e fiável, pois pequenos erros em termos geográficos podem tornar-se erros mais graves em termos de topologia do edifício (quarto errado, piso errado, etc.). No contexto do projeto ADINLOC foram desenvolvidos algoritmos especializados para calcular as localizações dos indivíduos, tais como os algoritmos de multilateração e multiobjetivo. No entanto, estes métodos nem sempre calculam a localização com a exatidão pretendida. Assim, será conveniente minimizar os erros.

Com o presente trabalho pretende-se estudar e implementar métodos eficientes para funcionar em tempo real, capazes de calcular limites inferiores para o tempo mínimo que um indivíduo leva para se deslocar entre dois pontos dados de um edifício. Estes limites servirão para validar (ou rejeitar) localizações identificadas pelos algoritmos de multilateração e multiobjetivo. É importante que não sejam validadas localizações incorretas, pois pode dar-se o caso em que haja idosos em zonas proibidas, como sair do edifício.

Os algoritmos aqui descritos foram desenvolvidos em Java, de modo a poderem ser integrados com a restante plataforma, *onAll*.

O resto do trabalho organiza-se do seguinte modo. No Capítulo 2 apresentam-se alguns conceitos de grafos que serão usados posteriormente. É também descrito o problema do caminho mais curto e alguns dos algoritmos que o permitem resolver nas variantes "um nó inicial" e "todos os pares de nós". No final deste capítulo encontram-se alguns testes computacionais que permitirão avaliar o comportamento de vários algoritmos para resolver o problema do caminho mais curto entre qualquer par de nós. No Capítulo 3 é apresentado o problema principal do trabalho: o problema do caminho mais curto em edifícios, bem como a abordagem proposta, que utiliza os conceitos do capítulo anterior. Por fim, no último capítulo são apresentadas algumas conclusões e orientações para trabalho futuro.

Capítulo 2

Problema do Caminho Mais Curto

Neste capítulo introduzimos alguns conceitos de Teoria dos Grafos e de Otimização e Redes, que serão utilizados ao longo do trabalho. São apresentadas notações e noções introdutórias de grafos. Posteriormente descrevemos algumas variantes do problema do caminho mais curto e de algoritmos conhecidos para a sua resolução. Por fim, apresentam-se alguns testes computacionais de forma a avaliar a eficiência dos algoritmos.

2.1. Conceitos Elementares de Grafos

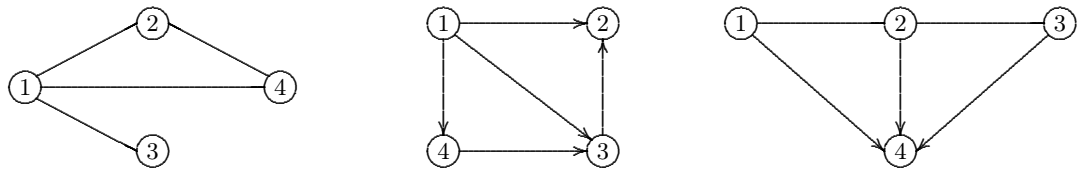
Um *grafo* é um par $G = (V, E)$, onde V é um conjunto finito, cujos elementos denominaremos por *nós* ou *vértices*, e $E \subseteq V \times V$ é também um conjunto finito, cujos elementos denominaremos por *arcos* ou *arestas*.

Cada nó de um grafo G será representado por um número natural i, $i \in \{1, \ldots, n\}$, e cada arco de G será representado por outro número natural $a_k \in E$, $k \in \{1, \ldots, m\}$. Além disso, cada arco a_k pode ser identificado por um par de nós de G, i e j. Um arco pode ser *orientado* ou *não orientado*, sendo representado por (i, j) ou por $\{i, j\}$, respetivamente. Um arco orientado apenas permite ser percorrido num sentido, enquanto que um arco não orientado pode ser percorrido em ambos os sentidos. Quando um arco é orientado, (i, j), diz-se que i é o *nó inicial do arco* e que j é o *nó terminal do arco*, ou que i é o *antecessor* de j e que j é o *sucessor* de i.

O grafo G diz-se *não orientado* se E contiver apenas arcos não orientados, diz-se *orientado* se contiver apenas arcos orientados e diz-se *misto* se contiver arcos orientados e não orientados.

Graficamente um grafo pode ser representado como na Figura 2.1, onde os nós são os círculos, enquanto que os arcos orientados são representados por uma seta e os não orientados por um segmento de reta.

A *densidade* de um grafo é o quociente entre o número de arcos e o número de nós do grafo.



(a) Grafo Não Orientado (b) Grafo Orientado (c) Grafo Misto

Figura 2.1: Grafos

Um grafo $G' = (V', E')$ diz-se um *subgrafo* de $G = (V, E)$ se $V' \subseteq V$ e $E' \subseteq E$.

Considerem-se dois nós, i_1 e i_r, de um grafo orientado G. Um *trajeto* de i_1 para i_r é uma sequência alternada de nós e arcos $\langle i_1, a_1, i_2, a_2, \ldots, a_{r-1}, i_r \rangle$, tal que $i_k \in V, \forall k \in \{1, \ldots, r\}$ e $a_k = (i_k, i_{k+1}) \in E, \forall k \in \{1, \ldots, r-1\}$. Se $a_k = (i_{k+1}, i_k)$ para algum $k \in \{1, \ldots, r\}$, então a sequência diz-se uma *cadeia*. Os nós i_1 e i_r denominam-se os nós *inicial* e *terminal* do trajeto, respetivamente. Admitindo que o grafo não contém *arcos paralelos*, isto é, arcos que ligam o mesmo par de nós, um trajeto pode representar-se apenas pelos seus nós, $\langle i_1, i_2, \ldots, i_r \rangle$.

Um *caminho* de i_1 para i_r é um trajeto de i_1 para i_r sem nós repetidos. Um nó *intermédio* de um caminho $p = \langle i_1, i_2, \ldots, i_r \rangle$ é qualquer nó de p diferente de i_1 ou i_r, ou seja, qualquer nó do conjunto $\{i_2, i_3, \ldots, i_{r-1}\}$. Um *ciclo* é um caminho com $\langle i_1, i_2, \ldots, i_r \rangle$ juntamente com o arco (i_r, i_1). É usual representar um ciclo por $\langle i_1, i_2, \ldots, i_r, i_1 \rangle$.

Um grafo diz-se *conexo* se existir uma *cadeia* entre cada par de nós do grafo e diz-se *fortemente conexo* se existir um trajeto entre qualquer par dos seus nós. Uma *árvore* é um grafo conexo sem ciclos. Uma *árvore com raiz num nó* é uma árvore com um nó designado *raiz*, tal que a partir da raiz existe um trajeto para todos os restantes nós.

Uma *rede* é um grafo a cujos nós, ou arcos, se associam valores numéricos. Daqui em diante associaremos a cada arco (i, j) de uma rede G um valor real d_{ij}, que designaremos por *distância* (ou *custo*) do arco (i, j). A Figura 2.2 ilustra uma destas redes.

2.2. Problema do Caminho Mais Curto

No que se segue, e caso nada seja dito em contrário, iremos considerar, sem perda de generalidade, que G é uma rede orientada sem arcos paralelos e que s e t, com $s \neq t$, são, respetivamente, os nós inicial e terminal.

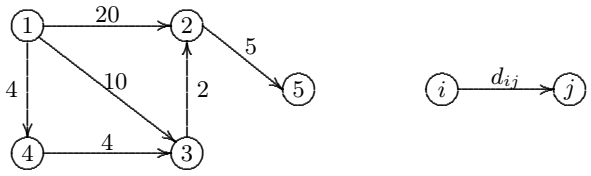


Figura 2.2: Rede

Dados dois nós $u, v \in V$, denotaremos por $\mathcal{P}_{uv}$ o conjunto dos trajetos que se definem de u para v em G e por $\mathcal{P}$ o conjunto $\mathcal{P}_{st}$. Iremos supor que existe pelo menos um trajeto em G de s para i e pelo menos um trajeto de i para t, onde i é um qualquer nó da rede, isto é, que $\mathcal{P}_{si} \neq \emptyset$ e $\mathcal{P}_{it} \neq \emptyset$, para todo o $i \in V$.

Cada trajeto em G pode ser associado a um valor, que pretendemos otimizar. Considerando que cada trajeto está associado a uma distância, definimos também a *distância* (ou *custo*) de um trajeto entre dois nós. Esta função é dada pela soma das distâncias (ou custos) de todos os arcos desse trajeto. Assim, dados dois nós $u, v \in V$, consideramos

$$d \ : \ \mathcal{P}_{uv} \longrightarrow \mathbb{R} \ : \ p \ \mapsto d(p) = \sum_{(i,j)\in p} d_{ij},$$

denominada *função distância* (ou *custo*). Por forma a simplificar a notação escreveremos $d(p) = \sum_{p} d_{ij}$.

No problema do trajeto mais curto pretendemos determinar um trajeto $p^\star$ de s para t, cuja distância (ou custo) seja mínima, isto é, tal que $d(p^\star) \leq d(p)$, para todo o $p \in \mathcal{P}$.

Caso não existam ciclos negativos em G, isto é, ciclos com distância negativa, o problema do trajeto mais curto tem sempre uma solução ótima que é um trajeto que não contém ciclos, ou seja, que é um caminho [1, 5]. Nestas circunstâncias o problema do trajeto mais curto reduz-se ao problema do caminho mais curto, cujo objetivo é determinar um caminho de s para t com distância mínima.

Diz-se que o problema do caminho mais curto verifica o *Princípio de Otimalidade* se qualquer caminho mais curto é constituído por subcaminhos mais curtos. A inexistência de ciclos negativos numa rede é condição necessária e suficiente para que o problema do caminho mais curto verifique o Princípio de Otimalidade.

Existem diversas variantes de problemas do caminho curto, das quais destacaremos as versões de um nó para outro nó; de um nó para todos os outros e entre todos os pares de nós. Os dois primeiros problemas serão tratados na Secção 2.2.1 e

o último na Secção 2.2.2.

2.2.1. Problema do Caminho Mais Curto Entre Um Par de Nós

Em seguida descrevemos o algoritmo geral de rotulação que, dada uma rede sem ciclos negativos, permite determinar a árvore dos caminhos mais curtos com raiz em s.

Consideremos um conjunto auxiliar Q, constituído pelos nós para os quais já se determinou um caminho com início em s e a partir dos quais ainda se poderão determinar alternativas melhores aos caminhos já encontrados.

A cada nó i da rede associamos um par, (π_i^s, ξ_i^s). O valor π_i^s armazena a distância do melhor caminho de s para i determinado numa dada altura da execução do algoritmo e o valor ξ_i^s armazena o nó que antecede i nesse caminho. Denominamos π_i^s por *rótulo* de i e ξ_i^s pelo seu *nó pai*.

Inicialmente é atribuído a π_i^s o valor $+\infty$, que é um majorante dos seus possíveis valores, $i \neq s$. A fase de inicialização destas variáveis encontra-se resumida no Algoritmo 1.

Algoritmo 1 Inicialização

Seja $G = (V, E)$ uma rede
for $i \in V$ **do**
  $\pi_i^s \leftarrow +\infty$
  $\xi_i^s \leftarrow -1$
end for
$\pi_s^s \leftarrow 0$
$\xi_s^s \leftarrow 0$

Sempre que possível é atribuído a π_i^s um valor melhor, ou seja, um valor inferior. Suponhamos que se conhece um caminho de s para v, denotado por p, com distância π_v^s e um caminho de s para w, denotado por q, com distância π_w^s. Então, se o arco (v, w) é tal que $\pi_w^s > \pi_v^s + d_{vw}$, isso significa que o caminho de s para w que resulta de p acrescido do arco (v, w) define um caminho de s para w com uma distância inferior à de q. Esta é a ideia fundamental utilizada em algoritmos de rotulação para o problema do caminho mais curto e encontra-se ilustrada na Figura 2.3.

O algoritmo geral de rotulação é resumido em seguida, no Algoritmo 2.

O algoritmo geral de rotulação pode ser implementado de várias formas, dependendo cada uma da estrutura de dados utilizada para representar o conjunto Q. Essa estrutura determina, em princípio, o modo como Q é manipulado e o nó a retirar de

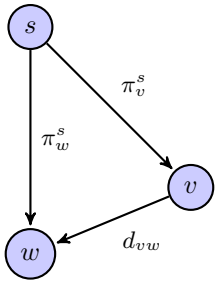


Figura 2.3: Melhoramento de um rótulo no algoritmo de rotulação

Algoritmo 2 Algoritmo geral de rotulação

```
Inicialização
Q ← {s}
while Q ≠ ∅ do
  i ← um nó de Q
  Q ← Q\{i}
  for (i, j) ∈ E do
    if π_j^s > π_i^s + d_ij then
      π_j^s ← π_i^s + d_ij
      ξ_j^s ← i
      Q ← Q ∪ {j}
    end if
  end for
end while
```

Q em cada passo, o que não é especificado no algoritmo.

Na literatura são conhecidas diversas versões do algoritmo geral de rotulação, entre as quais encontramos o algoritmo de Bellman-Ford e o algoritmo de Dijkstra [1]. Em qualquer dos casos admitimos que não existem ciclos com distância negativa.

No algoritmo de Bellman-Ford, a lista Q armazena nós por analisar. Nesta lista insere-se um nó v sempre que π_v^s tiver sido atualizado e remove-se o nó v sempre que este nó é analisado. O algoritmo começa com $Q = \{s\}$ e termina quando não existirem nós por analisar, ou seja, quando $Q = \emptyset$. O conjunto Q é manipulado como uma fila, isto é, como uma lista *first-in-first-out* (FIFO). Isto significa que as inserções de novos nós são realizadas no final da lista, enquanto que as remoções de nós para analisar são efetuadas no início da lista. O Algoritmo 3 apresenta o pseudo-código que resume este método.

Este algoritmo apresenta um número de operações com complexidade $\mathcal{O}(nm)$. Quando um nó v é analisado, o seu rótulo, π_v^s, ainda pode vir a ser melhorado novamente. Por este motivo diz-se que o algoritmo de Bellman-Ford utiliza uma técnica de *rotulação temporária*.

Algoritmo 3 Algoritmo de Bellman-Ford com lista FIFO

Inicialização
$Q \leftarrow \{s\}$
while $Q \neq \emptyset$ **do**
  $i \leftarrow$ primeiro nó de Q
  $Q \leftarrow Q \backslash \{i\}$
  for $(i,j) \in E$ **do**
    if $\pi_j^s > \pi_i^s + d_{ij}$ **then**
      $\pi_j^s \leftarrow \pi_i^s + d_{ij}$
      $\xi_j^s \leftarrow i$
      Inserir j no final de Q
    end if
  end for
end while

Tabela 2.1: Aplicação do algoritmo de Bellman-Ford com lista FIFO à rede da Figura 2.2

π_i^1 ξ_i^1			$(1,2)$	$(1,3)$	$(1,4)$	$(2,5)$		$(3,2)$		$(4,3)$		$(2,5)$		$(3,2)$		$(2,5)$	
1	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0
2	$+\infty$	-1	20		1	20	1	12	3	12	3	12	3	10	3	10	3
3	$+\infty$	-1	10		1	10	1	10	1	8	4	8	4	8	4	8	4
4	$+\infty$	-1	4		1	4	1	4	1	4	1	4	1	4	1	4	1
5	$+\infty$	-1	$+\infty$		-1	25	2	25	2	25	2	17	2	17	2	15	2

Tomemos como exemplo de aplicação do método de Bellman-Ford a rede da Figura 2.2, considerando o nó inicial $s = 1$. Os passos desta aplicação encontram-se resumidos na Tabela 2.1. A lista Q começa por conter apenas o nó 1, portanto é esse o primeiro nó a ser analisado. Em particular analisam-se os arcos com início no nó 1, $(1,2)$, $(1,3)$ e $(1,4)$. Todos os rótulos de sucessores destes arcos vão ser atualizados, pois d_{12}, d_{13} e d_{14} são valores finitos, fixando-se os rótulos $\pi_2^1 = 20$, $\pi_3^1 = 10$ e $\pi_4^1 = 4$. Simultaneamente o nó 1 é fixado como nó pai dos nós 2, 3 e 4. Agora temos $Q = \{2,3,4\}$ e escolhemos analisar o primeiro nó da lista, o nó 2. O rótulo do nó 5 vai ser atualizado, pois d_{25} é um valor finito. Agora $Q = \{3,4,5\}$.

O nó de Q a examinar em seguida é o nó 3. O arco $(3,2)$ é o único que se inicia neste nó, e permite corrigir o rótulo do nó 2, pois $\pi_2^1 = 20 > 10 + 2$. Procedemos então às atualizações $\pi_2^1 = 12$ e $\xi_2^1 = 3$ e a lista passa a ser $Q = \{4,5,2\}$. O mesmo raciocínio é repetido até Q estar vazia, como resumido na Tabela 2.1. É de notar que o nó 2 volta a ser escolhido em Q.

No fim do algoritmo a árvore dos caminhos mais curtos com raiz em 1 pode ser recuperada através do vetor ξ^1. Esta árvore é apresentada na Figura 2.4 e permite obter o caminho mais curto do nó 1 para qualquer outro nó. Junto a cada nó encontra-se o seu rótulo. Por exemplo, o caminho mais curto de 1 para 3 é $\langle 1, 4, 3 \rangle$ e tem custo $\pi_3^1 = 8$.

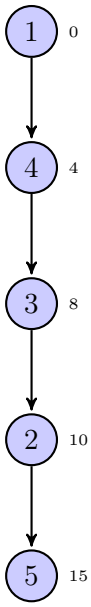
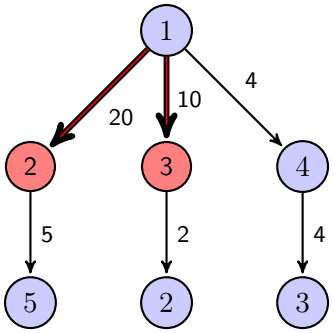


Figura 2.4: Árvore dos caminhos mais curtos a partir de 1, na rede da Figura 2.2

Uma variação do algoritmo de Bellman-Ford mostrou um melhoramento do desempenho relativamente à versão original, na exaustiva experiência computacional apresentada em [4]. Suponhamos que um nó v foi removido da lista Q e que o pai de v, $u = \xi_v^s$, ainda se encontra na lista. Note-se que π_v^s foi atualizado quando u foi verificado e que π_v^s passou a ser definido por $\pi_u^s + d_{uv}$. Como u foi adicionado à lista, então π_u^s foi atualizado. Nestas condições sabemos que u deverá voltar a ser analisado e que, provavelmente, isso implicará uma atualização de π_v^s. Então, intuitivamente, será mais conveniente analisar o nó v quando o nó u não pertencer à lista de nós por analisar. Neste método, que designaremos como algoritmo de Bellman-Ford com verificação do pai (*Parent Checking* no original), continuamos a manipular a lista de nós por analisar como uma lista FIFO, mas acrescentamos a condição de apenas analisar um nó se o seu pai não pertencer à lista Q. Caso contrário, a análise é adiada. Esta ideia pode ser implementada facilmente utilizando-se um vetor de valores lógicos indexado por V, que assume o valor verdade se o pai do nó v pertence a Q e falso caso contrário, para cada posição v do vetor.

Continuemos a considerar a rede da Figura 2.2 e apliquemos o algoritmo de



$Q = \{5, 2, 3\}$

Figura 2.5: Solução parcial dada pelo algoritmo de Bellman-Ford com verificação do pai, para a rede da Figura 2.2

Bellman-Ford com verificação do pai, para $s = 1$. Até à análise do arco $(4, 3)$, o processo é igual ao algoritmo anterior. Nesta fase do algoritmo a lista de nós por analisar é $Q = \{5, 2, 3\}$. É então altura de examinar o nó 5, no entanto o seu pai, o nó $\xi_5^1 = 2$, está na lista. Isso significa que π_2^1 já foi atualizado, e portanto o nó 2 ainda será novamente analisado, o que implica também uma atualização de π_5^1. Por este motivo não é conveniente analisar o nó 5 neste momento e, então, passa-se a analisar o próximo nó da lista, o nó 2, "ignorando" por agora o nó 5. Com o nó 2 passa-se o mesmo, pois o seu pai é o nó $\xi_2^1 = 3$, que está na lista Q. Analisa-se então o próximo nó, o nó 3. A árvore dos caminhos mais curtos determinada até esta fase do algoritmo está representada na Figura 2.5. Repare-se que o nó 3 aparece duas vezes na árvore. Isto acontece porque o rótulo de 3, a vermelho, foi atualizado para o representado a azul, apesar de ainda não ter sido concluída a mesma operação relativamente ao rótulo do nó 2. O mesmo acontece para o nó 2, pois a operação relativa ao nó 5 ainda não foi concluída.

Tabela 2.2: Aplicação do algoritmo de Bellman-Ford com verificação do pai à rede da Figura 2.2

π_i^1 ξ_i^1			$(1,2)$	$(1,3)$	$(1,4)$	$(2,5)$		$(3,2)$		$(4,3)$		$(3,2)$		$(2,5)$	
1	0	0	0		0	0	0	0	0	0	0	0	0	0	0
2	$+\infty$	-1	20		1	20	1	12	3	12	3	10	3	10	3
3	$+\infty$	-1	10		1	10	1	10	1	8	4	8	4	8	4
4	$+\infty$	-1	4		1	4	1	4	1	4	1	4	1	4	1
5	$+\infty$	-1	$+\infty$		-1	25	2	25	2	25	2	25	2	15	2

A Tabela 2.2 resume os passos da aplicação do algoritmo de Bellman-Ford com

verificação do pai à rede da Figura 2.2.

É de referir que, além de determinarem o caminho mais curto entre um par de nós s e t, tanto o algoritmo de Bellman-Ford como o algoritmo de Bellman-Ford com verificação do pai são também capazes de encontrar o caminho mais curto de s para todos os outros nós de G. Em ambas as situações a ordem de complexidade em termos de tempo no pior caso é a referida anteriormente, $\mathcal{O}(mn)$.

O algoritmo de Dijkstra utiliza uma ideia semelhante à do algoritmo de Bellman-Ford. A diferença reside nos factos de Q ser tratado como um conjunto e quando se remove um nó de Q ser escolhido aquele que tiver o menor rótulo. Este algoritmo pode ser aplicado desde que $d_{ij} \geq 0$, para todos os arcos (i, j) da rede.

Algoritmo 4 Algoritmo de Dijkstra

```
Inicialização
Q ← {s}
while Q ≠ ∅ do
  Seja i ∈ Q tal que π_i^s é mínimo
  Q ← Q\{i}
  for (i, j) ∈ E do
    if π_j^s > π_i^s + d_ij then
      π_j^s ← π_i^s + d_ij
      ξ_j^s ← i
      Q ← Q ∪ {j}
    end if
  end for
end while
```

Uma vez que se admite que todas as distâncias são não negativas, pode demonstrar-se que no algoritmo de Dijkstra cada nó é analisado uma única vez [1]. Por este motivo, e por oposição ao que acontece com o algoritmo de Bellman-Ford, o algoritmo de Dijkstra é dito um algoritmo de *rotulação definitiva*. Uma das consequências práticas da rotulação definitiva é permitir que o processo de rotulação seja interrompido assim que o nó terminal é analisado, obtendo-se assim apenas o caminho mais curto entre s e t. Continuando o processo até que Q vazio, permite a determinação dos caminhos mais curtos de s para todos os outros nós de G. Em qualquer dos casos o número de operações realizadas pelo algoritmo de Dijkstra em termos teóricos tem complexidade $\mathcal{O}(m + n\log(n))$.

A Tabela 2.3 resume um exemplo de aplicação deste método à rede da Figura 2.2, considerando $s = 1$. No início do algoritmo temos $Q = \{1\}$. Então é o nó 1 que começa por ser analisado. Tal como nos algoritmo anteriores são esteja analisados os

arcos $(1,2)$, $(1,3)$ e $(1,4)$. Então tem-se $Q = \{2,3,4\}$. O menor rótulo destes nós é π_4^1, pelo que vamos analisar o nó 4. Examinando o arco $(4,3)$ nota-se que $\pi_3^1 > \pi_4^1 + d_{43}$, então o rótulo π_3^1 é corrigido, tendo-se agora a lista $Q = \{2,3\}$. Em seguida o rótulo mínimo é o do nó 3, portanto será este o nó a analisar. Analisa-se agora o nó 2, o único nó da lista Q. Examinando o arco $(2,5)$ nota-se que $\pi_5^1 > \pi_2^1 + d_{25}$, então corrige-se o rótulo π_5^1, passando a tomar o valor 15. Por fim, resta analisar o nó 5 e, como não há nenhum arco com início em 5, o algoritmo termina.

Tabela 2.3: Aplicação do algoritmo de Dijkstra à rede da Figura 2.2

π_i^1 ξ_i^1			$(1,2)$ $(1,3)$ $(1,4)$		$(4,3)$		$(3,2)$		$(2,5)$	
1	0	0	0	0	0	0	0	0	0	0
2	$+\infty$	-1	20	1	20	1	10	3	10	3
3	$+\infty$	-1	10	1	8	4	8	4	8	4
4	$+\infty$	-1	4	1	4	1	4	1	4	1
5	$+\infty$	-1	$+\infty$	-1	$+\infty$	-1	$+\infty$	-1	15	2

2.2.2. Problema do Caminho Mais Curto Entre Todos os Pares de Nós

Tal como o nome indica, o problema dos caminhos mais curtos entre todos os pares de nós é semelhante ao anterior, mas agora consideram-se caminhos entre todos os possíveis pares de nós da rede.

É possível generalizar os algoritmos apresentados na secção anterior para determinar todos estes caminhos mais curtos, bastando para tal executá-los n vezes tomando para nó inicial cada nó da rede. Uma tal extensão do algoritmo de Bellman-Ford tem um número de operações com complexidade $\mathcal{O}(n^2 m)$ e, admitindo que todas as distâncias associadas aos arcos são não negativas, empregando o mesmo raciocínio podemos estender o algoritmo de Dijkstra, obtendo um método com um número de operações de $\mathcal{O}(nm + n^2 \log(n))$.

Contudo, existem métodos alternativos a estes, que permitem resolver o mesmo problema. Em seguida descrevemos brevemente dois dos métodos mais populares para este efeito, o algoritmo de Johnson e o algoritmo de Floyd-Warshall, válidos independentemente dos valores associados a cada arco.

O algoritmo de Johnson foi proposto em 1977 em [10]. A sua ideia base é transformar as distâncias em valores não negativos, por forma a permitir a aplicação do algoritmo de Dijkstra para determinação de cada caminho mais curto com um dado

nó inicial. No algoritmo de Johnson é acrescentado um novo nó s' ao grafo e é criado um arco de s' para todos os outros nós de V, com distância nula. Em seguida é aplicado o algoritmo de Bellman-Ford, tomando s' como nó inicial. Desta forma encontram-se os caminhos mais curtos de s' para todos os outros nós. Posteriormente os arcos do grafo original são re-pesados usando os valores calculados com o algoritmo de Bellman-Ford. Mais concretamente, a distância d_{ij} de um qualquer arco (i, j) é substituída pela distância

$$d'_{ij} = d_{ij} + \pi_i^{s'} - \pi_j^{s'}. \tag{2.1}$$

Pode mostrar-se que então

$$d'_{ij} \geq 0, \text{ para todo } (i, j) \in E,$$

o que permite a aplicação do algoritmo de Dijkstra nos passos seguintes. Por fim, o nó s' é removido e o algoritmo de Dijkstra é aplicado n vezes, de forma a encontrar o caminho mais curto de cada um dos nós para todos os outros. Este processo permite resolver o problema com um número de operações de $\mathcal{O}(nm + n^2 \log(n))$, em vez das $\mathcal{O}(n^2 m)$ operações necessárias numa implementação repetida do algoritmo de Bellman-Ford.

O algoritmo de Floyd-Warshall foi proposto de forma independente pelos dois autores, em 1962 [7, 18]. Na implementação deste algoritmo utilizam-se duas matrizes, π e ξ, indexadas por $V \times V$. Para $i, j \in V$, $\pi(i, j)$ armazena o custo do melhor caminho de i para j determinado numa dada altura da execução do algoritmo e $\xi(i, j)$ armazena o nó que antecede j nesse caminho.

Seja $\{1, 2, \ldots, k\} \subseteq V$ um subconjunto de k nós do grafo G, para algum k. Para qualquer par de nós $i, j \in V$ considerem-se todos os caminhos de i para j cujos nós intermédios estão no conjunto $\{1, 2, \ldots, k\}$ e p o caminho mais curto de todos esses caminhos. O algoritmo de Floyd-Warshall explora uma relação entre o caminho p e o caminho mais curto de i para j com todos os nós intermédios no conjunto $\{1, 2, \ldots, k-1\}$. A relação depende do facto de k ser, ou não, um nó intermédio do caminho p.

1. Se k não é um nó intermédio de p, então todos os nós intermédios do caminho p estão no conjunto $\{1, 2, \ldots, k-1\}$. Assim, o caminho mais curto de i para j com todos os nós intermédios no conjunto $\{1, 2, \ldots, k-1\}$ é também um caminho mais curto de i para j com todos os nós intermédios no conjunto $\{1, 2, \ldots, k\}$.

2. Se, pelo contrário, k é um nó intermédio no caminho p, então dividimos p em dois caminhos: o caminho p_1 do nó i até ao nó k e o caminho p_2 do nó k até ao nó j. Neste caso k não é nó intermédio de p_1 nem de p_2 e os nós intermédios comuns a estes dois caminhos pertencem ao conjunto $\{1, 2, \ldots, k-1\}$.

Seja $\pi^{(k)}(i,j)$ a distância do caminho de i para j determinado numa altura do algoritmo em que já se percorreram os nós intermédios no conjunto $\{1, 2, \ldots, k\}$. Então o algoritmo percorre os caminhos de i para j com todos os possíveis nós intermédios, isto é, para todo o $k \in V$, tem-se

$$\pi^{(k)}(i,j) = \begin{cases} d_{ij} & \text{se } k = 0 \\ \min\left\{\pi^{(k-1)}(i,j), \pi^{(k-1)}(i,k) + \pi^{(k-1)}(k,j)\right\} & \text{se } k \geq 1 \end{cases}$$

Este método é resumido no Algoritmo 5.

Algoritmo 5 Algoritmo de Floyd-Warshall

```
Seja G = (V, E) uma rede
for i, j ∈ V do
    π(i, j) ← +∞
    ξ(i, j) ← −1
    π(j, j) ← 0
end for
for (i, j) ∈ E do
    π(i, j) ← d_ij
end for
for i, j ∈ V do
    if π(i, j) < +∞ then
        ξ(i, j) ← i
    end if
end for
for k, i, j ∈ V do
    if π(i, j) > π(i, k) + π(k, j) then
        π(i, j) ← π(i, k) + π(k, j)
        ξ(i, j) ← ξ(k, j)
    end if
end for
```

A complexidade do número de operações realizadas pelo algoritmo de Floyd-Warshall é $\mathcal{O}(n^3)$. Este valor é sempre pior do que o da complexidade no pior dos casos para o algoritmo de Johnson, podendo ser próximo do apresentado por este último se a rede for muito densa (ou seja, quando $m \approx n^2$).

A Tabela 2.4 resume um exemplo de aplicação deste método à rede da Figura 2.6. Neste exemplo temos um grafo fortemente conexo, de forma a podermos calcular

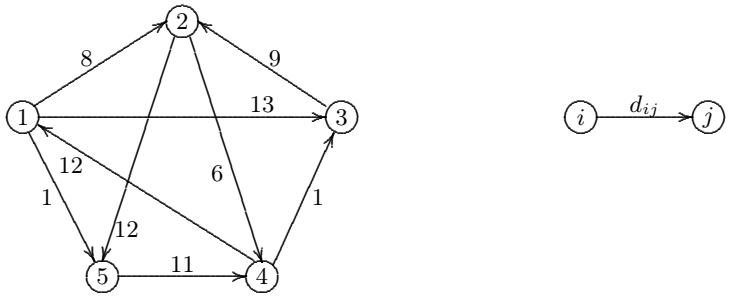


Figura 2.6: Rede

o caminho mais curto entre qualquer par de nós do grafo. O primeiro quadro é referente à inicialização, e são apresentados mais cinco quadros, um para cada uma das iterações do algoritmo. Na primeira iteração, $k = 1$, quando $i = 4$ e $j = 2$ temos $\pi(4,2) > \pi(4,1)+\pi(1,2)$, logo o valor de $\pi(4,2)$ é melhorado, passando a ser 20. Para $i = 4$ e $j = 5$, temos $\pi(4,5) > \pi(4,1) + \pi(1,5)$, então também há uma atualização de $\pi(4,2)$. Nas outras iterações também há rótulos que são atualizados. As árvores dos caminhos mais curtos entre todos os pares de nós encontram-se representadas na Figura 2.7.

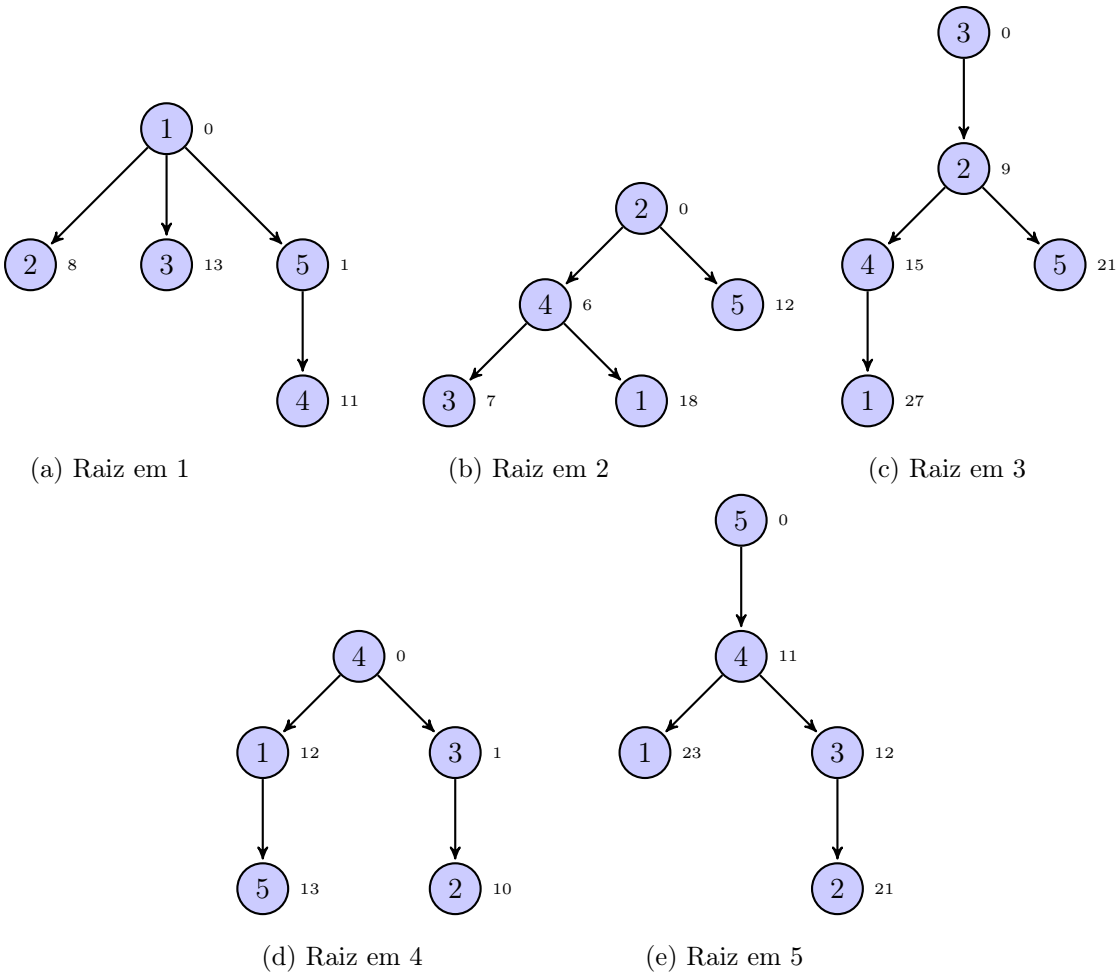
2.3. Testes Computacionais

Com o objetivo de avaliar empiricamente o comportamento dos métodos descritos para o problema dos caminhos mais curtos entre todos os pares de nós, foram testados alguns deles. Tendo em vista a aplicação destes métodos ao problema descrito no capítulo seguinte, supomos que os valores associados aos arcos da rede são todos não negativos.

Nestas condições o algoritmo de Johnson dispensa a fase de substituição das distâncias e coincide com a versão proposta inicialmente, de calcular os caminhos mais curtos com início num nó, variando o nó inicial pelos n nós da rede.

Foram implementados quatro códigos,

- a generalização do algoritmo de Bellman-Ford com lista FIFO, para n raízes, representada no que se segue por *APBF*,
- a generalização do algoritmo de Bellman-Ford com lista FIFO e verificação do pai, para n raízes, designada por *APBFPC*,
- a generalização do algoritmo de Dijkstra, para n raízes, designada por *APD* (e que, neste caso, corresponde a uma simplificação do algoritmo de Johnson),



(a) Raiz em 1 (b) Raiz em 2 (c) Raiz em 3

(d) Raiz em 4 (e) Raiz em 5

Figura 2.7: Árvores dos caminhos mais curtos com raiz em vários nós, na rede da Figura 2.6

- o algoritmo de Floyd-Warshal, representado por *APFW*.

Estes códigos foram implementados em linguagem Java e foram testados numa máquina com um processador Intel(R) Pentium(R) Dual CPU T3200 @ 2.00GHz e memória RAM 3.00 GB.

Para uma comparação do desempenho dos vários métodos em termos de tempo de execução, procedeu-se a testes em redes geradas aleatoriamente, tendo-se variado a dimensão. Resolveram-se 10 problemas de cada dimensão. Utilizaram-se redes com $n = 100, 500, 1000, 1500$ nós e densidades $a = 5, 7, 10$, isto é, $m = an = 5n, 7n, 10n$. As distâncias d_{ij} foram geradas uniformemente no intervalo $[0, 200]$, $(i, j) \in E$.

Os resultados médios (em segundos) obtidos para cada dimensão estão resumidos na Tabela 2.5. Os valores a negrito destacam o método mais eficiente para cada conjunto de problemas.

Na resolução deste conjunto de problemas destacaram-se as versões *APBF* e

APBFPC, que utilizam o algoritmo de Bellman-Ford com lista FIFO (com e sem verificação do pai), dado que foram as implementações que se mostraram mais eficientes em termos de tempo de execução. De acordo com os resultados na Tabela 2.5, na maioria dos casos, o método mais rápido foi o de Bellman-Ford com lista FIFO e utilizando verificação do pai.

Tabela 2.4: Aplicação do algoritmo de Floyd-Warshall à rede da Figura 2.6

	$\pi(i,j)$ $\xi(i,j)$	1		2		3		4		5	
	1	0	1	8	1	13	1	$+\infty$	-1	1	1
	2	$+\infty$	-1	0	2	$+\infty$	-1	6	2	12	2
	3	$+\infty$	-1	9	3	0	3	$+\infty$	-1	$+\infty$	-1
	4	12	4	$+\infty$	-1	1	4	0	4	$+\infty$	-1
	5	$+\infty$	-1	$+\infty$	-1	$+\infty$	-1	11	5	0	5
	$\pi(i,j)$ $\xi(i,j)$	1		2		3		4		5	
	1	0	1	8	1	13	1	$+\infty$	-1	1	1
$k=1$	2	$+\infty$	-1	0	2	$+\infty$	-1	6	2	12	2
	3	$+\infty$	-1	9	3	0	3	$+\infty$	-1	$+\infty$	-1
	4	12	4	20	1	1	4	0	4	13	1
	5	$+\infty$	-1	$+\infty$	-1	$+\infty$	-1	11	5	0	5
	$\pi(i,j)$ $\xi(i,j)$	1		2		3		4		5	
	1	0	1	8	1	13	1	14	2	1	1
$k=2$	2	$+\infty$	-1	0	2	$+\infty$	-1	6	2	12	2
	3	$+\infty$	-1	9	3	0	3	15	2	21	2
	4	12	4	20	1	1	4	0	4	13	1
	5	$+\infty$	-1	$+\infty$	-1	$+\infty$	-1	11	5	0	5
	$\pi(i,j)$ $\xi(i,j)$	1		2		3		4		5	
	1	0	1	8	1	13	1	14	2	1	1
$k=3$	2	$+\infty$	-1	0	2	$+\infty$	-1	6	2	12	2
	3	$+\infty$	-1	9	3	0	3	15	2	21	2
	4	12	4	10	3	1	4	0	4	13	1
	5	$+\infty$	-1	$+\infty$	-1	$+\infty$	-1	11	5	0	5
	$\pi(i,j)$ $\xi(i,j)$	1		2		3		4		5	
	1	0	1	8	1	13	1	14	2	1	1
$k=4$	2	18	4	0	2	7	4	6	2	12	2
	3	27	4	9	3	0	3	15	2	21	2
	4	12	4	10	3	1	4	0	4	13	1
	5	23	4	21	3	12	4	11	5	0	5
	$\pi(i,j)$ $\xi(i,j)$	1		2		3		4		5	
	1	0	1	8	1	13	1	12	5	1	1
$k=5$	2	18	4	0	2	7	4	6	2	12	2
	3	27	4	9	3	0	3	15	2	21	2
	4	12	4	10	3	1	4	0	4	13	1
	5	23	4	21	3	12	4	11	5	0	5

Tabela 2.5: Tempos médios de execução (em segundos)

n	$m = 5n$	APD	$APBF$	$APBFPC$	$APFW$
100	500	0,02143	**0,00665**	0,00806	0,01642
500	2500	1,33796	0,20549	**0,19775**	0,38837
1000	5000	10,80130	0,83628	**0,82955**	2,27756
1500	7500	43,77685	2,11090	**2,09696**	6,95475
n	$m = 7n$	APD	$APBF$	$APBFPC$	$APFW$
100	700	0,03257	0,00947	**0,00931**	0,01107
500	3500	2,31104	**0,30262**	0,30734	0,50873
1000	7000	17,79301	1,17531	**1,15007**	2,60224
1500	10 500	88,13362	3,07534	**3,04416**	7,95891
n	$m = 10n$	APD	$APBF$	$APBFPC$	$APFW$
100	1000	0,04767	0,01594	**0,01424**	0,01622
500	5000	3,11656	0,41445	**0,41442**	0,61051
1000	10 000	33,10510	**1,74165**	1,74356	3,22091
1500	15 000	170,70834	4,36594	**4,28247**	9,17498

Capítulo 3

Determinação de Caminhos Mais Curtos em Edifícios

No presente capítulo discutimos a aplicação dos métodos apresentados anteriormente ao cálculo de caminhos mais curtos no interior de edifícios e a sua articulação com outros algoritmos para localização em edifícios. A implementação computacional a integrar a plataforma *OnAll* constituiu uma forte componente do trabalho realizado. Assim sendo, sempre que se justifique, a apresentação teórica que se segue será complementada por um resumo das rotinas desenvolvidas.

3.1. Definição do Problema

Existem vários sistemas para localização, ou posicionamento, no exterior, sendo o *Global Positioning System* (GPS) o mais difundido de entre eles. Este sistema é baseado em mensagens enviadas por um conjunto de satélites, que incluem informação sobre a sua localização e sobre o instante em que a mensagem foi enviada. A informação recebida é utilizada para determinar a posição de um dado objeto. O sinal enviado por um satélite pode sofrer alterações ao encontrar obstáculos, como telhados ou paredes e, como tal, a utilização do sistema GPS em espaços fechados não permite a localização com a precisão pretendida. Habitualmente são utilizados métodos alternativos quando se pretende localizar algo, ou alguém, no interior de edifícios. Algumas das técnicas aplicadas nestas condições são conhecidas como *Pedestrian Dead Reckoning*, *Wi-Fi*, *Bluetooth* [8, 2]. O método *Pedestrian Dead Reckoning* baseia-se na utilização de informação sobre o caminhar humano. Essencialmente, são usadas estimativas do deslocamento de um indivíduo, para determinar a sua localização. As estimativas dependem dos passos e da direção de deslocamento do indivíduo, que são detetados através de sensores dispostos no indivíduo. Por sua vez, o funcionamento da técnica *Wi-Fi* depende da existência de vários pontos de acesso no edifício, que transmitem sinais continuamente. É fornecido um sensor a cada indivíduo, que permite receber os sinais emitidos pelos pontos de acesso e,

então, localizá-lo.

No contexto deste projeto pretende-se desenvolver uma plataforma integrada de fácil implementação e que seja capaz de localizar indivíduos dentro de um espaço fechado. A estratégia proposta consiste em utilizar dispositivos sem fios para efetuar a localização. O meio sem fios escolhido é o *Wi-Fi*. Com este propósito, o projeto contempla o estudo de dois métodos para a localização nestes ambientes, um baseado em otimização multiobjetivo e outro baseado em multilateração, apresentados nos trabalhos de Jorge e Pereira [11, 15]. Quase sempre os algoritmos utilizados para a determinação de uma localização podem ser afetados pelo ruído dos dados recebidos, resultante da oscilação do sinal emitido pelos dispositivos eletrónicos envolvidos e que são muito sensíveis a obstáculos dentro dos edifícios. Neste capítulo apresentamos uma ferramenta complementar aos métodos de localização, cujo objetivo é eliminar, ou pelo menos reduzir, eventuais erros que possam ser produzidos pelos algoritmos multiobjetivo ou multilateração. Mais concretamente, pretendemos desenvolver algoritmos baseados no problema do caminho mais curto, a aplicar após o cálculo de várias possíveis posições de um indivíduo pelos métodos anteriores, e que permitam identificar algumas posições como inválidas. A identificação é conseguida com base no cálculo de limites inferiores relativamente ao tempo que um utilizador do sistema poderá levar para se deslocar entre duas dadas posições. Se o caminho mais curto entre esses dois locais demorar mais tempo a percorrer do que o tempo decorrido entre as duas localizações consecutivas, então concluímos que o indivíduo não se pode encontrar ali. Como veremos, isto passará por encontrar o caminho mais curto (isto é, um caminho com a menor distância) entre quaisquer dois locais do edifício.

O caso de estudo que acompanha este trabalho é um dos edifícios da residência assistida TrofaSenior Residências, situada em Alfena. A Figura 3.1 apresenta a planta de um dos pisos do edifício principal desta instituição, que utilizaremos para ilustrar várias facetas do trabalho.

3.2. Representação do Edifício

Em trabalhos que envolvem a navegação em edifícios fechados é frequente encontrar um edifício representado por um grafo de vizinhanças, que associa o ponto médio de cada divisão a um nó, estando este, por sua vez, ligado a todos os nós que correspondem a divisões diretamente acessíveis a partir da primeira [12, 13, 14, 20]. Por vezes também se considera que corredores, escadas ou elevadores são um nó do

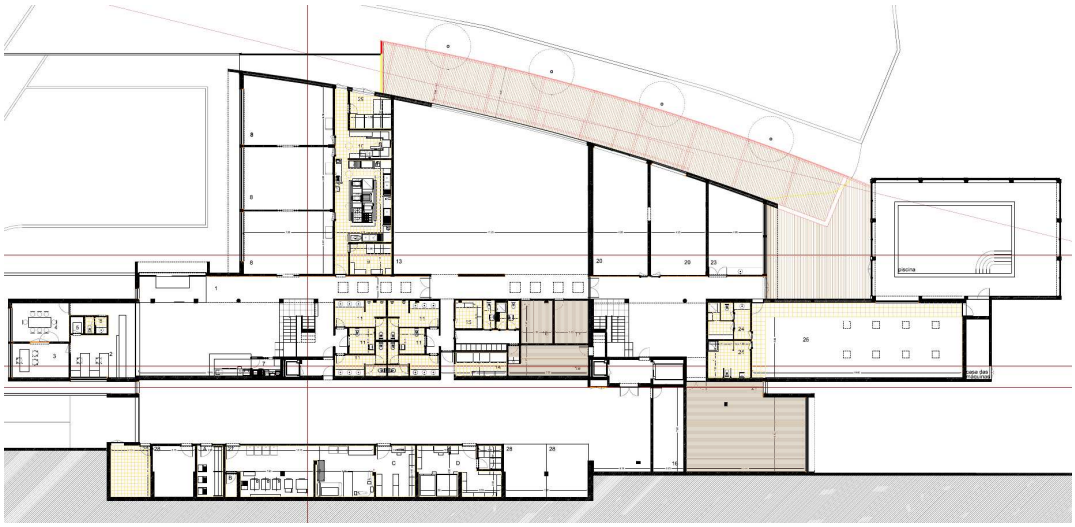


Figura 3.1: Planta do nível 1 do edifício principal da TrofaSenior Residências

grafo. Outros modelos não têm em conta a geometria do espaço interior e, por isso, calculam o caminho mais curto fornecendo apenas uma sequência de identificadores sem instruções muito detalhadas. Desta forma, não são capazes de calcular a distância entre dois locais sem introduzir alguns dados manualmente. Alguns modelos ignoram certas restrições, como as portas e desta forma os caminhos não são precisos o suficiente para utilizar de modo prático.

Em [14] propõe-se dividir a planta de um edifício em áreas, onde as transições entre divisões (portas) se representam por áreas de pequenas dimensões e as áreas sem transições por áreas de maiores dimensões. Neste caso, a utilização do grafo permite calcular, para cada passo do utilizador, a posição em que ele se encontra na área de um determinado nó.

Temos como primeiro objetivo definir um grafo que permita calcular o caminho mais curto entre quaisquer dois locais, tendo em conta a geometria do espaço. No caso em análise, atravessar uma divisão está associado a um percurso, com uma determinada distância, e portanto um determinado tempo. Assim, será mais interessante definir um grafo $G = (V, E)$, cujos nós representam pontos de transição entre divisões e cujos arcos representam ligações entre pontos de transição que sejam diretamente acessíveis entre si. Para simplificar a linguagem designaremos os pontos de transição apenas por *portas*. Para cada nó v do grafo são conhecidas as duas divisões que a porta correspondente, v, delimita.

Na planta representada na Figura 3.2 cada divisão está identificada a azul. Além disso, as portas entre divisões distintas estão numeradas com o valor indicado a

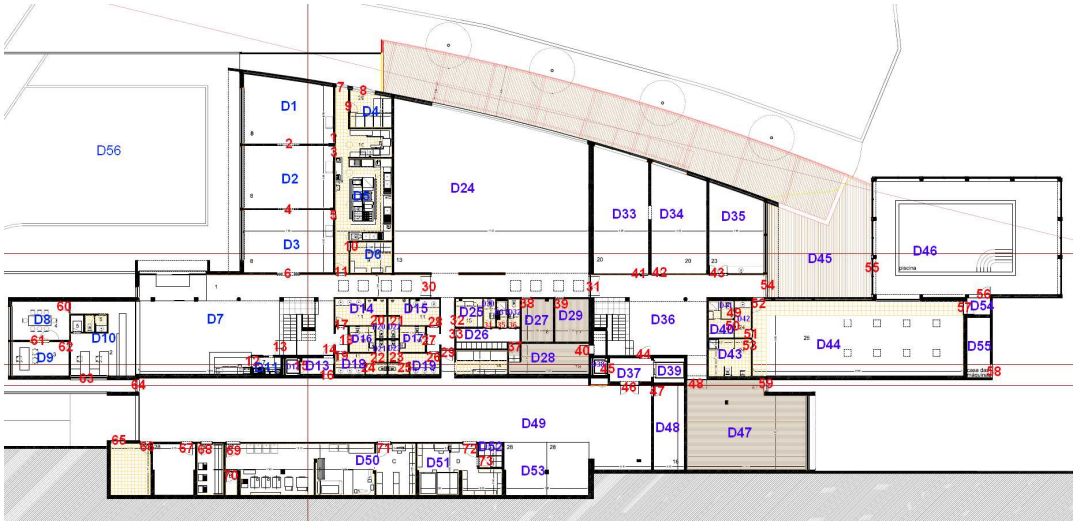
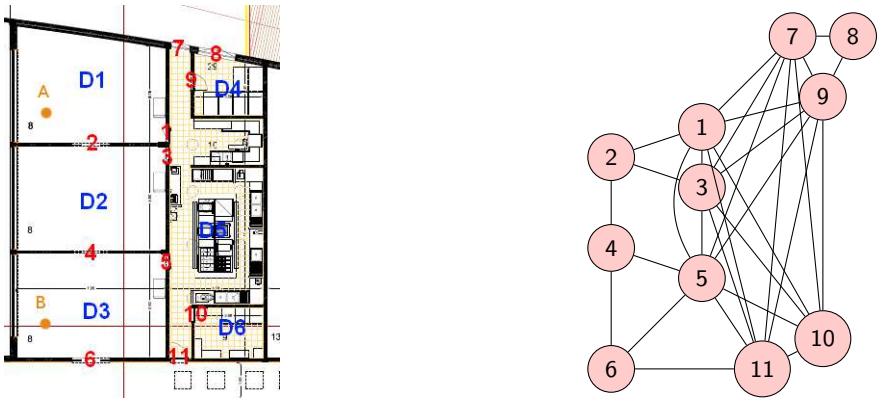


Figura 3.2: Planta do nível 1 do edifício principal da TrofaSenior Residências: com identificação de portas e divisões

vermelho. A Figura 3.3a mostra apenas uma parte da planta da Figura 3.2, enquanto que o grafo correspondente a esta planta é apresentado na Figura 3.3b.



(a) Parte da planta

(b) Grafo associado

Figura 3.3: Planta parcial do nível 1 do edifício principal da TrofaSenior Residências: representação usando um grafo

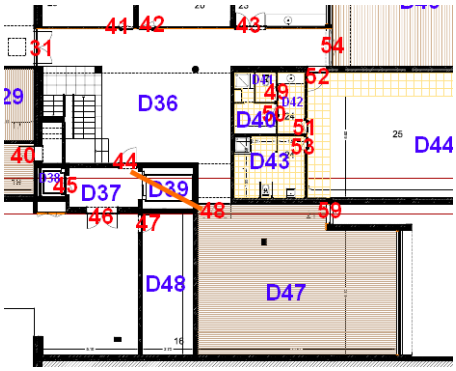
Ao grafo definido anteriormente acresce que, a cada aresta (i, j) se associa d_{ij}, um valor estimado da distância entre i e j. Pretendemos calcular um limite inferior para o tempo que um indivíduo pode demorar a percorrer o caminho mais curto entre quaisquer dois locais do edifício. Este limite terá sempre várias condicionantes, tais como a velocidade a que o indivíduo se desloca, os obstáculos que se podem encontrar ao longo do percurso, além das paredes do edifício, etc. Por outro lado, este limite deve ser calculado da forma mais rigorosa possível, de modo a obter uma

boa aproximação, isto é, uma aproximação realista, do tempo mínimo que a pessoa demora para se deslocar entre dois pontos. De facto, se o valor encontrado for muito inferior ao real corremos o risco de, na prática, não ser possível descartar soluções que estejam erradas. Por outro lado, se o limite for muito relaxado pode levar à aceitação de soluções incorretas e levará a falsos positivos.

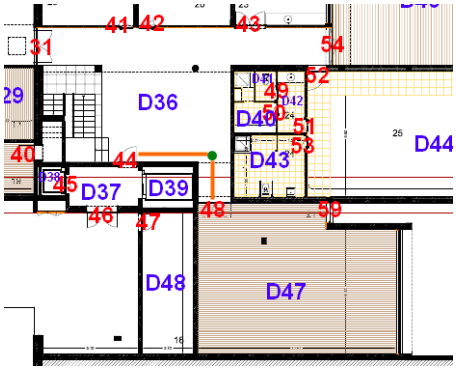
Os valores associados aos arcos do grafo são calculados usando a distância euclidiana da ligação entre as portas associadas a um par de nós. É de notar que optar por esta métrica implica desprezar possíveis obstáculos dentro das divisões, como mobília ou outras pessoas. Admitimos que, muito mais do que a estrutura do edifício propriamente dita, estes fatores podem ser alterados ao longo do tempo e que, portanto, não serão contemplados no grafo. A simples distância euclidiana tem outras limitações, por exemplo, se considerarmos um par de portas numa divisão não convexa. Formalmente um conjunto X diz-se *convexo* se, dados quaisquer dois pontos de X, o segmento de reta que os liga continua a estar em X. Neste caso diremos, informalmente, que duas portas correspondentes a esses pontos não são *visíveis entre si*. Tomar como distância entre essas duas portas a distância euclidiana entre os dois pontos pode introduzir um erro considerável. Para melhorar este valor podem definir-se pontos auxiliares, que não correspondem a portas e portanto não correspondem a nós do grafo, mas que permitem definir um percurso entre duas portas em que todos os pontos consecutivos são visíveis entre si.

A situação descrita é ilustrada na Figura 3.4. As portas identificadas com os números 44 e 48 são acessíveis diretamente, no entanto não são visíveis uma a partir da outra. Considerando apenas a distância euclidiana entre ambas corresponde a usar o percurso marcado a laranja na Figura 3.4a, e que sabemos não ser admissível. Em alternativa podemos acrescentar um ponto artificial intermédio, entre as portas número 44 e número 48, que se encontra marcado a verde na Figura 3.4b. Neste caso a distância estimada entre estas portas corresponde a usar o percurso a laranja nesta figura e fornece um valor mais correto do que o obtido com o primeiro processo.

É de realçar que a estrutura descrita é compatível com a representação de edifícios com mais do que um nível. Para tal basta considerar que os pontos que permitem fazer a transição entre pisos consecutivos, tais como elevadores ou escadas, são nós do grafo, ou seja, são "portas" que separam duas divisões vizinhas. Desta forma teremos apenas um grafo para todo o edifício e poderemos considerar qualquer caminho entre dois pontos do edifício. Mais concretamente, o local de acesso a cada um desses



(a) Distância euclidiana simples



(b) Distância euclidiana corrigida

Figura 3.4: Planta parcial do nível 1 do edifício principal da TrofaSenior Residências: distância entre portas não visíveis

pontos em cada piso é identificado por um nó. O par de nós que representa os extremos de uma tal transição entre pisos é ligado por dois arcos, um no sentido ascendente e outro no sentido ascendente. De acordo com o procedimento anterior, o valor associado a estes arcos deve estimar a distância a percorrer para transitar entre pisos. No entanto, este caso é mais dependente do modo como a transição é realizada do que da distância euclidiana entre os dois pontos no espaço. Assim sendo, optámos por arbitar valores adequados no momento em que a rede é construída. Nomeadamente, deve ser levado em conta que os tempos de subida e de descida não são os mesmos, pelo que, como se referiu, os arcos são orientados e a cada um devem ser associados valores diferentes.

3.2.1. Implementação

Existem várias estruturas de dados para representar uma rede, tendo em vista a sua utilização em termos computacionais. Entre elas encontramos listas de adjacência e matrizes de adjacência ou de incidência, sendo que estas últimas podem ser pouco

eficientes relativamente ao espaço de memória que ocupam [1].

Na representação por lista de adjacência, dado um grafo $G = (V, E)$, define-se um vetor Adj de n listas. Estas listas denominam-se *listas de adjacência*. Este vetor é indexado pelos nós do grafo e, para cada $u \in V$, contém todos os nós que lhe são adjacentes, isto é, contém os nós $v \in V$ tais que $(u, v) \in E$. Para redes, armazena-se adicionalmente a distância de cada arco, juntamente com o seu nó sucessor. Se o grafo G for orientado, então a soma dos comprimentos de todas as listas de adjacência é m. Caso o grafo seja não orientado, esta soma é $2m$, pois para cada aresta $\{i, j\}$ armazenam-se os arcos (i, j) e (j, i). Em termos de memória esta representação usa um espaço de $\mathcal{O}(n + m)$.

Na representação por matriz de adjacência, dado um grafo $G = (V, E)$, define-se uma matriz $A = (a_{ij})$, $n \times n$, onde

$$a_{ij} = \begin{cases} 1 & \text{se } \{i, j\} \in E \\ 0 & \text{caso contrário} \end{cases}$$

Se o grafo subjacente é orientado os valores podem ser modificados para

$$a_{ij} = \begin{cases} 1 & \text{se } (i, j) \in E \\ -1 & \text{se } (j, i) \in E \\ 0 & \text{caso contrário} \end{cases}$$

Para redes, a matriz de adjacência pode armazenar a distância de cada arco, ao invés do valor 1.

O espaço de memória ocupado por esta representação é de $\mathcal{O}(n^2)$. Note-se que esse é o limite para a representação anterior, no caso em que a rede é completa.

Ao utilizar uma matriz de incidência, dado um grafo $G = (V, E)$, define-se uma matriz $B = (b_{ik})$, $n \times m$. Cada uma das linhas de B corresponde a um nó $i \in V$ e cada coluna corresponde a uma aresta $a_k = (i_k, j_k) \in E$. Então,

$$b_{ik} = \begin{cases} 1 & \text{se } (i, j_k) \in E \\ -1 & \text{se } (j_k, i) \in E \\ 0 & \text{caso contrário} \end{cases}$$

Em termos de memória esta representação ocupa $\mathcal{O}(nm)$.

Outra possível representação de uma rede consiste em usar a forma dos arcos emergentes (*forward star form* no original) [6]. Esta forma baseia-se em três vetores para representar uma rede $G = (V, E)$, $point \in \mathbb{N}^{n+1}$, $suc \in \mathbb{N}^m$, e $dist \in \mathbb{N}^m$. Os

arcos da rede são numerados de 1 a m. Se o arco $(i,j) \in E$ é o k-ésimo arco da rede, então define-se

$$\begin{cases} suc(k) &= j \\ dist(k) &= d_{ij} \end{cases}$$

A informação fornecida pelos vetores *suc* e *dist* é complementada pelo vetor *point*, definido recursivamente por

$$\begin{cases} point(1) &= 1 \\ point(i+1) &= point(i) + \text{número de arcos que saem de } i,\ i = 1, \ldots, n \end{cases}$$

Note-se que $point(n+1) = m+1$ e que, em termos de espaço de memória ocupado, esta representação tem $\mathcal{O}(m+n)$. A representação de uma rede na forma dos arcos emergentes permite aceder diretamente a todos os arcos que saem de um dado nó. Uma vez que pretendemos calcular caminhos mais curtos, esta será a forma mais conveniente para armazenar a rede.

A partir da planta do edifício em causa é construída uma lista dos nós do grafo correspondente, que armazena para cada nó, as duas divisões a que se pode aceder a partir de cada "porta". É também construída uma lista de arcos, que armazena o nó inicial, o nó terminal e a distância de cada arco. Esta informação encontra-se resumida na Classe 6.

Classe 6 Classe *Graph*

Nós
1 - Div_{11}, Div_{12}
2 - Div_{21}, Div_{22}
$\vdots$
n - Div_{n1}, Div_{n2}
Arestas
$(i_1, j_1, d_{i_1 j_1})$
$(i_2, j_2, d_{i_2 j_2})$
$\vdots$
$(i_m, j_m, d_{i_m j_m})$

Assim sendo, é necessário proceder à conversão da estrutura fornecida para a forma dos arcos emergentes. O método utilizado para este efeito é resumido no Algoritmo 7. Para transformar a rede dada numa rede orientada na forma dos arcos emergentes, cada aresta $\{i,j\}$ da rede não orientada é substituída por dois arcos na rede orientada, (i,j) e (j,i), o que é realizado no primeiro ciclo apresentado no pseudo-código. Note-se que no inicio do algoritmo, m representa o número de arcos

da rede não orientada. Mas na primeira instrução, m passa a ser o número de arcos da rede orientada.

Algoritmo 7 Conversão da rede dada na forma dos arcos emergentes

$m \leftarrow 2m$
for i ímpar de 1 até m **do**
  $aresta \leftarrow$ próxima $aresta$
  $tail(i) \leftarrow$ primeiro nó de $aresta$
  $head(i) \leftarrow$ segundo nó de $aresta$
  $cost(i) \leftarrow$ distância de $aresta$
  $tail(i+1) \leftarrow$ segundo nó de $aresta$
  $head(i+1) \leftarrow$ primeiro nó de $aresta$
  $cost(i+1) \leftarrow$ distância de $aresta$
end for
for i de 1 até n **do**
  $aux(i) \leftarrow 0$
end for
for i de 1 até m **do**
  $aux(tail(i)) \leftarrow aux(tail(i)) + 1$
end for
$point(1) \leftarrow 1$
for i de 1 até n **do**
  $point(i+1) \leftarrow point(i) + aux(i)$
  $aux(i) \leftarrow 0$
end for
for i de 1 até m **do**
  $x \leftarrow tail(i)$
  $suc(point(x) + aux(x)) \leftarrow head(i)$
  $dist(point(x) + aux(x)) \leftarrow cost(i)$
  $aux(x) \leftarrow aux(x) + 1$
end for

É ainda de notar que o vetor aux utilizado no Algoritmo 7 começa por armazenar o número de arcos que saem de i, para cada $i \in V$, e que permite a construção do vetor $point$. Posteriormente o vetor aux auxilia a definição suc e $dist$ por forma a considerar os arcos ordenados lexicograficamente.

3.3. Problema do Caminho Mais Curto num Edifício

Nesta secção será descrito o método utilizado para determinar o caminho mais curto entre quaisquer dois pontos dados no interior de um edifício.

Começamos por observar que os pontos inicial e terminal do percurso ao longo do edifício podem não corresponder a nós do grafo definido na secção anterior, isto

é, podem não ser portas. O resultado pretendido é formado por um percurso que liga o ponto inicial a uma porta, seguido de outro até uma outra porta, e ainda de um terceiro que liga esta porta ao ponto terminal. Como exemplo consideremos que um indivíduo se pretende deslocar do ponto A para o ponto B, representados na Figura 3.5. Aparentemente a melhor solução é a definida pela sequência $\langle A, 2, 4, B \rangle$, ilustrada pelas linhas amarelas e verde na Figura 3.5.

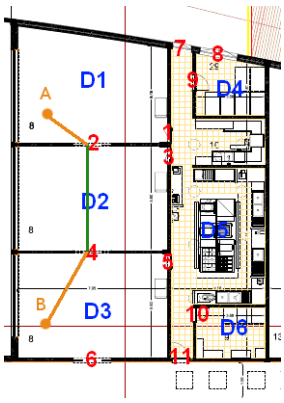


Figura 3.5: Planta parcial do nível 1 do edifício principal da TrofaSenior Residências: caminho entre dois pontos

Problemas do caminho mais curto em estradas, em que surgem estradas de pelo menos dois tipos diferentes têm sido estudados na literatura. Neste contexto destacamos o trabalho de Schultes [16], que, face à elevada dimensão da rede, propõe uma classificação hierárquica entre as estradas, uma contração da rede de acordo com essa hierarquia e uma resolução faseada do problema, com o intuito de devolver uma solução rapidamente. Como referimos, também o caminho mais curto num edifício apresenta dois tipos distintos de troços e, portanto, a sua determinação recorre a duas fases. Uma das fases é dedicada à determinação dos caminhos mais curtos entre qualquer par de portas do edifício, ou seja, na rede que representa o edifício (designada por Fase 1). A outra fase foca a determinação do caminho mais curto entre um par de locais (designada por Fase 2).

O problema a resolver na Fase 1 reduz-se ao problema do caminho mais curto entre todos os pares de nós, na rede que representa o edifício. Como tal, qualquer dos métodos descritos na Secção 2.2.2 pode ser aplicado para resolver o problema. Uma vez que os valores associados aos arcos da rede representam distâncias entre dois locais no edifício, então todos são não negativos. Isto implica que a substituição dos valores iniciais proposta por Johnson e descrita por (2.1) pode ser evitada. Nestas condições o algoritmo de Johnson pode ser simplificado para a abordagem dita

ingénua, em que é aplicado o algoritmo de Dijkstra.

Os tempos de execução apresentados na Secção 2.3 servem como referência para o tempo de execução que poderá exigir a resolução da a Fase 1, ressalvando que, na prática, o número de nós não deverá exceder 500, uma vez que se refere ao número de "portas" de um edifício. Por fim, notamos que estes caminhos são estáticos (admitindo que a estrutura do edifício se mantém) e que farão parte da solução encontrada na Fase 2. Assim sendo, nada impede que sejam calculados *a priori* e armazenados até que essa informação seja necessária, o que permite que o único cálculo efetuado em tempo real, seja aquele que envolve a Fase 2.

A Fase 2 depende dos pontos inicial e terminal recebidos num determinado instante. Retomemos o exemplo da determinação do caminho mais curto entre os pontos A e B, na Figura 3.5. Suponhamos que são conhecidas as portas das divisões em que A e B se encontram, assim como o caminho mais curto entre qualquer par dessas portas (informação obtida na Fase 1). Conhecendo ainda a distância de A e de B a cada uma dessas portas, podemos combinar todas as ligações possíveis, de forma a obter os caminhos de A para B. No exemplo, as portas da divisão onde está assinalado o ponto A são numeradas com 1 e 2, e as da divisão onde se encontra B são as portas 4, 5 e 6. Então, as ligações possíveis para um indivíduo se deslocar de A para B são representadas pelas sequências:

- $\langle A, 1, 3, 4, B\rangle$
- $\langle A, 1, 5, B\rangle$
- $\langle A, 1, 3, 4, 6, B\rangle$
- $\langle A, 2, 4, B\rangle$
- $\langle A, 2, 3, 5, B\rangle$
- $\langle A, 2, 4, 6, B\rangle$

Comparando as distâncias destas soluções obtemos o caminho mais curto entre A e B, que, como havíamos observado, é $\langle A, 2, 4, B\rangle$.

Do exposto concluímos que a Fase 2 pode ser resolvida comparando as distâncias entre os pontos inicial e terminal e as portas das divisões em que se encontram, combinadas com as distâncias entre cada par dessas portas em divisões distintas. As primeiras distâncias mencionadas (às portas das divisões inicial e terminal) podem

ser estimadas utilizando a distância euclidiana. As segundas (entre duas portas) são obtidas recorrendo aos resultados da Fase 1. A comparação entre as distâncias dos caminhos encontrados desta forma deve ser exaustiva, pelo que o número de caminhos a considerar é dado pelo número de portas na divisão inicial a multiplicar pelo número de portas na divisão terminal.

Conclui-se então que teremos uma complexidade $\mathcal{O}(n^2m)$, em termos de número de operações, para resolver o problema do caminho mais curto num edifício.

3.3.1. Implementação

Como referimos, o problema relativo à Fase 1 pode ser resolvido recorrendo a qualquer dos algoritmos descritos na Secção 2.2.2. Optámos por usar o algoritmo de Bellman-Ford implementado com lista FIFO e com verificação do pai, face aos resultados apresentados na Tabela 2.5 pelo código *APBFPC*.

Depois de calculado o caminho mais curto entre qualquer par de portas, é calculado o caminho mais curto entre qualquer par de pontos. Para resolver este último problema de forma eficiente é conveniente conhecer as divisões onde se encontram os pontos inicial e terminal, assim como conhecer as portas dessa divisão. A divisão e o piso são fornecidos pelo sistema, bem como o par de divisões a que cada porta está associada, isto é, as divisões a que se pode aceder a partir de cada porta. Com esta informação foi criada uma classe, *Divisao*, descrita na Classe 8, que armazena uma divisão e uma lista com as suas portas. Para consultar as portas de cada uma das divisões foi criada uma lista de *Divisao*, a que chamamos a classe *GrafoDivisoes*, descrita na Classe 9.

Classe 8 *Divisao*

int divisao
Lista <int> portas

Classe 9 *GrafoDivisoes*

Lista<Divisao> grafoDiv

Para a construção da classe *GrafoDivisoes* é utilizado o Algoritmo 10, em que se percorrem os nós da rede e para cada um se verifica se alguma das suas divisões já foi armazenada na rede. Em caso negativo, acrescenta-se essa divisão e a porta correspondente. Em caso afirmativo, adiciona-se apenas a porta. No final obtemos a lista de divisões, *grafoDiv*, que armazena as portas de cada divisão.

Algoritmo 10 Construção de *GrafoDivisoes*

```
nPorta ← 1
pos1 ← 1
pos2 ← 1
while Houver nós por analisar do
  no ← próximo nó
  encontraDiv1 ← falso
  encontraDiv2 ← falso
  i ← 1
  while i ≤ |grafoDiv| e não encontraDiv1 ou não encontraDiv2 do
    for k = 1, 2 do
      if não encontraDivk e divk de no = i-ésima divisão de grafoDiv then
        encontraDivk ← verdadeiro
        posk ← k
      end if
      i ← i + 1
    end for
  end while
  for k = 1, 2 do
    if não encontraDivk then
      adiciona divk do no no fim da lista grafoDiv
      adiciona a porta nPorta na última divisão adicionada
    else
      adiciona a porta nPorta na divisão posk
    end if
  end for
  nPorta ← nPorta + 1
end while
```

Finalmente, para calcular o caminho mais curto entre qualquer par de pontos do edifício usa-se o Algoritmo 11. São-nos dadas a última localização calculada, *ultimaLoc* e a localização atual, *locAtual*. Se as duas localizações se encontrarem na mesma divisão, calcula-se a distância euclidiana entre os dois pontos dados. Se estiverem em localizações diferentes, calcula-se a distância euclidiana da última localização até uma das portas dessa localização, consulta-se a distância dessa porta até uma das portas da localização atual, que já foi calculada na Fase 1, e a distância euclidiana dessa porta até à localização atual. Juntando-se estas distâncias, obtêm-se as distâncias da última localização até à atual. Para saber qual o caminho mais curto entre estes dois pontos, basta verificar qual o que tem a menor distância.

Algoritmo 11 Cálculo do caminho mais curto entre *ultimaLoc* e *locAtual*

```
nU ← número de nós de ultimaLoc
nA ← número de nós de locAtual
if divAtual = ultDiv then
  distMin ← distância euclidiana entre ultimaLoc e locAtual
else
  criar GrafoDivisoes
  porta_i_min ← porta 1 de ultimaDiv
  porta_j_min ← porta 1 de divAtual
  distaMinima ← ∞
  for i de 1 até nU do
    for j de 1 até nA do
      porta_i ← porta i de ultimaDiv
      porta_j ← porta j de ultimaDiv
      dist ← distância euclidiana de ultimaLocalizacao a porta_i
      dist ← dist+ distância da porta_i à porta_j
      dist ← dist+ distância euclidiana de porta_j a locAtual
      if dist < distMin then
        porta_i_min ← porta_i
        porta_j_min ← porta_j
        distMin ← dist
      end if
    end for
  end for
end if
```

3.4. Limites Inferiores

O objetivo de determinar o caminho mais curto entre um par de pontos de um edifício, e em particular a sua distância, é o de utilizar esta informação para poder verificar se uma dada posição não é coerente com a posição definida como correta no instante imediatamente anterior. Para este efeito é necessário aproximar o tempo associado ao caminho mais curto entre as duas posições. Uma vez que cada caminho está associado a uma distância a percorrer, como descrito na Secção 3.2, há que relacionar este valor com o tempo.

A tradução do espaço percorrido em termos de tempo pode ser conseguida com base na velocidade média de deslocamento do indivíduo a localizar. No contexto da utilização da plataforma *OnAll* em residências assistidas esta velocidade média pode variar, dependendo do utilizador em causa. Por exemplo, é de esperar que, em geral, um prestador de cuidados se desloque mais rapidamente do que um idoso. Para ajustar o procedimento a utilizar à residência assistida, o sistema distingue cinco

perfis de utilizadores distintos: cuidadores, auxiliares e idosos, sendo este último perfil subdivido noutros três: utilizadores independentes, utilizadores dependentes e utilizadores com alterações comportamentais. Cada um destes perfis tem diferentes tipos de mobilidade e consequentemente é associado a diferentes velocidades médias de deslocação. Os valores médios de velocidade utilizados na implementação baseiam-se em literatura consultada sobre a velocidade de pessoas em andamento dentro e fora de edifícios, em várias faixas etárias [3, 17, 19].

Em [17] é referido que a velocidade média de uma pessoa entre os 40 e os 59 anos não chega a 3 m/s, sendo os valores médios à volta de 2 m/s. De acordo com o estudo apresentado em [3], a velocidade máxima de uma pessoa entre os 20 e os 50 anos pouco ultrapassa os $2,5$ m/s, sendo os valores médios aproximadamente $1,4$ m/s. Em [19] é indicada a velocidade máxima de uma pessoa entre os 30 e os 59 anos indicada é $3,12$ m/s, supondo que o edifício se encontra ocupado por alguns obstáculos, que é o caso. O valor médio da velocidade é de cerca de 2 m/s. Com base nesta informação, e tendo em conta que pretendemos eliminar possíveis erros na localização, nomeadamente quando a distância percorrida excede o esperado, optámos por admitir que a velocidade média é a menor, de entre as referidas na literatura, ou seja, $1,4$ m/s [3].

Relativamente a indivíduos séniores, [17] refere que a velocidade máxima observada pouco ultrapassa os 2 m/s. Em [3] a velocidade máxima registada é de aproximadamente 2 m/s, enquanto que a média é de aproximadamente $1,3$ m/s. Por fim, as velocidades máxima e média indicadas em [19] são de cerca de 2 m/s e de $1,7$ m/s, respetivamente. Desta forma, para os idosos com mobilidade optámos por utilizar $1,3$ m/s como valor de referência.

Para os perfis cuidadores e auxiliares a velocidade utilizada é $1,4$ m/s, e para os perfis idosos independentes e idosos com alterações comportamentais é utilizada $1,3$ m/s. Relativamente ao perfil idosos dependentes, parece razoável considerar o valor 1 m/s, dado que terão de ser acompanhados por um cuidador ou um auxiliar. Sempre que for adequado, estes valores de referência poderão ser ajustados diretamente na plataforma *OnAll*.

Suponhamos que uma pessoa se encontra no ponto A e passados 10 segundos é localizada no ponto B, assinalados na Figura 3.5. Como já vimos o caminho mais curto de A para B é o caminho $\langle A, 2, 4, B\rangle$. Suponhamos que a distância deste caminho são 6 metros. Se esta pessoa se tratar de um prestador de cuidados

(cuidador ou auxiliar), significa que se consegue deslocar a uma velocidade média de 1,4 m/s. Então, demora $1,4 \times 6 = 8,4$ segundos a deslocar-se de do ponto A para o ponto B. Isto significa que o prestador de cuidados teve tempo suficiente para fazer esta deslocação, logo a solução B é aceite.

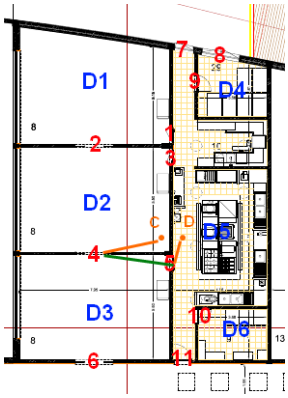


Figura 3.6: Parte da Planta do nível 1 do edifício principal da TrofaSenior Residências: caminho entre dois pontos quaisquer

Suponhamos agora que um prestador de cuidados se encontra no ponto C e passados 2 segundos é localizada no ponto D, assinalados na Figura 3.6. O caminho mais curto de C para D é o caminho $\langle C, 4, 5, D \rangle$. Supondo que a distância deste caminho são 4 metros, esta pessoa demora $1,4 \times 4 = 2,6$ segundos a deslocar-se do ponto C para o D. Logo, o indivíduo não teve tempo suficiente para fazer esta deslocação, por isso a solução D não é validada.

3.4.1. Implementação

Os vários pontos que vão sendo localizados podem ser validados ou não. Dessa forma foi criada uma lista, *historicoLoc*, que guarda os pontos que foram validados.

O algoritmo de validação está resumido no Algoritmo 12. Inicialmente é registado o instante em que uma posição é recolhida. Se a lista que guarda os pontos validados estiver vazia, a localização é aceite, caso contrário verifica-se se esta é válida. Se a posição recolhida se encontrar na mesma divisão que a posição anterior, é simplesmente calculada a distância euclidiana entre os dois pontos. Caso contrário percorrem-se as portas da divisão da posição anterior e da posição atual, calculando-se as correspondentes distâncias euclidianas. Procede-se do mesmo modo relativamente à posição atual e às portas da divisão em que esta se encontra. Percorrendo todos os pares de portas da divisão anterior e da divisão atual e consultando os valores da distância entre elas na tabela determinada na Fase 1, obtém-se a distância mínima.

Por fim, calcula-se o tempo necessário para a deslocação e rejeita-se a localização encontrada ou valida-se, acrescentando-se à lista de posições.

Algoritmo 12 Validação

```
tempoAtual ← tempo atual
locAtual ← localização dada pelos algoritmos de localização
if historicoLocalizacao = ∅ then
  adicionar locAtual a historicoLoc
  return verdadeiro
else
  ultimaLoc ← último elemento de historicoLoc
  calcular o caminho mais curto de ultimaLoc a locAtual
  difTempo ← tempoAtual − ultimoTempo
  tempoCaminhoMaisCurto ← distMin/velocidade
  if difTempo < tempoCaminhoMaisCurto then
    return falso
  else
    adicionar locAtual a historicoLoc
    return verdadeiro
  end if
end if
```

Capítulo 4

Conclusão

Tínhamos como objetivo apresentar uma ferramenta que permitisse reduzir possíveis erros produzidos pelos algoritmos de localização dentro do edifício. Esta ferramenta teria por base os limites inferiores do tempo que um indivíduo demora a percorrer um certo percurso no interior do edifício.

Começou-se por rever alguns algoritmos que permitem resolver o problema do caminho mais curto e fez-se um estudo que permitiu concluir que o mais eficiente era o algoritmo de Bellman-Ford com verificação do pai.

Em seguida apresentou-se o problema principal e uma sugestão de resolução do mesmo. Começou por se definir um grafo que representa o edifício. Os nós do grafo representam os pontos de transição entre divisões a que chamamos portas e os arcos representam todas as possíveis ligações diretas entre portas. Desta forma, para determinar o caminho mais curto entre dois locais do edifício dividiu-se o problema em duas fases. Na primeira fase calculou-se o caminho mais curto entre todos os pares de portas, usando o algoritmo que os testes revelaram ser o mais eficiente. Na segunda fase apresentou-se um algoritmo que permite determinar o caminho mais curto entre qualquer par de locais: para cada um dos dois locais identificaram-se as portas das divisões em que se encontram. Depois apresentaram-se todos os possíveis caminhos entre portas, cujas distâncias já tinham sido calculadas na fase anterior. Por fim, foi só calcular a distância de cada um dos pontos às portas das divisões, interligá-la com a distância entre portas e selecionar o caminho mais curto.

Dado que a finalidade da plataforma é ser usada numa unidade de cuidados continuados, distinguiram-se cinco perfis de utilizadores: dois para prestadores de cuidados e três para idosos e definiram-se as velocidades médias para cada um deles de acordo com bibliografia especializada.

Foram realizados alguns testes preliminares ao produto resultante da integração dos algoritmos de localização e o cálculo dos limites inferiores. Os resultados obtidos foram semelhantes aos apresentados nos exemplos no final do Capítulo 3, sendo

algumas localizações validadas como viáveis e outras rejeitadas por inviabilidade. Ao realizar estes testes observou-se que muitas localizações erradas não eram rejeitadas. O objetivo é encontrar o limite do tempo de deslocação que modele bem a realidade, de forma a poder rejeitar e aceitar as soluções o mais acertadamente possível. Em consequência ajustou-se a velocidade do prestador de cuidados para 1,1 m/s. No entanto, deverão realizar-se testes mais completos para poder tirar conclusões mais robustas.

Outro aspeto a implementar futuramente está relacionado com a identificação dos prestadores de cuidados mais próximos de um determinado local. Quando um idoso necessita de ajuda pretende-se identificar quais os prestadores de cuidados que mais rapidamente o podem acorrer, assim como a trajetória que este deve percorrer para alcançar o idoso. Esta solução permitirá otimizar o tempo dispendido pelos cuidadores e auxiliares, bem como reduzir o tempo de espera pelos idosos em situações de emergência. O processo a desenvolver para identificar o prestador de cuidados que se encontra mais próximo de uma determinada ocorrência será baseado no primeiro e no armazenamento de uma lista com as localizações de todos os utilizadores do sistema, assim como dos seus perfis diferenciados.

Bibliografia

[1] R. K. Ahuja, T. L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] N. Alves. Uma solução para navegação indoor. Master's thesis, Universidade do Minho, outubro 2012.

[3] R. W. Bohannon. Comfortable and maximum walking speed of adults aged 20-79 years: reference values and determinants. *Age and Ageing*, 26:15–19, 1997.

[4] B. V. Cherkassky, A. V. Goldberg, and T. Radzik. Shorthest paths algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73:129–174, 1996.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 2001.

[6] R. Dial, G. Glover, D. Karney, and D. Klingma. A computational analysis of alternative algorithms and labelling techniques for finding shortest path trees. *Networks*, 9:215–348, 1979.

[7] R. W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5:345, 1962.

[8] P. Y. Gilliéron, D. Büchel, I. Spassov, and B. Merminod. Indoor navigation performance analysis. *Proceedings of the European Navigation Conference GNSS, TOPO-EPFL*, pages 1–9, 2004.

[9] M. Goetz and A. Zipf. Formal definition of a user-adaptive and length-optimal routing graph for complex indoor environments. *Geo-spatial Information Science*, pages 119–128, 2011.

[10] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24:1–13, 1977.

[11] P. Jorge. Localização indoor com optimização multicritério. Master's thesis, Universidade de Coimbra, junho 2014.

[12] L. Liu and S. Zlatanova. A "door-to-door" path-finding approach for indoor navigation. *Proceedings of GeoInformation For Disaster Management Conference, Antalya, Turkey*, pages 3–8, 2011.

[13] L. Liu and S. Zlatanova. Towards a 3d network model for indoor navigation. *Urban and Regional Data Management, editors: S. Zlatanova, H. Ledoux, E. Fendel e M. Rumor. Taylor & Francis Group London*, pages 79–94, 2012.

[14] P. de Matos, A. P. Afonso, and M. B. Carmo. Point of interest awareness using indoor positioning with a mobile phone. *Proceedings of PECCS 2011, editors: C. Benavente-Peces e J. Filipe, SciTePress*, pages 5–13, 2011.

[15] A. Pereira. Multilateração para localização indoor (em preparação). Master's thesis, Universidade de Coimbra, junho 2014.

[16] D. Schultes. Fast and exact shortest path queries using highway hierarchies. Master's thesis, Universidade de Saarlandes, 2005.

[17] K. S. Sunnerhagen, M. Hedberg, B. G. Henning, A. Cider, and U. Svantesson. Muscle performance in an urban population sample of 40- to 79-year-old men and women. *Scandinavian Journal of Rehabilitation Medicine*, 32:159–167, 2000.

[18] S. Warshall. A theorem on boolean matrices. *Journal of the ACM*, 9:11–12, 1962.

[19] C. Willen, K. Lehmann, and K. S. Sunnerhagen. Walking speed indoors and outdoors in healthy persons and in persons with late effects of polio. *Journal of Neurology Research*, 3:62–67, 2013.

[20] W. Yuan and M. Schneider. inav: An indoor navigation model supporting length-dependent optimal routing. *Geospatial Thinking, editors: M. Painho, M. Y. Santos and H. Pundt, Lectures Notes in Geoinformation and Cartography, Springer Berlin/Heidelberg*, pages 299–313, 2010.