Pedro Sousa Colaço

Algoritmo e Plataforma para a Otimização do Planeamento de Serviços de Apoio Domiciliário

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador: Jácome Cunha, Assistant Professor, University of Minho

Co-orientadora: Maria Isabel Gomes, Associate Professor, NOVA University of Lisbon

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

March, 2018

Algoritmo e Plataforma para a Otimização do Planeamento de Serviços de Apoio Domiciliário

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2]. [1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Resumo

Nesta tese vamos abordar o problema de planeamento do serviço de apoio domiciliário, que pode ser modelado como uma extensão de um problema de rotas de veículos. Concretamente, este problema passa por planear a melhor sequência de visitas a serem efetuadas por auxiliares de apoio domiciliário a um conjunto de pacientes, de maneira a que sejam satisfeitas as necessidades de cada um destes.

Este tipo de problemas é NP-difícil, o que obriga o desenho de algoritmos heurísticos e não de uma resolução exacta.

O trabalho desenvolvido passou por duas fases. A primeira consistiu no desenho de um algoritmo do problema em questão, da maneira mais eficiente possível. Na segunda, integrámos a solução numa aplicação web "moderna" e *"user-friendly"*. Pretendíamos que a nossa solução fosse o mais generalizada possível, de modo a que esta pudesse ser usufruída por outras instituições, para além da instituição que serviu como nosso caso de estudo.

Fizemos estudos com diversos parâmetros da solução e, tendo em conta os bons resultados obtidos, consideramos que a nossa solução oferece uma versatilidade inovadora, visto que permite aos potenciais interessados a possibilidade de considerar diferentes tipos de tarefas a realizar, e de escolher entre diferentes tipos de soluções encontradas.

Palavras-chave: Cuidados de Saúde em Casa; Roteamento; Agendamento; Otimização; Aplicação Web

Abstract

In this thesis we will address the problem of the home health care routing problem, which can be modeled as an extension of the vehicle routing problem. Specifically, this problem aims at defining the best sequence of visits to be carried out by the home support members to a set of patients, so that the needs of each of them are met.

This type of problem is NP-hard, which forces the design of heuristic algorithms instead of an exact resolution.

The work developed went through two phases. The first consisted in designing an algorithm of the problem, in the most efficient way possible. In the second, we integrated the solution into a modern and user-friendly web application. We wanted our solution to be as generalized as possible, so that it could be used by other institutions, in addition to the institution that served as our case study.

We have made studies with several parameters of the solution and taking into account the good results obtained, we consider that our solution offers an innovative versatility, since it offers potential interested organizations the possibility to consider different types of tasks, and to choose between different types of solutions found.

Keywords: Home Health Care; Routing; Scheduling; Optimization; Web Application

Índice

Lista de Figuras

Lista de Tabelas

Listagens

Capítulo 1

Introdução

O envelhecimento da população em Portugal (e no mundo denominado de "desenvolvido") é uma realidade, que não pode ser desmentida. O Instituto Nacional de Estatística (INE) aponta que, em 2031, a população terá baixado dos simbólicos 10 milhões de habitantes, que sempre associámos como referência de número populacional, em Portugal. O INE prevê ainda que o índice de envelhecimento mais do que duplicará até 2080, passando de 147 para 317 idosos por cada 100 jovens entre os 15 e os 29 anos (*Projeções de População Residente em Portugal* 2017). Estas projeções representam, no fundo, vários desafios de saúde pública, aos quais, não podemos ficar alheios.

Com o avançar da idade, as pessoas vão perdendo capacidades motoras, que outrora tiveram. Esta situação, somada ao isolamento social das pessoas idosas, leva à necessidade de existência de instituições capazes de servir estas populações. Estas instituições têm o propósito de garantir, da melhor forma possível, os cuidados básicos necessitados pelos idosos, traduzindo-se numa melhor qualidade de vida destes.

As tarefas de serviços domésticos prestadas pelos auxiliares destas instituições podem ir desde garantir a administração da medicação referente a cada utente, como a fazer um acompanhamento da alimentação, ou assegurar os cuidados básicos de higiene e conforto pessoal.

1.1 Problema

O problema do serviço de apoio domiciliário relaciona dois tipos de intervenientes: auxiliares, que são membros da instituição e que realizam as tarefas que lhes são atribuídas; utentes, que recebem um qualquer tipo de serviço, relacionado com cuidados sociais ou de saúde. Assim, o problema passa por atribuir um conjunto de utentes a cada auxiliar, e planear a sequência de visitas (rota) a ser efetuada por cada um destes, de maneira a serem cumpridas as necessidades de cada utente. Muitas das instituições que oferecem estes serviços, geralmente, planeiam os seus serviços "à mão". Isso resulta num grande esforço da pessoa responsável pelo planeamento, e numa solução, que muito provavelmente, não será ótima.

Planear uma solução para um problema destes é um desafio complexo de otimização, que passa por diferentes níveis de decisão, tais como a distribuição de auxiliares por turnos de trabalho, a afetação de auxiliares a utentes, ou a definição de rotas a serem efetuadas na área geográfica considerada. Por vezes, nem todos os auxiliares apresentam as mesmas qualificações ou competências. Isto leva a que seja necessário restringir a afetação de auxiliares, de maneira a que o serviço a ser efetuado, não o seja feito por alguém não capaz de o realizar. Também se pode ajustar as afetações às preferências dos utentes, ou considerar a existência de uma continuidade de cuidado, que no fundo, passa por garantir que cada utente seja visitado pelo mesmo auxiliar o maior número de vezes possível, de maneira a puder proporcionar o maior conforto e satisfação possível aos utentes. Para além dos aspetos mencionados, nos problemas do serviço de apoio domiciliário também é preciso ter em consideração o tempo, o que aumenta a complexidade do problema. Por exemplo, um utente diabético que necessite que lhe sejam administradas injeções de insulina, precisa necessariamente de ser visitado numa determinada janela de tempo.

O problema de serviço de apoio domiciliário é, no fundo, equivalente ao problema do caixeiro viajante múltiplo com janelas temporais (PCVMJT), com algumas restrições específicas. O PCVMJT trata de determinar um conjunto de rotas ótimas a serem percorridas por veículos, de maneira a servir um conjunto de clientes com localização pré-definida, cada um dentro de uma janela de tempo especificada.

1.2 Solução

A grande motivação deste trabalho passava por desenvolver uma solução inovadora e um software que, em primeira instância, ajude a instituição de solidariedade social da

Grande Lisboa, que representa um caso de estudo da dissertação a realizar, a fazer uma transição para um mecanismo tecnológico. Os desafios levantados pelo problema descrito podem ser abordados de diferentes maneiras. O primeiro grande desafio passava, então, por resolver o problema da maneira mais eficiente possível, em termos computacionais. É necessário ter um especial cuidado, pois estamos perante um problema de uma complexidade potencialmente elevada, e que contém um grande número de restrições, o que pode levar, à partida, a uma resolução em tempo não-útil. Isto acontece porque o problema de planeamento do serviço de apoio domiciliário é um problema NP-difícil, o que significa que não se sabe se o problema pode ser resolvido em tempo polinomial. Foi necessário testar diferentes abordagens, e definir a melhor estratégia de resolução.

O segundo grande objetivo passava por materializar a resolução encontrada num software, de maneira a que as dificuldades existentes neste momento em instituições de solidariedade social semelhantes ao caso de estudo discutido, possam ser suprimidas. O trabalho desenvolvido passou por integrar um algoritmo de resolução do problema, numa aplicação web, de modo a que esta possa ser usada por funcionários de instituições, com o objetivo de auxiliar no planeamento das atividades e otimizar o seu trabalho.

Apesar do foco do desenvolvimento da dissertação ser o já referido caso de estudo, a ideia passava por criar uma solução o mais generalizada possível, de modo a que o trabalho desenvolvido possa ser usado por outras entidades. Pretendíamos também que a solução possa resolver casos de média/grande dimensão.

O problema do serviço de apoio domiciliário pode compreender vários objetivos e restrições. A nossa solução compreende um planeamento de rotas de veículos, podendo abranger vários dias, sendo que os objetivos a considerar passam por minimizar o tempo total de rotas, minimizar o tempo total de espera entre tarefas e maximizar a justiça de planeamento, de maneira a haver um equilíbrio de trabalho entre todos os funcionários. Em relação às restrições, para além da continuidade de cuidado já referida, consideramos também janelas temporais, definição de um limite máximo de horas que cada funcionário pode trabalhar diariamente, e pausas para almoço.

1.3 Organização do Documento

No capítulo 2 apresenta-se a revisão bibliográfica, onde é revisto o trabalho publicado que se relaciona com o problema a resolver; no capítulo 3 é descrito todo o algoritmo que soluciona o problema; no capítulo 4 são expostos os resultados e a real eficiência do algoritmo. No 5º é descrita toda a componente da aplicação, enquanto que no capítulo 6 está exposta a conclusão do trabalho desenvolvido.

Antes de se desenvolver a meta-heurística descrita no capítulo 3, foram investigadas algumas abordagens de resolução do Problema do Caixeiro Viajante (PCV), que porventura, poderiam servir para a resolução do nosso problema. Contudo, tendo em conta os resultados obtidos, acabámos por descartar estas opções. O trabalho desenvolvido encontra-se em Anexo.

Capítulo 2

Trabalho Relacionado

O problema de apoio domiciliário é um problema que já foi estudado por diversos autores, o que é muito importante, pois é um problema relativo a um contexto do mundo real. Em Fikar e Hirsch 2017 é feita uma revisão de um conjunto de artigos sobre o problema do serviço de apoio domiciliário. Aqui pudemos perceber as diferenças entre as abordagens efetuadas, seja ao nível dos objetivos, das restrições, ou métodos de resolução.

Os artigos científicos sobre esta temática podem-se dividir pelo período de tempo em que é considerado o problema. Tanto pode ser considerado, por exemplo, um único dia, como vários meses.

2.1 Um dia de planeamento

Em relação aos artigos que consideram apenas um dia de planeamento, pudemos constatar que a otimização da rota a planear é a principal preocupação. Como a rota está relacionada com o trabalho a ser efetuado pelos auxiliares, também podem ser considerados, como objetivos, tempos de espera ou horas extra. A preferência dos utentes e dos auxiliares também tem uma relativa importância. Por outro lado, a minimização do número de auxiliares a trabalhar, ou a maximização do número de tarefas a serem realizadas são apenas modelados como objetivos em dois artigos, relativamente ao conjunto de artigos revistos em Fikar e Hirsch 2017. Isto acontece porque na generalidade dos

casos estudados (Mankowska et al. 2014 ou Braekers et al. 2016) o número de auxiliares já está definido à priori; e porque todas as tarefas devem ser realizadas.

As restrições mais frequentemente usadas nos problemas em questão são as janelas temporais, que obrigam a que os utentes sejam visitados num determinado horário; as habilitações/qualificações dos auxiliares; e o tempo de trabalho, que pode ser modelado como um máximo de horas de trabalho que não deve ser excedido (Bachouch et al. 2011).

Por outro lado, não é atribuída muita importância em definir uma pausa para almoço. Ainda assim, é modelada nalguns dos artigos. Sincronização e precedência temporal assumem um pouco mais de importância: por exemplo, cada cliente tanto pode ser visitado por dois auxiliares ao mesmo tempo, como receber mais do que uma visita por dia, efetuada por diferentes auxiliares (Decerle et al. 2018). Aqui, é necessário ter um especial cuidado com as tarefas atribuídas, de maneira a que a satisfação e continuidade de cuidado do utente não fiquem comprometidos.

São poucos os artigos que consideram a incerteza existente neste tipo de problemas. A incerteza está ligada ao facto de num contexto de mundo real, nem tudo o que foi planeado irá necessariamente acontecer. Assim, ao invés de se presumir todos os dados como determinísticos, algumas abordagens podem ser consideradas como estocásticas (Lin et al. 2017).

Na maioria dos trabalhos efetuados, em relação aos artigos que consideram um dia de planeamento, foram desenvolvidas meta-heurísticas. Os métodos usados são variados, incluindo algoritmos de busca em vizinhança variável (Erdem e Bulkan 2017); algoritmos genéticos (Decerle et al. 2018) ou GRASP (Shao et al. 2013). O tempo de resolução está diretamente relacionado com o tamanho do problema considerado, os seus objetivos e restrições, linguagem de programação usada, bem como poder computacional.

Foi ainda observado que nalguns artigos, são combinadas as vantagens de métodos exatos com meta-heurísticas (Braekers et al. 2016).

Tendo em conta que o trabalho desenvolvido nesta dissertação era vocacionado para um possível planeamento de múltiplos dias, acabámos por explorar mais esse tipo de artigos, comparativamente com os artigos que compreendem um planeamento de um dia.

2.2 Múltiplos dias de planeamento

Enquanto que nos problemas referentes a um dia, a minimização de tempo e custos relacionados com a rota a solucionar são as grandes preocupações, nos problemas com um período de planeamento superior a um dia, há outros aspetos que ganham especial relevância. Um deles é a continuação dos cuidados. Ao serem considerados múltiplos dias, é minimizado o número de diferentes auxiliares que visitam um mesmo paciente. Há uma maior preocupação com o trabalho laboral a ser executado pelos auxiliares. Por exemplo, em relação à existência de um equilíbrio de horas de trabalho entre os auxiliares, ou não ser excedido um máximo de horas a trabalhar.

Os métodos de resolução dos artigos considerados para múltiplos dias focam-se tanto em meta-heurísticas como em métodos exatos.

O tempo de planeamento dos artigos revistos varia desde 5 dias até 1 ano, o que, obviamente, influencia em muito o tempo de processamento de uma solução ótima.

Como se pôde verificar, o trabalho realizado nesta área é muito heterogéneo e não existe nenhum método de solucionamento que seja aceite de maneira comum.

De seguida iremos analisar com maior detalhe alguns dos trabalhos, com um planeamento de vários dias, cujas abordagens adotadas foram inspiradoras no trabalho desenvolvido na dissertação.

Bachouch et al. 2011 formulam o problema como um problema de programação linear inteira, sendo a função objetivo a minimização da distância total percorrida. Verificou-se que a dimensão das instâncias consideradas é relativamente pequena. O caso mais complexo permite um planeamento de 5 dias, com 7 auxiliares e 20 utentes. Neste caso específico não foi encontrada a solução ótima ao fim de duas horas, tendo sido encontrada uma admissível. É importante referir que o problema foi resolvido com o auxílio de software comercial: Lingo e CPLEX.

Em Shao et al. 2013 procurou-se demonstrar que o uso de métodos exatos não é a abordagem de resolução mais adequada a fazer. Os autores propõem que os algoritmos devam ser desenhados, tendo em conta as especificações do problema em si. O problema foi inicialmente formulado como um problema de programação linear inteira. Com o auxílio do CPLEX (foi verificado noutros artigos que este é o software mais usado para resolver problemas deste tipo) percebeu-se que existem limitações ao nível das variáveis e restrições máximas a usar, o que limita a resolução de problemas para situações de larga escala.

Assim, os autores desenvolveram uma meta-heurística GRASP (Greedy Randomized Adaptive Search Procedure), de maneira a puder resolver instâncias de casos reais, em

que o tamanho do problema fosse grande. GRASP é um tipo/classe de algoritmo que, a partir de uma solução inicial encontrada, procura melhorar a qualidade dessa solução por procura local.

É importante referir que num artigo anterior estes mesmos autores já tinham desenvolvido uma meta-heurística paralela (Shao et al. 2012). De maneira geral, para este tipo de problemas, uma heurística paralela cria múltiplas rotas paralelamente, enquanto que uma sequencial gera uma de cada vez e procura explorar ao máximo os recursos dessa rota. Nesse caso específico, é usada uma estratégia de duas fases onde na primeira fase é gerada uma rota admissível. De seguida a rota gerada é, se possível, otimizada, obedecendo a um conjunto de critérios. Este artigo terá sido a primeira grande inspiração para no nosso trabalho, visto que foi exactamente esta ideia que tentámos implementar: a criação paralela de rotas para todos os auxiliares numa primeira fase; e numa segunda, procurámos optimizar as rotas geradas inicialmente.

Em Shao et al. 2013 procurou-se comparar os resultados de um GRASP sequencial, com um GRASP paralelo e com um método exato. Procurava-se então perceber se no algoritmo sequencial há uma melhor minimização de custos em relação ao algoritmo paralelo. De facto, isso veio-se a verificar: à medida que o tamanho do problema aumenta, em termos de número de auxiliares e utentes a considerar, há uma melhoria em termos de custo e tempo das soluções obtidas. O algoritmo foi capaz de gerar soluções para todos os casos testados, tendo havido uma melhoria média de 18% em relação ao algoritmo paralelo. Os resultados mostraram que o software comercial usado não consegue resolver problemas de média/larga dimensão. Ao fim de 15 horas não foi encontrada a solução ótima para um problema com cerca de 16 auxiliares, 120 pacientes, e apenas um dia de escalonamento.

2.2.1 Meta-heurísticas

Em Erdem e Bulkan 2017 é desenvolvida uma solução meta-heurística dividida em duas fases "cluster first and schedule route second". A primeira fase passa por dividir, em grupos, os pacientes a visitar, tendo em conta a proximidade geográfica. Esta divisão leva a uma diminuição do tamanho do problema. Na segunda fase, uma primeira solução inicial é gerada através de três possíveis estratégias: determinística, onde se se começa no funcionário de índice mais pequeno e se tenta associá-lo à tarefa de índex mais pequeno, tendo em conta um conjunto de restrições, tais como janelas temporais ou níveis de habilitações; aleatória, onde se associam funcionários a tarefas, de maneira aleatória; híbrida das duas anteriores. Depois, a solução é melhorada com recurso a um

algoritmo de busca em vizinhança variável, onde são aplicadas diferentes estratégias.

Em Lin et al. 2017 é proposto um "Modified Harmony Search" (MHS), que é um algoritmo que tem como objetivo aprimorar os resultados de um "Harmony Search Algorithm" (HSA) através de um esquema de saturação. Metaforicamente, num HSA, o objetivo passa por encontrar uma harmonia perfeita, simulando o comportamento dos diferentes músicos.

Um dos objetivos do artigo passava por provar que, o MHS obtém melhores resultados, relativamente ao HSA. O outro objectivo, diz respeito à incerteza da modelação do problema, em contexto real. Foram testados diferentes incidentes não planeados que levam à necessidade um novo roteamento do problema, sem pôr em causa a admissibilidade da solução. Os possíveis cenários incluem: um funcionário pede para sair; um paciente pede para sair; um paciente altera a janela temporal a ser atendido.

Em Decerle et al. 2018 é desenvolvido um algoritmo memético, no qual o foco da modelação do problema passa por garantir sincronização de visitas, através de restrições "suaves" de janelas temporais. Basicamente, nos casos em que são necessários dois funcionários para realizar uma tarefa, estes têm que se encontrar, simultaneamente, no mesmo local. Ou seja, não é necessário que os dois funcionários estejam juntos durante o dia todo, mas apenas nos momentos exatos, em que as tarefas tiverem de ser efetuadas. De maneira a minimizar uma potencial diferença de chegadas entre os dois funcionários ao local do utente a servir, a função objetivo sofre penalizações sempre que estas diferenças ocorrem. Neste trabalho, a população inicial corresponde a um conjunto de rotas geradas, sendo que os indivíduos com os valores da função objetivo mais baixa, têm mais probabilidade de serem escolhidos. Depois, os indivíduos escolhidos passam pelo processo de "crossover", de maneira a criar novas soluções, com base na informação genética dos indivíduos. Segue-se o processo de mutação, de modo a preservar diversidade genética da população. Por fim, é aplicada procura local, com o intuito de otimizar a solução, até ser atingido algum dos critérios de paragem (solução ótima encontrada, tempo limite de procura atingido). É esta procura local que distingue um algoritmo genético de um memético. Este foi outro dos trabalhos que nos influenciou, visto que o nosso algoritmo também está preparado para sincronizar visitas de elementos da mesma equipa a um determinado auxiliar.

2.3 Interação com o Utilizador

Nesta secção destacamos o trabalho encontrado que contempla a interacção dos utilizadores com este tipo de algoritmos. Primeiro, começámos com um artigo mais antigo

(Begur et al. 1997) onde é desenvolvida uma plataforma que serve de ajuda à tomada de decisões, no planeamento de rotas. A principal preocupação ao analisar este artigo não foi a de perceber concretamente que métodos de solucionamento são utilizados ou que restrições são consideradas, mas sim perceber as vantagens de utilização de um Sistema de Apoio à Decisão Espacial (SADE).

Os autores desenvolveram um SADE, que pode ser definido como um sistema interativo, que serve de auxílio à tomada de decisões ao resolver um problema espacial. Está integrado num software GIS (Sistema de Informação Geográfica) com heurísticas de planeamento de rotas e bases de dados, que contêm informação geográfica, de maneira a criar uma plataforma user friendly - de fácil utilização

O principal objetivo era fornecer uma lista de utentes a serem visitados por cada auxiliar, ordenadamente, de maneira a otimizar a produtividade destes mesmos auxiliares. Essencialmente, esta plataforma permite aos utilizadores interpretarem visualmente uma resolução de um problema de serviço de apoio domiciliário, tendo por base a digitalização de uma cidade ou de um mapa com as localizações dos utentes a serem servidos. A funcionalidade da plataforma que foi considerada mais relevante na análise feita, é a existência de uma função re-route disponibilizada na interface.

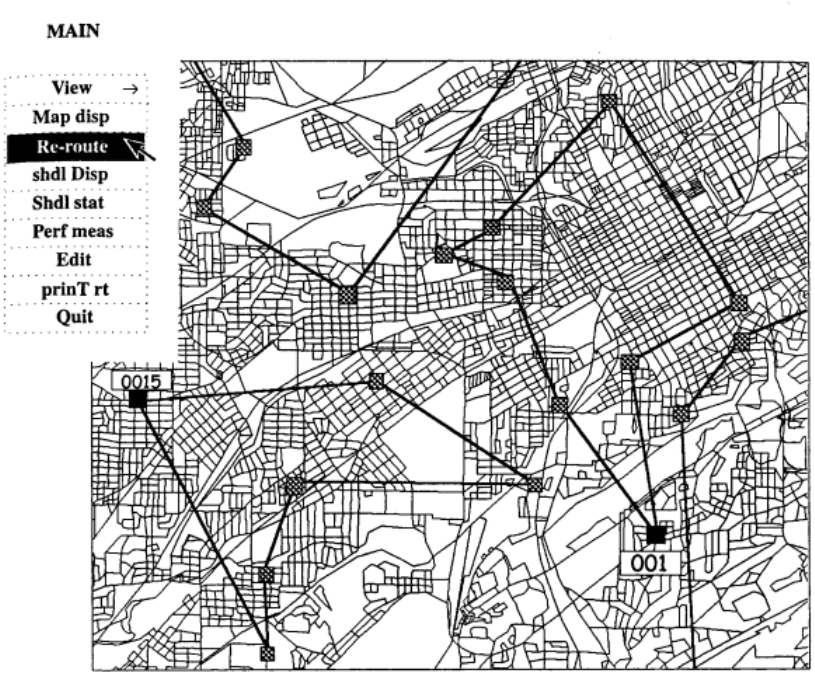

Figura 2.1: Interface da plataforma desenvolvida (fonte: Begur et al. 1997)

Na Figura 2.1 podemos ver as rotas atribuídas a dois auxiliares. Os quadrados preenchidos representam a localização dos auxiliares; os restantes representam a localização dos pacientes a serem visitados. Um utilizador pode usar a interface disponibilizada para analisar os resultados obtidos, ou alterar, ele mesmo, as rotas propostas pelo algoritmo. É possível alterar a ordem de visita dos pacientes, de maneira a lidar com algum potencial problema relacionado com o caminho definido pelo algoritmo; é ainda possível, alterar alguma afetação auxiliar-utente, de forma a, por exemplo, potenciar as capacidades e habilitações de cada auxiliar.

A plataforma disponibiliza ainda tabelas com a informação relativa a cada auxiliar: distância total a ser percorrida e estimativa de tempo de rota; diferença destes valores com os valores médios atribuídos aos restantes auxiliares, de modo a puder ser feita uma comparação, a nível de equilíbrio de trabalho. Este tipo de abordagem provou-se ser bastante útil e prático para problemas de tamanho pequeno. Incorporámos na nossa solução características semelhantes.

Da pesquisa efetuada, pudemos ainda perceber que existem plataformas web vocacionadas para a resolução do problema de roteamento de veículos com janelas temporais (PRVJT). O PRVJT é uma extensão do PCVMJT, na medida em que impõe adicionalmente restrições aos veículos, o que condiciona as rotas a serem efetuadas. A plataforma, neste momento, mais popular é apresentada em Erdoğan 2016. Unifica Excel, GIS (Geographic Information System) público e meta-heurísticas, conseguindo resolver problemas com cerca de 200 clientes. Esta plataforma não cobre, no entanto, todas as especificidades do planeamento do serviço de apoio domiciliário, como a preocupação com a continuidade de cuidados, já explicada anteriormente, ou o facto de nem todos os auxiliares terem as mesmas habilitações. Outra limitação da plataforma é que esta está apenas preparada para resolver soluções de um só dia, o que é incompatível com a necessidade de continuidade de cuidados.

Não há conhecimento de plataformas recentes diretamente vocacionadas para a resolução do problema do planeamento do serviço de apoio domiciliário.

Capítulo 3

Algoritmo para o Planeamento das Rotas

Neste capítulo vamos abordar e explicar todo o algoritmo desenvolvido para o planeamento de rotas.

O planeamento de rotas a efetuar compreende um período variável de dias a ser definido pelo utilizador. A meta-heurística está, portanto, preparada para resolver problemas com múltiplos dias de planeamento.

De acordo com Fikar e Hirsch, (2017) os objetivos associados ao problema do planeamento de serviços de apoio domiciliário são: minimização do tempo total de rotas, minimização de custos das rotas, minimização das distâncias totais percorridas nas rotas, minimização de tempo de espera, minimização das horas extra a serem efetuadas pelos auxiliares, maximização de preferências dos utentes, minimização dos auxiliares a trabalhar, maximização da continuidade de cuidados, maximização da justiça e maximização das tarefas a realizar. Destes, os que foram considerados no algoritmo desenvolvido são: minimização do tempo total de rotas, minimização de tempo de espera, e maximização da justiça de planeamento, de maneira a haver um equilíbrio de trabalho entre todos os funcionários (Figura 3.1).

Em relação às restrições (Figura 3.2), as janelas temporais e a definição de um limite máximo de horas que cada funcionário pode trabalhar diariamente são indispensáveis. Também considerámos pausas para almoço, e a possibilidade de existir continuidade de cuidado.

Não considerámos diferenças qualificativas entre auxiliares, o que pressupõe que

Objetivos Maximização (+) / Minimização(-)	*TR* (-)	*CR* (-)	*DR* (-)	*TE* (-)	*HE* (-)	*PR* (+)	*#A* (-)	*CC* (+)	*JU* (+)	*#T* (+)
A Nossa Proposta	✓			✓					✓	

TR – Tempo de Rota; CR – Custo de Rota; DR - Distância de Rota; TE - Tempo de Espera; HE – Horas Extra; PR – Preferências; #A – Auxiliares; CC – Continuidade de Cuidado; JU – Justiça; #T - Tarefas

Figura 3.1: Objectivos considerados na meta-heurística

cada funcionário estará apto a realizar qualquer tarefa. Não foi considerado, também, a ocorrência de acontecimentos incertos, não previstos, que poderiam levar a alterações no planeamento de rotas, tais como, a cessão repentina de funções de um funcionário, ou a alteração da janela temporal de um determinado paciente. A nível de continuidade de cuidado, o utilizador terá a possibilidade de escolher se um paciente terá de ser visitado sempre pela mesma equipa num dia, ou se não haverá restringimento a esse nível. Não existem diferentes tipos de utentes ou equipas, sendo que cada utente poderá ter tanto tarefas que requeiram duas pessoas, como tarefas que requeiram apenas uma pessoa.

A meta-heurística será abordada com mais detalhe ao longo do capítulo, mas a ideia principal passa por planear um dia de cada vez. Para cada dia, vamos tentar encontrar uma solução inicial, onde as rotas vão sendo construídas paralelamente, tendo em conta as distâncias entre nós, janelas temporais e duração das tarefas, com vista a minimização total do tempo gasto. No caso de todas as tarefas terem sido atribuídas, o algoritmo transitará para a segunda fase, se o utilizador assim o entender, onde será feita uma melhoria da solução. Aqui, é feita uma troca de tarefas (nós) até um determinado tempo limite. No final, são retornadas três soluções diferentes que até poderão ser a mesma no caso de se verificar que os três parâmetros de compararão são todos otimizados pela mesma solução. As três solução a retornar são, então, a solução onde é minimizado o tempo total das rotas, a solução onde é minimizado o tempo total de espera, e a solução onde é maximizada a justiça.

Restrições	*JT*	*HA*	*TT*	*PA*	*CC*	*IN*
A Nossa Proposta	✓		✓	✓	✓	

JT – Janelas Temporais; HA – Habilidades/Qualificações dos Auxiliares; TT – Tempo de Trabalho; PA – Pausas (almoço); CC – Continuidade de Cuidado; IN - Incerteza

Figura 3.2: Restrições consideradas na meta-heurística

3.1 Objectos

Antes de se passar à modelação e explicação do algoritmo vamos introduzir os conceitos necessários para a compreensão do mesmo. Na Figura 3.3 apresenta-se o diagrama de classes relativamente aos objectos considerados no programa.

Um nó (Node) representa uma tarefa a realizar ou um "centro" de onde partem todos os funcionários. Cada tarefa é representada por uma localização geográfica; uma janela temporal; duração prevista; variável booleana relativamente à permissão de puderem ver a sua janela temporal estendida; tipo de tarefa:

- 1 - Tarefa onde um funcionário é suficiente;
- 2 - Tarefa onde será necessária a acção de dois funcionários.

Para representar transportes de utentes existe uma outra variável:

- 0 - A tarefa não é relativa a transportes;
- 1 - Transporte não-prioritário desde casa até ao centro;
- 2 - Transporte prioritário de um utente desde sua casa até ao centro;
- 3 - Transporte não prioritário desde o centro até sua casa;
- 4 - Transporte prioritário desde o centro até sua casa.

Nesta terminologia, prioritário significa apenas que o transporte irá ser efectuado sem existência de possibilidade de outros pacientes seguirem nessa mesma viagem. Nos casos de não prioridade, o paciente poderá ser transportado juntamente com outros pacientes.

Um arco (Arc) representa um caminho entre um par de nós, percorrido por um funcionário, numa carrinha. O algoritmo de selecção de tarefas baseia-se, sobretudo, na selecção do melhor arco possível, ou seja, aquele que, cumprindo as circunstâncias requeridas no momento de selecção, irá minimizar o tempo da rota. Isto é, tendo em conta que todas as tarefas têm de ser realizadas, o arco a escolher será um arco que parta do nó actual e chegue a um outro ainda não existente na solução. Este último nó não será necessariamente o que está mais perto, pois esse poderia, por exemplo, ter uma janela temporal distante do tempo actual. No fundo, esse valor que determina a escolha do arco é representado através da variável *bestPossibleTime*. Representa, então, o momento temporário no qual pode ser realizada a tarefa do segundo nó, partindo do primeiro nó.

Este valor vai estar dependente da distância entre ambos os nós, do tipo de transporte, da janela temporal da tarefa a realizar, e do tempo actual. Tendo isto em conta, sempre que um novo arco é adicionado à solução, é necessário actualizar os *bestPossibleTime* de todos os possíveis arcos que possam vir a ser escolhidos.

Uma carrinha (Van) é representada por uma capacidade inicial de lugares e uma variável booleana, que determina se está ou não disponível num determinado momento.

Um paciente (Patient) tem uma posição geográfica atribuída e poderá ter um conjunto de tarefas. Aqui não há restrições de tarefas. Cada paciente poderá ter tarefas que requeiram duas pessoas, apenas uma pessoa, ou ainda tarefas de transportes.

Um auxiliar (Worker) irá ter um horário de trabalho baseado na quantidade de funcionários disponíveis e das tarefas a realizar, sendo que este horário não poderá exceder um determinado número de horas, a definir pelo utilizador. Terá direito a uma pausa de almoço, se o utilizador entender que devem ser considerados almoços para os auxilaires. Os auxiliares partem todos de um centro comum, e vão então, realizar tarefas, até as mesmas estarem todas concluídas. Cada auxiliar tem a possibilidade de usufruir de diferentes carrinhas. Sempre que um auxiliar necessitar de uma carrinha, ir-lhe-á ser atribuída aquela com maior capacidade, dentro das disponíveis.

Uma equipa (Team) é um conjunto de auxiliares. No nosso caso específico, as equipas terão no máximo dois elementos. Uma equipa serve, sobretudo, para lidar com pacientes que necessitem de dois trabalhadores para certas tarefas. Nestes casos, os auxiliares da mesma equipa, que poderão estar a visitar diferentes pacientes num mesmo momento, irão deslocar-se ao encontro do paciente, após concluírem as tarefas respectivas correntes. Um dos trabalhadores poderá ter de esperar, no caso do segundo chegar ao local mais tarde; ou até os dois, no caso de ambos chegarem antes da janela temporal definida. Uma equipa serve também para poder planear que um mesmo paciente seja visitado pelos mesmos funcionários. Uma das variáveis de entrada do programa (mais à frente voltaremos a abordar estas particularidades, com mais detalhe) é precisamente a selecção, ou não, de uma restrição relativa ao facto de cada paciente ter de ser visitado, num mesmo dia, pela mesma equipa. No caso de ser seleccionada, poderá significar um maior conforto para o paciente, visto que não terá de lidar com funcionários diferentes.

3.2 Plano

Numa visão mais geral, cada plano permite gerar um conjunto de soluções, sendo que os parâmetros de entrada são os seguintes:

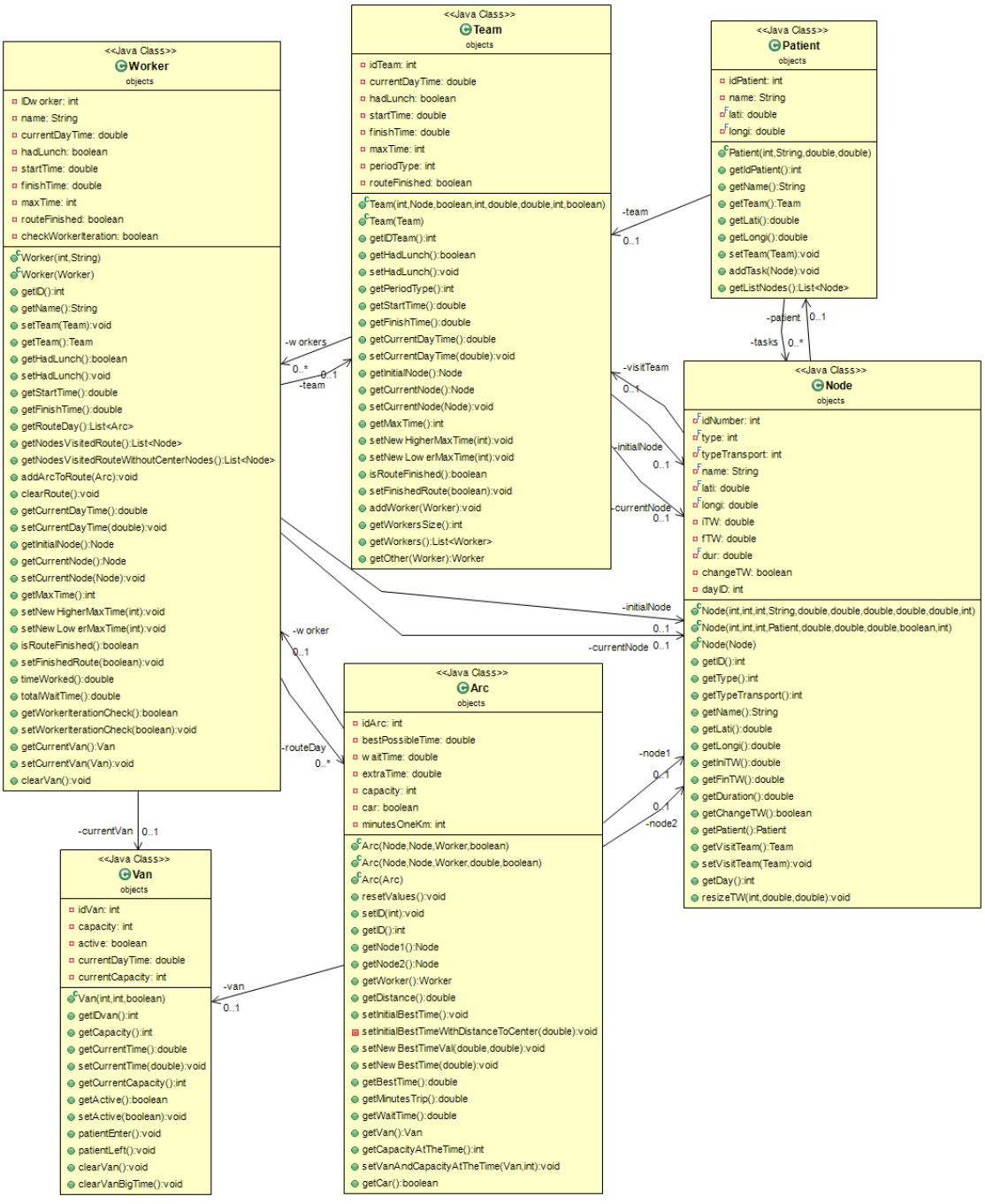

Figura 3.3: Diagrama de Classes - Objectos

- **string** namePlan - nome identificativo do plano;
- **int** days - número de dias a considerar;
- **boolean** lunch - booleano, relativo à existência, ou não, de almoços para os auxiliares;
- **boolean** car - booleano, relativo ao modo de transporte, entre nós. True, serão considerados 2 minutos por cada km percorrido; False, serão considerados 10 minutos por km. No caso de existir pelo menos uma tarefa de transporte, será sempre considerada a primeira opção, como modo de transporte;
- **int** numberVans - número de carrinhas a considerar. Por cada elemento, é inserido o número de lugares disponíveis;
- **int** maxTime - tempo, em horas, que cada trabalhador não pode exceder, num dia;
- **int** maxWaitTimeBetweenTasks - máximo tempo de espera permitido, em minutos, para cada auxiliar antes da realização de uma tarefa:
- **int** lunchTime – hora de almoço;
- **boolean** sameTeamCondition – booleano. No caso da opção ser seleccionada, os pacientes, num mesmo dia, têm de ser visitados pela mesma equipa, para todas as suas tarefas;
- **int** tasksTWresize – valor, em percentagem, de alargamento das janelas temporais. Cada tarefa tem uma variável booleana (changeTW) que permite, ou não, este alargamento;
- **boolean** extensiveSearch – booleano, relativo à procura exaustiva. False, não irá existir uma troca exaustiva de tarefas entre trabalhadores, relativamente à primeira solução, na procura de melhores soluções; True, a procura de melhores soluções irá acontecer;
- **int** timeout - tempo limite, em minutos, da execução do algoritmo.
- **double | double** - latitude e longitude do centro inicial, de onde partirão todos os auxiliares. Na aplicação web o utilizador tem a possibilidade de indicar um ponto no mapa, gerando as coordenadas automaticamente.

O algoritmo permitirá, então, obter uma lista de soluções diárias, que ficará associada ao plano.

3.3 Solução

Cada dia poderá ter quatro soluções diferentes: uma solução inicial; e três tipos de soluções diferentes, no caso de ser seleccionada a ocorrência de uma busca extensiva (ver Figura 3.4). Estas três soluções irão ter como base a primeira. O que o algoritmo faz é uma troca extensiva de tarefas entre trabalhadores, e sempre que um dos três parâmetros seguintes é melhorado, a solução é guardada:

- Minimização do tempo total de rotas;
- Minimização do tempo de espera entre tarefas - Por vezes um trabalhador chega a um determinado destino, mas tem de aguardar algum tempo, pois pode ter chegado antes da janela temporal definida para essa tarefa. Neste caso, a diferença entre o início da realização da tarefa e a chegada de trabalhador é o tempo de espera. Esta solução será, então, aquela onde os tempos totais de espera são minimizados;
- Justiça - Aqui o que se pretende é obter a solução mais justa, no que se refere à distribuição das horas de trabalho entre os auxiliares. Serão seleccionados o trabalhador com maior, e menor carga horária. Quanto menor, em percentagem, for esta diferença, mais justa será a solução.

  Justice = 100.0 - ((timeWorkedLeast*100.0) / timeWorkedMost)

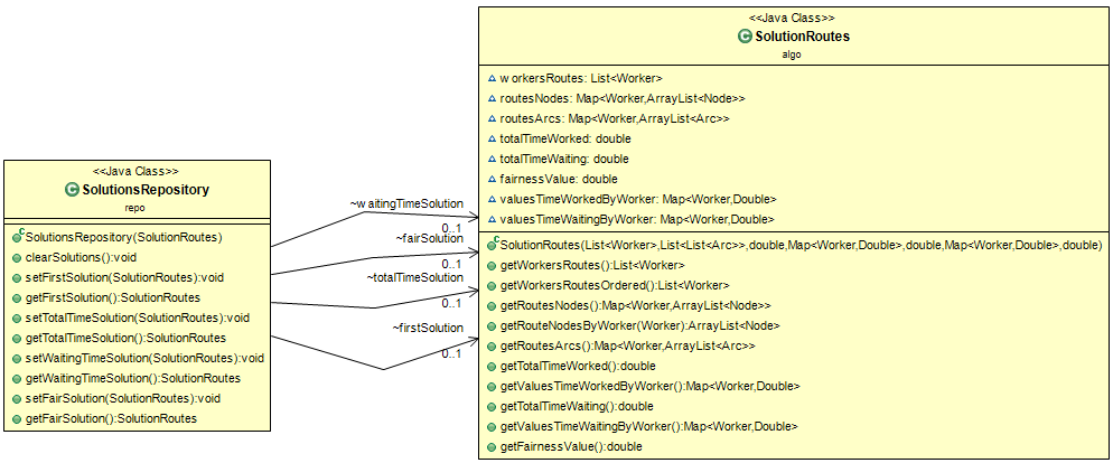

Figura 3.4: Diagrama de Classes - Soluções

3.4 Repositórios

Na Figura 3.5 apresenta-se o diagrama de classes relativo aos repositórios. É aqui que as listas de dados são guardadas. Inicialmente introduz-se toda a informação relativa às carrinhas, aos pacientes, aos auxiliares e às tarefas a realizar. Depois, para cada dia, é introduzida informação nos repositórios dos arcos com base nas tarefas a realizar. Depois de geradas as soluções desse mesmo dia, os repositórios são limpos e volta-se a repetir o mesmo processo para os restantes dias.

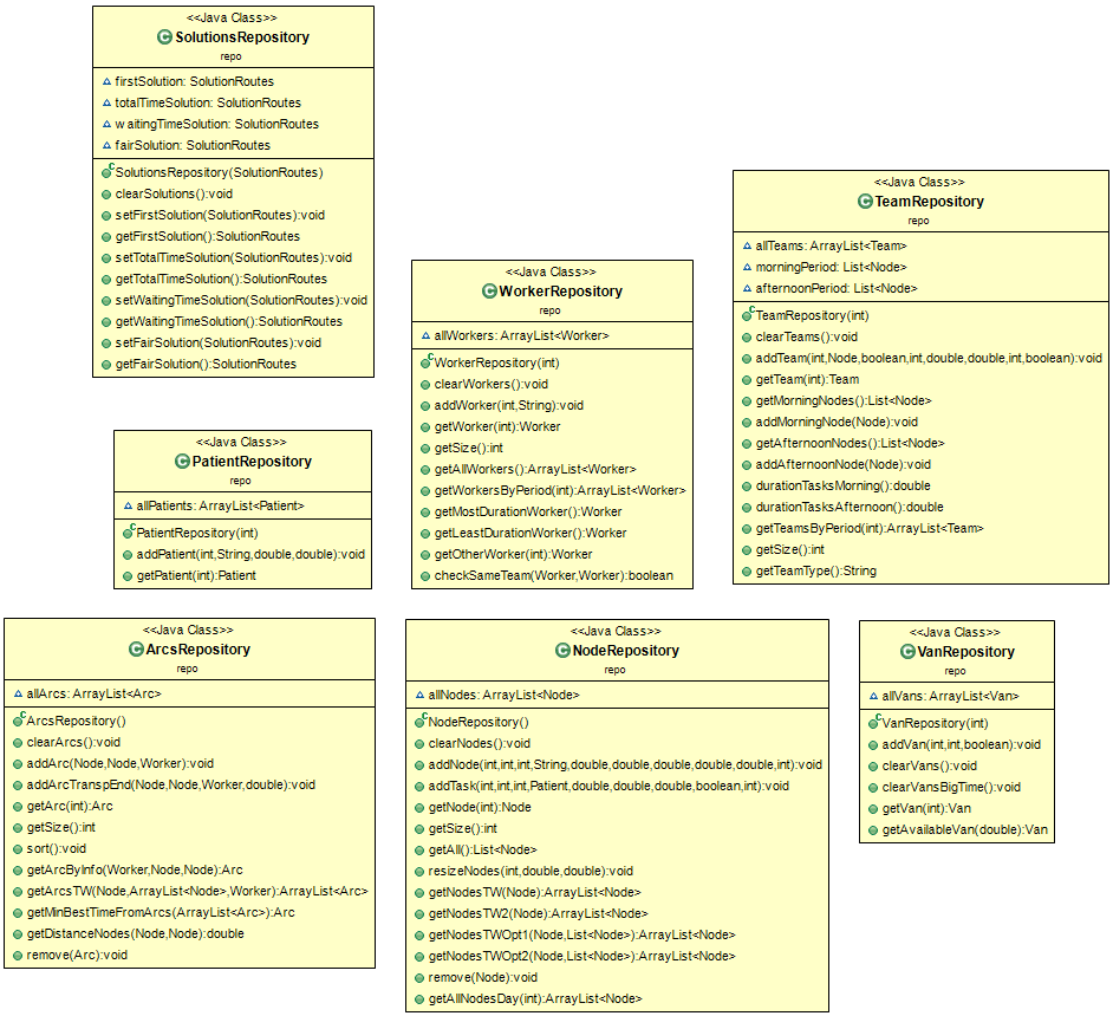

Figura 3.5: Diagrama de Classes - Repositórios

3.5 Algoritmo

Algorithm 1: Overall Algorithm

```
1    Input Plan p      Output solusPlan

     List< SolutionsRepository > solusPlan

2  try
3  for each day in p.days do
4  |   SolutionsRepository dayS = MakeDaySolution(day, p)
5  |   solusPlan.add(dayS)
   end
6  catchTimeout exception
7  [InterromperPrograma]
   end try
```

Algorithm 2: Make Day Solution

```
1    Input int day, Plan p      Output solutionsDay

     SolutionsRepository solutionsDay

2  PreSetup()
3  ResizeTasks()
4  Setup()
5  MakeInitialSolution()
6  OptimizeSolution()
```

No Algoritmo 1 apresenta-se o pseudocódigo do algoritmo desenvolvido. É recebido um plano, e devolvido um conjunto de soluções. Como se pode observar, o programa vai ser executado até serem atribuídas soluções para todos os dias, ou até ser atingido o tempo limite definido.

Para cada dia são efectuados uma série de procedimentos (Algoritmo 2), que irão ser aprofundados de seguida.

3.5.1 Fase Preliminar 1

Representa a função, na linha 2, do Algoritmo 2 - PreSetup.

Iremos considerar dois períodos de trabalho: manhã e tarde, sendo que isto é uma designação não necessariamente literal, visto que os períodos de trabalho estão, no fundo, dependentes das tarefas que existam para realizar.

Nesta fase são definidos os horários iniciais e finais para os dois períodos.

O horário inicial do primeiro período será no limite 8h da manhã. Mas se, por exemplo, a primeira tarefa a realizar for às 9h da manhã então o período começará às 9h da manhã. Para se definir o horário final do período é só somar o período inicial com o máximo horário diário definido. O horário final do segundo período será o horário da última tarefa. Para definir o horário inicial basta, então, subtrair o horário final com a máxima carga horária para um dia.

Isto significa que os dois períodos se podem intersectar.

3.5.2 Alargamento de Janelas Temporais

Representa a função, na linha 3, do Algoritmo 2 - ResizeTasks. Aqui irá ser feito um alargamento das janelas temporais das tarefas a realizar, no caso de ter sido inserido um valor superior a zero da percentagem a alargar. Tal como já foi referido, cada tarefa tem uma variável que permite, ou não, o seu alargamento. O alargamento é feito para ambos os sentidos. Ou seja, vamos supor que uma tarefa tenha uma janela temporal das 9h às 10h e a percentagem de alargamento seja de 20%. Os minutos de alargamento serão 12 (20% de 60 minutos). Tendo em conta que o alargamento é feito em ambos os sentidos, a nova janela temporal será das 8.54h às 10.06h.

3.5.3 Fase Preliminar 2

Representa a função, na linha 4, do Algoritmo 2 - Setup. O Setup é a fase preliminar da geração da primeira solução. Há uma série de passos importantes nesta fase.

Primeiro, começa-se por distribuir as tarefas a realizar pelos dois períodos definidos no PreSetup. É importante perceber se existe um equilíbrio de tarefas ao longo do dia, ou se, por exemplo, há uma clara frequência de tarefas a realizar no período da manhã.

Depois começa-se a gerar equipas com base na frequência de tarefas em cada período. Uma equipa é atribuída ao período da manhã, no caso de haver tarefas de manhã, enquanto que outra é atribuída ao período da tarde, no caso de haver tarefas. A partir daqui, tendo em conta o número de equipas inserido inicialmente, as restantes equipas serão distribuídas pelos períodos onde a duração total do tempo de tarefas é maior. O processo passa por dividir o tempo total de tarefas a realizar de cada período com o número de equipas que já estão atribuídas a cada período; o valor maior significará que

a seguinte equipa a atribuir irá ser associada a esse mesmo período; o processo repete-se até todas as equipas estarem atribuídas a um dos períodos.

Depois, serão atribuídos dois trabalhadores a cada equipa. Decidiu-se assim, devido à forma de trabalho existente na instituição do nosso caso de estudo, onde um paciente necessitará no máximo de dois trabalhadores por tarefa. Ainda assim, se o número total de trabalhadores não for par, naturalmente que um deles não terá parceiro. A existência de equipas é relevante para o caso de se seleccionar a opção de que um paciente tem de ser visitado pela mesma equipa num dia. E mais ainda, para o caso de existirem tarefas que necessitem de dois elementos para serem executadas.

O passo seguinte é inserir os nós relativamente aos almoços. Cada almoço terá uma janela temporal de duas horas, com uma respectiva duração de uma hora. Sendo que o início da janela temporal de cada almoço será o valor inserido nos parâmetros iniciais (lunchTime).

Para finalizar o Setup, irão ser adicionados os arcos. Voltando a recordar que um arco é composto por dois nós, aqui o processo passa por cruzar todos os nós e considerar, primeiro, os cruzamentos onde as janelas temporais dos nós verificam a admissibilidade do arco. Ou seja, se o início da janela temporal do primeiro nó mais a duração for inferior ao final da janela temporal do segundo nó. De seguida é verificado se o arco pertence ao período da manhã, tarde ou aos dois, e associa-se o arco a todos os trabalhadores do período onde o mesmo se insere.

3.5.4 Solução Inicial

No Algoritmo 3 apresenta-se o pseudocódigo da implementação da função (Algoritmo 2 - linha 5) que produz a primeira solução. A ideia geral passa por atribuir arcos a cada trabalhador até não ser mais possível, ou seja, até a primeira solução ter sido gerada ou até ser impossível continuar a formar a solução. Primeiro, é seleccionado o arco que minimizará o tempo total das rotas (Algoritmo 3 - linha 8), e depois, dependente do tipo de arco que é, o mesmo passará por uma segunda fase (Algoritmo 3 - linhas 9-24). Se não foi seleccionado nenhum arco, a rota do trabalhador é terminada. Sintetizando as variáveis a considerar no algoritmo:

- nodesSolution - representa a lista de tarefas atribuídas;
- stop - a variável booleana que impede a continuação do ciclo de atribuição de tarefas;

- noArc - identifica o número de trabalhadores aos quais não é possível serem atribuídas mais tarefas;

- checkWorkerIteration - é um vector de booleanos que serve para impedir que um trabalhador não escolha duas tarefas numa mesma iteração. Cada elemento do vector permite saber, se para um determinado trabalhador, já foi atribuída alguma tarefa na iteração corrente. Isto poderia-se verificar no caso de ser atribuído a um trabalhador (chamemos-lhe "A") uma tarefa onde sejam necessários dois trabalhadores. O trabalhador "A" iria então realizar a tarefa juntamente com o seu parceiro de equipa (neste caso, o "B"). Como o ciclo (Algoritmo 3 - linha 6) itera sobre os trabalhadores do plano, a não existência deste vector poderia levar a que fossem atribuídas duas tarefas ao trabalhador "B" (a tarefa inicialmente atribuída ao parceiro "A", e uma eventual segunda tarefa a ser-lhe atribuída directamente). Assim, é verificado, antes da escolha de uma tarefa, se a esse trabalhador já alguma foi atribuída na iteração corrente. No caso de já ter sido, o trabalhador só poderá escolher uma nova tarefa na iteração seguinte do ciclo;

- lastArcsTried - lista com o último arco que o algoritmo seleccionou para o trabalhador. Serve para impedir que se itere infinitamente sobre o mesmo arco. Imaginemos que um arco foi seleccionado a um determinado trabalhador na função choseArcWithBestTime(), mas por alguma razão, o mesmo acabou por não ser adicionado à solução: o que iria acontecer é que o algoritmo forçaria a inserção desse arco na solução infinitamente. Portanto, ao ser verificado que um mesmo arco foi seleccionado na última iteração do trabalhador e este não foi adicionado, dá-se por terminada a rota do trabalhador.

No Algoritmo 4 é explorada a função choseArcWithBestTime(), onde será seleccionado o arco que, tendo em conta as restrições existentes, minimizará o tempo total das rotas.

Um arco só poderá ser escolhido se a rota do trabalhador ainda não tiver sido dada como terminada, e se ainda não lhe tiver sido atribuído um arco na iteração corrente. O algoritmo pára se o trabalhador exceder o tempo máximo diário permitido (Algoritmo - linha 4).

A variável booleana checkOnlyType2 permite limitar a procura de nós a tarefas que só envolvem um trabalhador. Imaginemos que foi atribuída uma tarefa ao trabalhador "A", e que este é parceiro do trabalhador "B". Nesta iteração, o trabalhador "B" só poderá

Algorithm 3: Make Initial Solution

```
List< Node > nodesSolution
boolean stop ←false
int noArc ←0
boolean[] checkWorkerIteration
List < Arc > lastArcsTried

while stop is false do
    if noArc = totalWorkers then
        stop ← true
    for each i in checkWorkerIteration do
        i ← false
    end
    for each i in workers do
        Worker otherWorker = partner(i)
        aChosen ←choseArcWithBestTime(i, otherWorker, stop)
        if aChosen is null then
            noArc ← noArc + 1
            routeFinished of i ← true
        else if aChosen.finishNode.getTypeTransport() is 0 then
            if aChosen.finishNode.getType() is not 1 then
                notBedriddenTasks(i, aChosen) ;
            else if aChosen.finishNode.getType() is 1 then
                bedriddenTasks(i, otherWorker, aChosen) ;
        else if aChosen.finishNode.getTypeTransport() is 1 then
            transportToCenterNotPriorityTasks() ;
        else if aChosen.finishNode.getTypeTransport() is 2 then
            transportToCenterPriorityTasks() ;
        else if aChosen.finishNode.getTypeTransport() is 3 then
            transportFromCenterNotPriorityTasks() ;
        else if aChosen.finishNode.getTypeTransport() is 4 then
            transportFromCenterPriorityTasks() ;
        end
    end
end
```

seleccionar tarefas que envolvem um trabalhador, pois seria inútil incluir tarefas que necessitem de dois trabalhadores, visto que o seu parceiro já teria uma tarefa atribuída.

Será seleccionado o arco com o menor bestTime, dentro dos que satisfazem as condições. Ou seja, terá de ser um arco em que o nó de partida seja o mesmo que o nó actual do trabalhador (Algoritmo 4 - linha 11), um arco que não permita que o trabalhador almoce duas vezes (Algoritmo 4 - linha 12), e um arco onde o tempo total de espera não exceda o tempo permitido (Algoritmo 4 - linha 13). É ainda verificado se o plano exige a escolha de arcos que permitam que um paciente seja sempre visitado pela mesma equipa (Algoritmo 4 - lLinha 20).

Após a selecção do arco, seguir-se-á a segunda fase, onde se vai confirmar, ou não, a adição do arco à solução. Primeiro verifica-se que tipo de arco se trata, ou seja, o tipo de tarefa atribuída, e depois o algoritmo prossegue conforme o tipo de tarefa seleccionada. Foram considerados seis diferentes tipos de tarefas:

1. Tarefa **normal**, que necessite de um trabalhador (Algoritmo 5);
2. Tarefa **normal**, que necessite de dois trabalhadores (Algoritmo 6);
3. Tarefa de **transporte** de casa para o centro, sem prioridade (Algoritmo 7);
4. Tarefa de **transporte** de casa para o centro, com prioridade (Algoritmo 8);
5. Tarefa de **transporte** do centro para casa, sem prioridade (Algoritmo 9);
6. Tarefa de **transporte** de centro para casa, com prioridade (Algoritmo 10);

3.5.4.1 Tarefa normal, que necessite de um trabalhador

Antes de se adicionar a tarefa à solução, o paciente fica associado à equipa do trabalhador (Algoritmo 5 - linha 4); se a tarefa seleccionada for relativa ao almoço, o trabalhador fica "marcado" como almoçado (Algoritmo 5 - linha 6) e, naturalmente, impedido de voltar a almoçar nesse mesmo dia; no caso do trabalhador não ter nenhuma carrinha associada, aquela com maior capacidade disponível irá ser-lhe atribuída.

O arco é adicionado à solução (Algoritmo 5 - linha 10), sendo que os melhores tempos entre o nó adicionado e os próximos nós possíveis a visitar serão actualizados, tendo em conta o tempo corrente (Algoritmo 5 - linha 12).

Algorithm 4: ChoseArcWithBestTime

```
1  Input Worker i, Worker otherWorker, boolean stop Output Arc aChosen

2  if i.routeFinished() is false & checkWorkerIteration[id of i] is false then
3      if (i.currentTime() - i.startTime()) > i.maxTime() then
4          stop ← true
       end
       else
5          int bestCurrentTime ← Integer.MAXvalue
6          Arc arcChosen ← null
7          boolean checkOnlyType2 ← false
8          if checkWorkerIteration[id of otherWorker] is true then
9              checkOnlyType2 ← true
           end
10         for each a in arcs do
11             if a.getWorker() = i & id of a.getStartNode() = id of i.currentNode() then
12                 if i.hadlunch() is false or (i.hadlunch() is true & name of a.finishNode() is not "lunch") then
13                     if a.waitTime() <= maxWaitTimeBetweenTasks then
14                         if a.bestTime() >= 0 & a.bestTime() <= bestCurrentTime & a.bestTime() >=
                           i.startTime() & a.bestTime() <= i.finishTime() then
15                             if a is not in nodesSolution then
16                                 patient ←a.finishNode().patient()
17                                 if patient is null then
18                                     bestCurrentTime ←a.bestTime()
19                                     aChosen ← a
                                   end
                                   else
20                                     if ((sameTeamCondition is true & patient.team() is not null &
                                       patient().team() = i.team()) or (sameTeamCondition is true &
                                       patient.team() is null) or (sameTeamCondition is false)) then
21                                         if checkOnlyType2 is true then
22                                             if a.finishNode().type() = 2 then
23                                                 bestCurrentTime ←a.bestTime()
24                                                 aChosen ← a
                                               end
                                           end
                                           else
25                                             bestCurrentTime ←a.bestTime()
26                                             aChosen ← a
                                           end
                                       end
                                   end
                               end
                           end
                       end
                   end
               end
           end
       end
   end
```

Algorithm 5: notBedriddenTasks

```
1  Input Worker i, Arc aChosen

2  checkWorkerIteration[id of i] ← true
3  Patient patient ← aChosen.finishNode().patient()
4  patient.setTeam(i.team())

5  if name of aChosen.finishNode() is "lunch" then
6  |   i.hadLunch() ←true
7  |   clearVanOfWorker(i)
   end
8  if i.currentVan() is null then
9  |   setVan(i, bestTime of aChosen)
   end

10 addArcToSolution(aChosen, i)
11 nodesSolution.add(aChosen.finishNode())
12 updateArcsBestTimes(aChosen, i)
```

3.5.4.2 Tarefa normal, que necessite de dois trabalhadores

Inicialmente é verificado se a última tarefa seleccionada para o trabalhador em questão, é precisamente a tarefa escolhida. No caso de ser, a rota do trabalhador é terminada (Algoritmo 6 - linha 5), pois como já foi referido anteriormente, estariamos perante um caso onde infinitamente iria ser sempre escolhida esta tarefa, e a mesma nunca seria adicionada à solução devido a alguma condição posterior do algoritmo.

De seguida, é verificado se o parceiro do trabalhador relativo à tarefa em questão já realizou alguma tarefa nesta iteração. Se não o fez, o algoritmo prossegue (Algoritmo 6 - linha 7). Posteriormente, é determinado o arco do parceiro, que será o arco entre o seu nó corrente e o nó relativo à tarefa a ser realizada. Os tempos de chegada dos dois parceiros ao nó são comparados, de maneira a perceber o tempo extra que o trabalhador que chega primeiro terá de esperar (Algoritmo 6 - linhas 17-22).

O resto do processo é idêntico ao tipo de tarefa anterior: se os trabalhadores não têm uma carrinha associada, alguma lhes irá ser atribuída (Algoritmo 6 - linhas 23-26); os arcos são adicionados à solução (Algoritmo 6 - linhas 29-30), e os tempos para os possíveis próximos nós são actualizados (Algoritmo 6 - linhas 35-36).

Algorithm 6: bedriddenTasks

```
1  Input Worker i, Worker otherWorker, Arc aChosen

2  if lastArcsTried(i) is not null then
3      if id of aChosen = id of lastArcsTried(i) then
4          noArc ←noArc + 1
5          routeFinished of i ←true
       end
   end
6  lastArcsTried.add(i, aChosen)
7  if checkWorkerIteration [id of otherWorker] = false then
8      checkWorkerIteration[id of otherWorker] ←true
9      startNode ←otherWorker.currentNode()
10     finishNode ←aChosen.finishNode()
11     arcOther ←getArcByInfo(otherWorker,startNode,finishNode)
12     if arcOther is not null then
13         if arcOther.waitTime() <= maxWaitTimeBetweenTasks then
14             if arcOther.bestTime() >= 0 & arcOther.bestTime() <= bestCurrentTime & arcOther.bestTime() >=
                   arcOther.startTime() & arcOther.bestTime() <= arcOther.finishTime() then
15                 bestTime1 ←aChosen.bestTime()
16                 bestTime2 ←arcOther.bestTime()
17                 if bestTime1 >= bestTime2 then
18                     extraTime ←bestTime1 - bestTime2
19                     bestTime of arcOther ←newBestTimeVal(bestTime1, extraTime)
                   end
20                 else
21                     extraTime ←bestTime2 - bestTime1
22                     bestTime of arcOther ←newBestTimeVal(bestTime2, extraTime)
                   end
23                 if i.currentVan() is null then
24                     setVan(i, aChosen.bestTime())
                   end
25                 if otherWorker.currentVan() is null then
26                     setVan(otherWorker, bestTime of arcOther)
                   end
27                 aChosen.setVanAndCapacityAtTheTime()
28                 arcOther.setVanAndCapacityAtTheTime()
29                 addArcToSolution(aChosen, i)
30                 addArcToSolution(arcOther, otherWorker)
31                 aChosen.finishNode.visitTeam ←team of i
32                 Patient patient ←aChosen.finishNode().patient()
33                 patient.setTeam(i.team())
34                 nodesSolution.add(finishNode of aChosen)
               end
           end
35         updateArcsBestTimes(aChosen, i)
36         updateArcsBestTimes(aChosen, otherWorker)
       end
   end
```

3.5.4.3 Tarefa de transporte de casa para o centro, sem prioridade

Neste tipo de tarefas começa-se por adicionar um arco à solução que consiste no nó corrente do trabalhador até ao paciente a ser transportado (Algoritmo 7 - linha 6). Depois, serão adicionados, numa lista, todos os nós relativos a outros pacientes que também pretendam ser transportados para o centro, com uma janela temporal compreendida entre o primeiro paciente a ser transportado (Algoritmo 7 - linha 11). Tendo em conta a capacidade da carrinha corrente do trabalhador responsável pelo transporte, irão ser adicionados mais arcos à solução que consistem no nó do paciente corrente até ao nó do paciente seguinte. A escolha da ordem pela qual os pacientes são recolhidos é sempre feita de maneira a minimizar os tempos de rotas (Algoritmo 7 - linhas 13-27).

Após terem sido recolhidos todos os pacientes, é gerado um arco para o centro, de modo a que os pacientes sejam deixados no centro (Algoritmo 7 - linha 29). Posteriormente, são criados arcos desde o centro até às tarefas que ainda estão por realizar (Algoritmo 7 - linha 34). Este passo é necessário porque estes arcos não poderiam ter sido criados na fase do Setup.

3.5.4.4 Tarefa de transporte de casa para o centro, com prioridade

Neste caso, o paciente é transportado directamente para o centro, sem existir recolha de outros pacientes. No resto, o processo é idêntico ao tipo de tarefa anterior.

3.5.4.5 Tarefa de transporte do centro para casa, sem prioridade

Neste tipo de tarefa, a ideia é idêntica às tarefas de transporte para o centro, mas neste caso, em sentido inverso.

Inicialmente é gerado um arco para o centro, desde o nó actual do trabalhador, de modo a puder ir fazer a recolha dos pacientes (Algoritmo 9 - linha 3). É guardado numa lista, os nós a visitar, tendo em conta os pacientes que pretendem ser levados para casa (Algoritmo 9 - linha 9). É adicionado à solução um arco do centro para a casa do primeiro paciente (Algoritmo 9 - linha 10). Depois, é percorrido um ciclo que irá permitir a formação de arcos até às casas dos restantes pacientes, sempre de modo a minimizar o tempo das rotas (Algoritmo 9 - linhas 22-35).

3.5.4.6 Tarefa de transporte do centro para casa, com prioridade

Neste último caso, o processo é idêntico ao tipo de transporte anterior, mas sem as particularidades de se transportar mais que um paciente.

Algorithm 7: transportToCenterNotPriorityTasks

```
1  Input Worker i, Arc aChosen, Arc arcOther

2  if i.currentVan() is null then
3  |   setVan(workers(i), arcOther.bestTime())
   end
4  i.currentVan() ← patientEntered()
5  aChosen.setVanAndCapacityAtTheTime(i.currentVan(), i.currentVan().currentCapacity())
6  addArcToSolution(aChosen, i)
7  aChosen.finishNode().visitTeam() ←team of i
8  Patient patient ←aChosen.finishNode().patient()
9  patient.setTeam(i.team())
10 nodesSolution.add(aChosen.finishNode())
11 sameTWnodes ←getNodesTW(aChosen.finishNode())
12 fullVan ← false
13 while fullVan is false do
14 |   if size of sameTWnodes is 0 then
15 |   |   fullVan ←true
   |   end
16 |   if currentCapacity of currentVan of workers(i) is 0 then
17 |   |   fullVan ←true
   |   end
18 |   sameTWarcs ←getArcsTW(sameTWnodes, i, i.currentNode())
19 |   arcMin ← getMinBestTimeFromArcs(sameTWarcs)
20 |   sameTWnodes.remove(arcMin.finishNode());
21 |   if arcMin is not in nodesSolution then
22 |   |   if i.currentVan().currentCapacity() is > 0 then
23 |   |   |   i.currentVan() ← patientEntered()
24 |   |   |   aChosen.setVanAndCapacityAtTheTime( i.currentVan(), i.currentVan().currentCapacity())
25 |   |   |   addArcToSolution(arcMin, i)
26 |   |   |   nodesSolution.add(aChosen.finishNode())
27 |   |   |   updateArcsBestTimes(aChosen, i)
   |   |   end
   |   end
   end
28 newNode ← nodes.NewNode("CenterTransportBegin")
29 newArc ←arcs.NewArc(i.currentNode(), newNode)
30 newArcToCenter.setNewBestTime(i.currentTime())
31 addArcToSolution(newArcToCenter, i)
32 clearVanOfWorker(i)
33 nodesSolution.add(newArcToCenter.finishNode())
34 generateMoreArcs(newArcToCenter, i)
35 updateArcsBestTimes(aChosen, i)
```

Algorithm 8: transportToCenterPriorityTasks

```
1  Input Worker i, Arc aChosen, Arc arcOther
2  if i.currentVan() is null then
3  |   setVan(i, arcOther.bestTime())
   end
4  i.currentVan() ← patientEntered()
5  aChosen.setVanAndCapacityAtTheTime( i.currentVan(),
   i.currentVan().currentCapacity())
6  addArcToSolution(aChosen, i)
7  aChosen.finishNode().visitTeam() ←i.team()
8  Patient patient ←aChosen.finishNode().patient()
9  patient.setTeam(i.team())
10 nodesSolution.add(aChosen.finishNode())
11 updateArcsBestTimes(aChosen, i)
12 newNode ← nodes.NewNode("CenterTransportBegin")
13 newArcToCenter ←arcs.NewArc(i.currentNode(), newNode)
14 newArcToCenter.setNewBestTime(i.currentTime())
15 addArcToSolution(newArcToCenter, i)
16 clearVanOfWorker(i)
17 nodesSolution.add(newArcToCenter.finishNode())
18 generateMoreArcs(newArcToCenter, i)
19 updateArcsBestTimes(aChosen, i)
```

3.5.5 Solução Optimizada

Ao ficar concluída a primeira solução, esta é guardada, simultaneamente, como aquela onde há uma minimização do tempo total de rotas, a que apresenta a minimização do tempo total de espera e a mais justa.

Se nas variáveis de entrada, o utilizador seleccionar a procura exaustiva, irá então ser efectuada uma troca exaustiva de tarefas onde se procurará obter soluções que melhorem os três parâmetros relativos à solução inicial: minimização do tempo total de rotas; minimização do tempo total de espera; justiça.

O primeiro passo passa por fazer a geração de todas as combinações de pares possíveis entre os trabalhadores existentes (Algoritmo 11 - Linha 2). Ou seja, se na primeira solução quatro trabalhadores (A, B, C, D) tiverem efectuado rotas, as combinações serão: (A-B; A-C; A-D; B-C; B-D; C-D). Para cada combinação de trabalhadores será então efectuada um conjunto de trocas de tarefas (Algoritmo 11 - Linha 12). Por cada troca

Algorithm 9: transportFromCenterNotPriorityTasks

```
1  Input Worker i, Arc aChosen, Arc arcOther
2  newNode ← nodes.NewNode("CenterTransportFinal")
3  newArcToCenter ←arcs.NewArc(i.currentNode(), newNode)
4  newArcToCenter.setNewBestTime(i.currentNode())
5  newArcToCenter.setVanAndCapacityAtTheTime(i.currentVan(), i.currentVan().currentCapacity())
6  addArcToSolution(newArcToCenter, i)
7  nodesSolution.add(newArcToCenter.finishNode())
8  clearVanOfWorker(i)
9  sameTWnodes ←getNodesTW(aChosen.finishNode())
10 newArcCenterToHome ←arcs.NewArc(i.currentNode(), newNode)
11 newArcCenterToHome.setNewBestTime(i.currentTime())
12 if i.currentVan() is null then
13 |  setVan(i, arcOther.bestTime())
   end
14 for each node in sameTWnodes do
15 |  i.currentVan() ←patientEntered()
   end
16 i.currentVan() ← patientLeft()
17 newArcCenterToHome.setVanAndCapacityAtTheTime(i.currentVan(), i.currentVan().currentCapacity())
18 addArcToSolution(newArcCenterToHome, i)
19 nodesSolution.add(newArcCenterToHome.finishNode())
20 updateArcsBestTimes(newArcCenterToHome, i)
21 emptyVan ← false
22 while emptyVan is false do
23 |  if size of sameTWnodes is 0 then
24 |  |  fullVan ←true
   |  end
25 |  if i.currentVan().currentCapacity() is 0 then
26 |  |  fullVan ←true
   |  end
27 |  sameTWarcs ←getArcsTW(sameTWnodes,i, i.currentNode())
28 |  arcMin ← getMinBestTimeFromArcs(sameTWarcs)
29 |  sameTWnodes.remove(arcMin.finishNode());
30 |  if arcMin is not in nodesSolution then
31 |  |  i.currentVan() ← patientLeft()
32 |  |  aChosen.setVanAndCapacityAtTheTime(i.currentVan(), i.currentVan().currentCapacity())
33 |  |  addArcToSolution(arcMin, i)
34 |  |  nodesSolution.add(aChosen.finishNode())
35 |  |  updateArcsBestTimes(aChosen, i)
   |  end
   end
```

Algorithm 10: transportFromCenterPriorityTasks

```
1  Input Worker i, Arc aChosen, Arc arcOther
2  newNode ← nodes.NewNode("CenterTransportFinal")
3  newArcToCenter ←arcs.NewArc(i.currentNode(), newNode)
4  newArcToCenter.setNewBestTime(i.currentTime())
5  newArcToCenter.setVanAndCapacityAtTheTime(i.currentVan(),
   i.currentVan().currentCapacity())
6  addArcToSolution(newArcToCenter, i)
7  nodesSolution.add(newArcToCenter.finishNode())
8  clearVanOfWorker(i)
9  newArcCenterToHome ←arcs.NewArc( i.currentNode(), aChosen.finishNode())
10 newArcCenterToHome.setNewBestTime(i.currentTime())
11 if i.currentVan() is null then
12 |   setVan(i, newArcCenterToHome.bestTime())
   end
13 aChosen.setVanAndCapacityAtTheTime(i.currentVan(),
   i.currentVan().currentCapacity())
14 addArcToSolution(newArcCenterToHome, i)
15 nodesSolution.add(newArcCenterToHome.finishNode())
16 updateArcsBestTimes(newArcCenterToHome, i)
```

de tarefas, será testada a solução, ou seja verificar-se-á se é uma solução admissível, e se apresenta melhorias em relação à melhor solução corrente (Algoritmo 11 - Linha 26). Serão consideradas todas as possíveis trocas de tarefas, até um máximo de 12 trocas, de modo a não sobrecarregar o algoritmo com um excesso de trocas. Isto significa que primeiro consideramos apenas uma troca de tarefas, e são feitas todas as combinações possíveis (Exemplo: Ao trabalhador A foi atribuído as tarefas T1, T2, T3; enquanto que ao B, foi atribuído as tarefas T4, T5, T6. Após a primeira troca, o trabalhador A passa a estar associado às tarefas T4, T2, T3; enquanto que o trabalhador B passa a estar encarregue das tarefas T1, T5, T6. Depois de testar a nova solução, segue-se uma nova combinação de troca de tarefas, e por aí sucessivamente). Depois o processo segue, realizando todas as combinações para duas trocas entre trabalhadores, e assim sucessivamente até o número de trocas ser o número total de tarefas que o trabalhador com menos tarefas entre os dois tem ou, como já foi referido, 12. No caso do número total de tarefas de um trabalhador ser superior a 12, serão escolhidos de maneira aleatória, 12 tarefas para trocar.

Depois de concluídas as trocas, a ideia da formação de uma nova solução é exactamente a mesma da formação da solução inicial, mas desta vez o conjunto de tarefas a realizar por cada trabalhador já está pré-definido (Algoritmo 11 - Linha 26). Importa então determinar a ordem das tarefas a realizar e aferir a sua admissibilidade. No caso da solução ser admissível, é então testado se a nova solução melhora um dos três parâmetros relativos à solução inicial. Se sim, a solução é guardada.

No final, são retornadas a primeira solução; a solução que apresenta a maior minimização do tempo total de rotas; a solução que apresenta a maior minimização do tempo total de espera; e a mais justa.

Algorithm 11: Optimize Solution

```
1  Input Solution firstSolution

2  comboWorkers ←getComboWorkers(firstSolution)
3  for each combo in comboWorkers do
4      workerSend ←combo.workerSend
5      workerReceive ←combo.workerReceive
6      if workerReceive.getRoutes.size > workerSend.getRoutes.size then
7          workerAux ←workerSend
8          workerSend ←workerReceive
9          workerReceive ←workerAux
       end
10     for i=1; i<workerSend.getRoutes.size; i++ do
11         numberOfSwipes ←i
12         nodesToSwipe ←getNodesToSwipe(firstSolution, workerSend,
           workerReceive, numberOfSwipes)
13         for each swipe in nodesToSwipe do
14             beforeSwipeNodesSendWorker ←workerSend.getRoutesNodes()
15             beforeSwipeNodesReceiveWorker ←workerReceive.getRoutesNodes()
16             afterSwipeNodesSendWorker ←beforeSwipeNodesSendWorker
17             afterSwipeNodesReceiveWorker ←beforeSwipeNodesReceiveWorker
18             nodesToSend ←swipe.nodesToSend
19             nodesToReceive ←swipe.nodesToReceive
20             for each node in nodesToSend do
21                 afterSwipeNodesSendWorker.remove(node)
22                 afterSwipeNodesReceiveWorker.add(node)
               end
23             for each node in nodesToReceive do
24                 afterSwipeNodesReceiveWorker.remove(node)
25                 afterSwipeNodesSendWorker.add(node)
               end
26             testNewSolution(firstSolution, workerSend, workerReceive
               afterSwipeNodesSendWorker, afterSwipeNodesReceiveWorker,
               nodesToSend, nodesToReceive)
           end
       end
   end
```

Capítulo 4

Validação do Algoritmo

Neste capítulo procurámos validar o trabalho realizado. Considerámos crucial efectuar primeiro a validação do algoritmo, antes de transitarmos para a sua integração numa plataforma web, pois esta só teria valor, se o algoritmo também o tivesse.

De modo a averiguar a eficiência do nosso algoritmo, pretendíamos comparar os resultados obtidos com os resultados publicados na literatura científica que abordem problemas similares. A ideia consistia em realizar um conjunto de testes pré-definidos e comparar os valores obtidos com o de outras soluções, de modo a perceber se a nossa poderia ser considerada "boa", e ainda comparar os tempos de execução. Tal não sucedeu devido à dificuldade de comparar justamente as soluções na literatura com a nossa solução.

Foram encontrados cerca de 11 artigos que disponibilizavam um conjunto de *benchmarks*. No entanto, tornou-se impossível realizar uma comparação adequada visto que as funções-objectivo deste conjunto de artigos não serem as mesmas que as nossas. Ou seja, no nosso caso o objectivo passa por minimizar o tempo total de trabalho dos trabalhadores, enquanto que nos artigos analisados o objectivo passava, por exemplo, por minimizar o custo total das rotas, o tempo total das rotas, ou a distância das rotas.

Houve de facto, um artigo (Trautsamwieser e Hirsch 2014) em que o objectivo era o de minimizar o tempo total de trabalho. No entanto, as diferenças entre esta e a nossa solução impediram uma comparação entre ambas as soluções. Neste artigo, os trabalhadores não partem de um centro comum; existe ainda um conjunto de níveis

de qualificação, o que leva a que as tarefas não possam ser realizadas por qualquer trabalhador.

Este capítulo está dividido em quatro partes: a primeira com um teste relativo a um caso real; as três seguintes com testes, respectivamente, de pequena, média e grande dimensão, sendo que cada uma destas três estão sub-divididas em resultados e uma posterior análise aos mesmos. Todos os testes foram executados num computador pessoal, num Windows de 64 bits; processador Intel i5 em 2.20 GHz; e 8GB de memória RAM.

Antes de se passar às secções mencionadas, segue-se a legenda que serve de apoio à compreensão das Tabelas deste capítulo:

- TN1 - Tarefas normais, que necessitem de um trabalhador;
- TN2 - Tarefas normais, que necessitem de dois trabalhadores;
- TTS - Tarefas de transporte, sem prioridade;
- TTC - Tarefas de transporte, com prioridade;
- D - Número de dias;
- MTE - Máximo Tempo de Espera Entre Tarefas;
- CC - Continuidade de Cuidado, que significa a obrigatoriedade do mesmo Paciente ser visitado pela mesma equipa;
- AJT - Alargamento das Janelas Temporais (%);
- Sol - Solução encontrada;
- Sol-1 - Primeira solução encontrada;
- Sol-TTT - Solução encontrada com menor tempo total de trabalho;
- Sol-TTE - Solução encontrada com menor tempo total de espera entre tarefas;
- Sol-J - Solução encontrada com maior justiça;
- SL - Sem limite;
- 2h - 2 horas;
- 30m - 30 minutos;

- T(s) - Tempo da performance, em segundos;
- T(m) - Tempo da performance, em minutos;
- S - Sim;
- N - Não;
- N (x / y) - Não (número de tarefas realizadas / número de tarefas totais);
- TTT - Tempo total de trabalho (minutos);
- TTE - Tempo total de espera entre tarefas (minutos);
- J - Justiça (%);
- M | T - Frequência (%) de tarefas no peíodo da manhã (M) e da tarde (T);
- TR - Tipo de transporte;
- C - Transporte de carro;
- P - Transporte a pé;
- TO - Tempo limite de execução do algoritmo excedido.

4.1 Caso Real

Primeiro, efectuámos um teste real relativo ao nosso caso de estudo, com o objectivo de perceber se a nossa solução seria capaz de ajudar a instituição a rotear os mapeamentos necessários.

Resultados

O teste consistia no problema relativo ao dia de 1 de Outubro de 2018, sendo que todos os dados relativos ao problema foram-nos fornecidos pela instituição. Este problema envolvia 2 auxiliares, 62 pacientes e cerca de 77 tarefas, sendo que 57 são relativas a tarefas de transporte, e as restantes 20 são tarefas de apoio domiciliário (TN1). Todos os pacientes têm uma posição geográfica localizada na zona de Algés, Portugal, cidade onde se situa a instituição. Não foi possível obter todos os dados necessários à resolução do problema, nomeadamente a duração prevista de cada tarefa, a possibilidade, ou não, de puder alargar a janela temporal e a hora de almoço dos auxiliares. Pela análise efectuada aos dados recebidos, pudemos constatar que algumas das tarefas foram realizadas

fora da janela temporal prevista, pelo que decidimos considerar um alargamento de 30%; a duração de cada tarefa foi bastante curta, pelo que decidimos considerar uma duração de 2 minutos para cada tarefa; e pela ausência de tarefas entre as 14h e as 16h, considerámos, então 14h como a hora de almoço.

Análise

Os resultados foram bastante positivos (Tabela 4.1). O algoritmo encontrou uma solução em menos de 1 segundo, sendo que a otimização foi concluída ao fim de 35 minutos. As soluções otimizadas não apresentaram melhorias ao nível da minimização do tempo total de espera; houve sim uma melhoria ao nível da minimização do tempo total de rotas: 1108 minutos para 521 minutos; e da justiça do tempo de trabalho: 13.6% para 1.7%, o que comprova claramente a versatilidade e a adequabilidade do nosso algoritmo.

Apesar do resultado positivo, decidimos ainda desenvolver um outro conjunto de testes, de maneira a se puder perceber o real potencial e valor da meta-heurística desenvolvida, e a validar o trabalho do algoritmo.

Tabela 4.1: Caso Real - Teste 1

TN1 (26%) | TN2 (0%) | TTS (74%) | TTC (0%)								
MTE	CC	AJT	Sol	Sol-1	Sol-TTT	Sol-TTE	Sol-J	T(m)
SL	N	30	S	TTT - 1107.99 TTE - 718.22 J - 13.63	TTT - 521.62 TTE - 731.76 J - 100.40	TTT - 1107.99 TTE - 718.22 J - 13.63	TTT - 989.34 TTE - 1188.84 J - 1.71	35.62

4.2 Dimensão Reduzida

Nos testes relativos a esta secção, e às duas seguintes, variámos os diferentes parâmetros envolvidos, de modo a perceber a resposta do algoritmo em diferentes situações.

Resultados

Nesta secção, foram realizados testes considerando 1 dia de planeamento; um alargamento das janelas temporais em 0%, 20% e 100%; o máximo tempo de espera entre tarefas considerado foi de 30 minutos, 2 horas ou sem limite; uma variação da frequência das tarefas de cada tipo; e obrigatoriedade, ou não, de cada paciente ser tratado pelo mesmo trabalhador. Os restantes parâmetros não foram alterados: o máximo trabalho diário de cada trabalhador foi estabelecido como 8 horas; a hora de almoço foi estipulada para as 13h; as tarefas têm todas uma duração de 30 minutos, com uma janela

temporal de 2 horas, e uma posição geográfica mais alargada que nos dados do caso de estudo. O tempo limite de execução do programa foi estabelecido para 1 hora.

Nesta secção foram, então, realizados testes em instâncias de dimensão reduzida, com apenas 9 tarefas e 2 auxiliares (1 equipa), sendo que o primeiro (Tabela 4.2) passava por efectuar um teste, onde houvesse uma distribuição igual dos diferentes tipos de tarefas (25% para cada tipo).

Nas Tabelas seguintes (Tabela 4.3 à 4.6) testámos o algoritmo para situações em que cada tipo de tarefa pudesse não ocorrer, ou seja, primeiro fizemos um teste (Tabela 4.3) sem tarefas do tipo 1 (Normais - 1 Auxiliar), e com uma distribuição uniforme de tarefas pelos 3 restantes tipos; depois outro teste (Tabela 4.4) sem tarefas do tipo 2 (Normais - 2 Auxiliares), e por aí sucessivamente. O objectivo passava por perceber se a diferença de resultados seria, ou não, significativa ao não considerar um dos tipos de tarefa.

Tabela 4.2: Tamanho Reduzido - Teste 1

TN1 (25%) | TN2 (25%) | TTS (25%) | TTC (25%)								
MTE	CC	AJT	Sol	Sol-1	Sol-TTT	Sol-TTE	Sol-J	T(s)
SL	N	0	N (13 / 15)	-	-	-	-	<5
SL	N	20	N (13 / 15)	-	-	-	-	<5
SL	N	100	N (14 / 15)	-	-	-	-	<5
2h	N	0	N (11 / 15)	-	-	-	-	<5
2h	N	20	N (11 / 15)	-	-	-	-	<5
2h	N	100	N (14 / 15)	-	-	-	-	<5
30m	N	0	N (6 / 13)	-	-	-	-	<5
30m	N	20	N (6 / 13)	-	-	-	-	<5
30m	N	100	N (7 / 13)	-	-	-	-	<5

Análise

Os primeiros testes permitiram perceber que uma distribuição idêntica dos tipos de tarefas a realizar poderá levar à não conclusão do algoritmo. Todos os testes efectuados para este caso acabaram por ser interrompidos, visto que não se gerou a solução inicial (Tabela 4.2). Nos casos onde não foi encontrada uma solução, é referida uma comparação com o número de tarefas atribuídas com o número de tarefas a atribuir. Este valor é superior ao número de tarefas inicialmente citado, pois contém todas as tarefas extra

Tabela 4.3: Tamanho Reduzido - Teste 2

TN1 (0%) | TN2 (33.33%) | TTS (33.33%) | TTC (33.33%)								
MTE	CC	AJT	Sol	Sol-1	Sol-TTT	Sol-TTE	Sol-J	T(s)
SL	N	0	S	TTT - 848.58 TTE - 428.79 J - 2	TTT - 558.90 TTE - 129.7 J - 10.5	TTT - 558.90 TTE - 129.7 J - 10.5	TTT - 848.97 TTE - 428.79 J - 1.8	<5
SL	N	20	S	TTT - 824.58 TTE - 404.79 J - 2.06	TTT - 546.90 TTE - 117.69 J - 14.57	TTT - 546.90 TTE - 117.69 J - 14.57	TTT - 848.97 TTE - 428.79 J - 1.8	<5
SL	N	100	S	TTT - 728.97 TTE - 308.79 J - 2.25	TTT - 498.90 TTE - 70.14 J - 30.85	TTT - 498.90 TTE - 70.14 J - 30.85	TTT - 728.97 TTE - 308.79 J - 2.25	<5
2h	N	0	S	TTT - 848.58 TTE - 428.79 J - 2	TTT - 558.90 TTE - 129.7 J - 10.5	TTT - 558.90 TTE - 129.7 J - 10.5	TTT - 848.97 TTE - 428.79 J - 1.8	<5
2h	N	20	S	TTT - 824.58 TTE - 404.79 J - 2.05	TTT - 546.90 TTE - 117.69 J - 14.57	TTT - 546.90 TTE - 117.69 J - 14.57	TTT - 824.97 TTE - 404.79 J - 1.93	<5
2h	N	100	N (14 / 15)	-	-	-	-	<5
30m	N	0	N (6 / 13)	-	-	-	-	<5
30m	N	20	N (6 / 13)	-	-	-	-	<5
30m	N	100	N (7 / 13)	-	-	-	-	<5

que eventualmente possam ter sido acrescentadas: desde almoços aos trabalhadores a viagens entre pacientes, de modo a ser efectuado o transporte de pacientes. De qualquer forma, é relevante referir que a maioria das tarefas que ficaram por realizar são referentes a almoços e não às tarefas propriamente ditas. Ou seja, em muitos dos casos, todas as tarefas foram de facto concluídas, mas por existirem tarefas extra relativas aos almoços que ficaram por atribuir a alguns dos trabalhadores, o algoritmo acabou por não concluir a primeira fase com sucesso.

Em relação aos testes seguintes (Tabelas 4.3 à 4.6), constatámos que é precisamente no primeiro teste (Tabela 4.3), onde não foram consideradas tarefas normais que necessitem de apenas 1 auxiliar, que são obtidos os melhores resultados, a nível de minimização total do tempo total de trabalho e minimização do total tempo total de espera. Em relação à justiça, notámos que os piores resultados acontecem no último teste (Tabela 4.6), onde não são consideradas tarefas de transporte, com prioridade. É importante recordar que quanto menor for o valor relativo à justiça, maior ela será, pois este valor representa a diferença entre os auxiliares com maior e menor tempos totais de rota.

Tabela 4.4: Tamanho Reduzido - Teste 3

TN1 (33.33%) | TN2 (0%) | TTS (33.33%) | TTC (33.33%)								
MTE	CC	AJT	Sol	Sol-1	Sol-TTT	Sol-TTE	Sol-J	T(s)
SL	N	0	S	TTT - 918.98 TTE - 446.76 J - 0	TTT - 879.79 TTE - 368.57 J - 8.64	TTT - 880.01 TTE - 359.32 J - 8.64	TTT - 918.98 TTE - 446.76 J - 0	<5
SL	N	20	S	TTT - 894.98 TTE - 422.76 J - 0	TTT - 855.79 TTE - 344.57 J - 8.87	TTT - 856.01 TTE - 335.32 J - 8.87	TTT - 894.98 TTE - 422.76 J - 0	<5
SL	N	100	S	TTT - 798.30 TTE - 325.873 J - 0	TTT - 760.48 TTE - 279.00 J - 9.94	TTT - 760.48 TTE - 279.00 J - 9.94	TTT - 798.30 TTE - 325.873 J - 0	<5
2h	N	0	S	TTT - 918.98 TTE - 446.76 J - 0	TTT - 879.79 TTE - 368.57 J - 8.64	TTT - 880.01 TTE - 359.32 J - 8.64	TTT - 918.98 TTE - 446.76 J - 0	<5
2h	N	20	S	TTT - 894.98 TTE - 422.76 J - 0	TTT - 855.79 TTE - 344.57 J - 8.87	TTT - 856.01 TTE - 335.32 J - 8.87	TTT - 894.98 TTE - 422.76 J - 0	<5
2h	N	100	N (13 / 16)	-	-	-	-	<5
30m	N	0	N (3 / 12)	-	-	-	-	<5
30m	N	20	N (3 / 12)	-	-	-	-	<5
30m	N	100	N (8 / 14)	-	-	-	-	<5

A obrigatoriedade do mesmo paciente ser visitado pela mesma equipa não é relevante na performance do algoritmo devido ao pequeno tamanho do problema, e de no caso específico dos testes, ter sido considerado apenas dois trabalhadores, ou seja, uma equipa.

O tempo da performance para os testes desta secção foram baixos, não tendo em nenhum caso ter sido ultrapassado os 5 segundos.

4.3 Dimensão Média

4.3.1 20 Tarefas / 4 Trabalhadores

De seguida passou-se a testes de maior dimensão, onde por exemplo, a obrigatoriedade do mesmo paciente ser visitado pela mesma equipa já poderia ser testada pois estávamos a lidar com casos com múltiplas equipas. As restrições consideradas foram semelhantes aos testes da secção anterior, com algumas diferenças, que vão ser explicitadas de

Tabela 4.5: Tamanho Reduzido - Teste 4

TN1 (33.33%) | TN2 (33.33%) | TTS (0%) | TTC (33.33%)								
MTE	CC	AJT	Sol	Sol-1	Sol-TTT	Sol-TTE	Sol-J	T(s)
SL	N	0	S	TTT - 909.08 TTE - 547.74 J - 1.87	TTT - 763.72 TTE - 287.19 J - 33.59	TTT - 764.00 TTE - 274.56 J - 32	TTT - 909.08 TTE - 547.74 J - 1.87	<5
SL	N	20	S	TTT - 885.08 TTE - 511.74 J - 1.92	TTT - 751.72 TTE - 275.19 J - 31.81	TTT - 751.74 TTE - 262.56 J - 33.39	TTT - 885.08 TTE - 511.74 J - 1.92	<5
SL	N	100	S	TTT - 750.95 TTE - 303.95 J - 7.78	TTT - 696.11 TTE - 218.83 J - 6.78	TTT - 696.11 TTE - 218.83 J - 6.78	TTT - 790.11 TTE - 309.06 J - 2.13	<5
2h	N	0	S	TTT - 909.08 TTE - 547.74 J - 1.87	TTT - 763.72 TTE - 287.19 J - 33.59	TTT - 763.72 TTE - 287.19 J - 33.59	TTT - 909.08 TTE - 547.74 J - 1.87	<5
2h	N	20	S	TTT - 885.08 TTE - 511.74 J - 1.92	TTT - 751.72 TTE - 275.19 J - 31.81	TTT - 752.00 TTE - 262.56 J - 30.16	TTT - 885.08 TTE - 511.74 J - 1.92	<5
2h	N	100	N (13 / 16)	-	-	-	-	<5
30m	N	0	N (3 / 12)	-	-	-	-	<5
30m	N	20	N (3 / 12)	-	-	-	-	<5
30m	N	100	N (8 / 14)	-	-	-	-	<5

seguida.

Resultados

Tal como na secção anterior, primeiros começámos por realizar um teste (Tabela 4.7), onde as tarefas de cada tipo são distribuídas de maneira igual (25% para cada tipo). No teste seguinte (Tabela 4.8) foi usado um dos casos da Tabela 4.7 (Linha 3), sendo que se variou o número de dias (1, 7 e 30), não tendo sido alterado o "conteúdo" dos mesmos. Portanto, todos os dias tinham exatamente as condições de planeamento.

Desde a Tabela 4.9 à Tabela 4.12, pudemos observar os resultados para casos de média complexidade, onde foi variado a frequência de cada tipo de tarefa. Nestes testes decidimos não excluir totalmente um dos tipos de tarefa, mas sim dar maior relevância a cada uma delas (55%) e distribuir uniformemente o resto pelos outros 3 tipos de tarefas (15% para cada um).

Tabela 4.6: Tamanho Reduzido - Teste 5

TN1 (33.33%) | TN2 (33.33%) | TTS (33.33%) | TTC (0%)								
MTE	CC	AJT	Sol	Sol-1	Sol-TTT	Sol-TTE	Sol-J	T(s)
SL	N	0	S	TTT - 931.83 TTE - 609.73 J - 6.36	TTT - 764.13 TTE - 309.33 J - 45.35	TTT - 764.13 TTE - 309.33 J - 45.35	TTT - 931.83 TTE - 609.73 J - 6.36	<5
SL	N	20	S	TTT - 907.83 TTE - 573.73 J - 6.52	TTT - 752.13 TTE - 297.33 J - 43.99	TTT - 752.13 TTE - 297.33 J - 43.99	TTT - 907.83 TTE - 573.73 J - 6.52	<5
SL	N	100	S	TTT - 811.83 TTE - 425.09 J - 7.27	TTT - 696.11 TTE - 194.04 J - 21.87	TTT - 696.11 TTE - 194.04 J - 21.87	TTT - 811.83 TTE - 425.09 J - 7.27	<5
2h	N	0	S	TTT - 931.83 TTE - 609.73 J - 6.36	TTT - 931.83 TTE - 609.73 J - 6.36	TTT - 931.83 TTE - 609.73 J - 6.36	TTT - 931.83 TTE - 609.73 J - 6.36	<5
2h	N	20	S	TTT - 907.83 TTE - 573.73 J - 6.52	TTT - 743.53 TTE - 328.33 J - 42.98	TTT - 752.13 TTE - 262.56 J - 43.99	TTT - 907.83 TTE - 573.73 J - 6.52	<5
2h	N	100	N (13 / 16)	-	-	-	-	<5
30m	N	0	N (3 / 12)	-	-	-	-	<5
30m	N	20	N (3 / 12)	-	-	-	-	<5
30m	N	100	N (7 / 13)	-	-	-	-	<5

Análise

Tendo em conta os resultados obtidos na Tabela 4.7, decidimos testar a execução do algoritmo para diferentes dias, de modo a confirmar que o tempo de execução seria aumentado de forma proporcional. Ou seja, como no algoritmo uma extensão dos dias de planeamento é tratado de forma iterativa, seria expectável que o tempo de execução do algoritmo crescesse proporcionalmente ao número de dias, ou seja, se para um dia de planeamento o tempo de execução rondasse um minuto, para sete dias de planeamento rondaria os sete minutos, e por aí adiante. Tal como já foi referido, o objectivo aqui passava por confirmar se o tempo de execução seria o expectável. O que se veio a confirmar.

Em relação aos testes seguintes (Tabela 4.9 à 4.12). As principais conclusões são as seguintes:

- Uma frequência mais elevada de tarefas do tipo 1 (Tabela 4.9) leva a um maior tempo de execução do algoritmo;

Tabela 4.7: Tamanho Médio - Teste 1

TN1 (25%) | TN2 (25%) | TTS (25%) | TTC (25%)								
MTE	CC	AJT	Sol	Sol-1	Sol-TTT	Sol-TTE	Sol-J	T(m)
SL	N	0	N (25 / 31)	-	-	-	-	<0.1
SL	N	20	N (25 / 31)	-	-	-	-	<0.1
SL	N	100	S	TTT - 1838.28 TTE - 787.83 J - 30.10	TTT - 1758.97 TTE - 670.33 J - 31.20	TTT - 1758.97 TTE - 663.12 J - 25.41	TTT - 1878.58 TTE - 848.04 J - 22.29	1.56
SL	S	0	N (23 / 30)	-	-	-	-	<0.1
SL	S	20	N (23 / 30)	-	-	-	-	<0.1
SL	S	100	S	TTT - 1989.05 TTE - 966.66 J - 18.53	TTT - 1815.21 TTE - 805.12 J - 56.07	TTT - 1863.50 TTE - 782.91 J - 7.96	TTT - 1863.50 TTE - 782.91 J - 7.96	1.87
2h	N	0	N (23 / 31)	-	-	-	-	<0.1
2h	N	20	N (23 / 31)	-	-	-	-	<0.1
2h	N	100	N (25 / 31)	-	-	-	-	<0.1
2h	S	0	N (26 / 31)	-	-	-	-	<0.1
2h	S	20	N (26 / 31)	-	-	-	-	<0.1
2h	S	100	N (27 / 31)	-	-	-	-	<0.1
30m	N	0	N (8 / 27)	-	-	-	-	<0.1
30m	N	20	N (8 / 27)	-	-	-	-	<0.1
30m	N	100	N (14 / 28)	-	-	-	-	<0.1
30m	S	0	N (8 / 27)	-	-	-	-	<0.1
30m	S	20	N (8 / 27)	-	-	-	-	<0.1
30m	S	100	N (14 / 28)	-	-	-	-	<0.1

Tabela 4.8: Tamanho Médio - Teste 2

TN1 (25%) | TN2 (25%) | TTS (25%) | TTC (25%)					
D	MTE	CC	AJT	Sol	T(m)
1	SL	N	100	S	1.87
7	SL	N	100	S	7.19
30	SL	N	100	S	34.43

- A média do tempo total de espera piora para casos de tarefas normais, com dois auxiliares (Tabela 4.10);
- A justiça das rotas dos auxiliares decresce nos casos onde há maior relevância de tarefas de transporte sem prioridade (Tabela 4.11);
- O algoritmo teve dificuldades em encontrar soluções para os casos onde há maior relevância de tarefas de transporte, com prioridade (Tabela 4.12);
- A obrigatoriedade de cada paciente ser visitado pela mesma equipa pode levar a que o algoritmo não seja concluído, ou que exista um aumento no tempo total de trabalho a efectuar.

4.3.2 50 Tarefas / 10 Trabalhadores

Resultados

Nesta subsecção decidimos voltar a aumentar a dimensão do problema, mas desta vez o foco do teste vai para outros parâmetros ainda não testados.

Até agora, todos os testes tinham tido uma distribuição "justa" entre a manhã e a tarde, sendo que todos os auxiliares usavam carros como meio de transporte. Aqui, manhã e tarde não tem um significado literal, trata-se apenas de uma designação para a divisão de dois períodos relativamente ao horário total a considerar na solução. Na Tabela 4.13 pudemos observar o comportamento do algoritmo, variando a frequência de tarefas entre os dois períodos. A Tabela 4.14 contém a resposta do algoritmo, alterando o meio de transporte. Para este caso específico, a duração das tarefas foi alterada para 100 minutos.

Análise

Ao variar a frequência das tarefas nestes dois períodos (Tabela 4.13), podemos constatar o óbvio: houve um aumento significativo da (in)justiça de trabalho entre trabalhadores: 27.09% para 95.10%. Isto significa que no segundo caso, o trabalhador com maior

Tabela 4.9: Tamanho Médio - Teste 3

TN1 (55%) | TN2 (15%) | TTS (15%) | TTC (15%)								
MTE	CC	AJT	Sol	Sol-1	Sol-TTT	Sol-TTE	Sol-J	T(m)
SL	N	0	S	TTT - 2126.411 TTE - 850.71 J - 6.07	TTT - 1948.22 TTE - 672.13 J - 49.21	TTT - 1948.22 TTE - 672.13 J - 49.21	TTT - 2124.18 TTE - 850.71 J - 5.72	5.5
SL	N	20	S	TTT - 2078.41 TTE - 802.71 J - 6.20	TTT - 1912.22 TTE - 636.13 J - 48.14	TTT - 1912.22 TTE - 636.13 J - 48.14	TTT - 2076.18 TTE - 802.71 J - 5.85	5.95
SL	N	100	S	TTT - 1886.41 TTE - 660.69 J - 6.82	TTT - 1833.40 TTE - 609.02 J - 30.86	TTT - 1834.43 TTE - 574.49 J - 30.83	TTT - 1884.18 TTE - 660.69 J - 6.43	6.99
SL	S	0	N (29 / 30)	-	-	-	-	<0.1
SL	S	20	N (29 / 30)	-	-	-	-	<0.1
SL	S	100	S	TTT - 1920.66 TTE - 787.62 J - 1.64	TTT - 1802.63 TTE - 695.81 J - 26.45	TTT - 1802.63 TTE - 695.81 J - 26.45	TTT - 1919.63 TTE - 826.78 J - 1.54	8.1
2h	N	0	S	TTT - 2126.41 TTE - 850.71 J - 6.07	TTT - 2038.22 TTE - 757.02 J - 33.72	TTT - 2038.22 TTE - 757.02 J - 33.72	TTT - 2124.18 TTE - 850.71 J - 5.72	6.5
2h	N	20	S	TTT - 2078.41 TTE - 802.71 J - 6.20	TTT - 1990.22 TTE - 711.56 J - 34.43	TTT - 1990.22 TTE - 711.56 J - 34.43	TTT - 1838.28 TTE - 787.83 J - 30.10	5.17
2h	N	100	N (29 / 30)	-	-	-	-	<0.1
2h	S	0	N (29 / 30)	-	-	-	-	<0.1
2h	S	20	N (29 / 30)	-	-	-	-	<0.1
2h	S	100	N (26 / 30)	-	-	-	-	<0.1
30m	N	0	N (10 / 27)	-	-	-	-	<0.1
30m	N	20	N (10 / 27)	-	-	-	-	<0.1
30m	N	100	N (13 / 27)	-	-	-	-	<0.1
30m	S	0	N (10 / 27)	-	-	-	-	<0.1
30m	S	20	N (10 / 27)	-	-	-	-	<0.1
30m	S	100	N (13 / 27)	-	-	-	-	<0.1

Tabela 4.10: Tamanho Médio - Teste 4

TN1 (15%) | TN2 (55%) | TTS (15%) | TTC (15%)								
MTE	CC	AJT	Sol	Sol-1	Sol-TTT	Sol-TTE	Sol-J	T(m)
SL	N	0	S	TTT - 2109.00 TTE - 1086.41 J - 6.92	TTT - 2109.00 TTE - 1086.41 J - 6.92	TTT - 2109.00 TTE - 1086.41 J - 6.92	TTT - 2109.00 TTE - 1086.41 J - 6.92	0.68
SL	N	20	S	TTT - 2097.06 TTE - 1091.29 J - 6.96	TTT - 1694.33 TTE - 681.22 J - 45.58	TTT - 1694.33 TTE - 681.22 J - 45.58	TTT - 2109.00 TTE - 1086.41 J - 6.92	0.56
SL	N	100	S	TTT - 1857.06 TTE - 851.17 J - 7.83	TTT - 1747.14 TTE - 769.72 J - 3.91	TTT - 1747.14 TTE - 769.72 J - 3.91	TTT - 1747.14 TTE - 769.72 J - 3.91	0.71
SL	S	0	S	TTT - 2149.98 TTE - 1263.12 J - 26.41	TTT - 2011.57 TTE - 1194.67 J - 35.18	TTT - 2073.23 TTE - 1103.93 J - 21.43	TTT - 2184.75 TTE - 1325.66 J - 20.70	0.83
SL	S	20	S	TTT - 2101.98 TTE - 1215.12 J - 26.94	TTT - 1975.57 TTE - 1158.67 J - 33.89	TTT - 2025.23 TTE - 1055.93 J - 21.88	TTT - 2136.75 TTE - 1277.66 J - 21.11	0.9
SL	N	100	N (29 / 30)	-	-	-	-	<0.1
2h	N	0	N (29 / 30)	-	-	-	-	<0.1
2h	N	20	N (29 / 30)	-	-	-	-	<0.1
2h	N	100	N (29 / 30)	-	-	-	-	<0.1
2h	N	0	N (25 / 30)	-	-	-	-	<0.1
2h	N	20	N (25 / 30)	-	-	-	-	<0.1
2h	N	100	N (27 / 30)	-	-	-	-	<0.1
30m	N	0	N (7 / 26)	-	-	-	-	<0.1
30m	N	20	N (7 / 26)	-	-	-	-	<0.1
30m	N	100	N (13 / 27)	-	-	-	-	<0.1
30m	N	0	N (7 / 26)	-	-	-	-	<0.1
30m	N	20	N (7 / 26)	-	-	-	-	<0.1
30m	N	100	N (13 / 27)	-	-	-	-	<0.1

Tabela 4.11: Tamanho Médio - Teste 5

TN1 (15%) | TN2 (15%) | TTS (55%) | TTC (15%)								
MTE	CC	AJT	Sol	Sol-1	Sol-TTT	Sol-TTE	Sol-J	T(m)
SL	N	0	S	TTT - 1928.90 TTE - 985.89 J - 10.30	TTT - 1753.89 TTE - 1273.13 J - 46.05	TTT - 1875.19 TTE - 856.41 J - 23.44	TTT - 1948.93 TTE - 1012.84 J - 8.34	1.47
SL	N	20	S	TTT - 1904.90 TTE - 961.89 J - 8.14	TTT - 1779.18 TTE - 812.85 J - 48.44	TTT - 1779.18 TTE - 812.85 J - 48.44	TTT - 1924.93 TTE - 987.35 J - 6.12	1.4
SL	N	100	S	TTT - 1808.90 TTE - 865.60 J - 2.48	TTT - 1674.06 TTE - 728.19 J - 43.47	TTT - 1674.06 TTE - 728.19 J - 43.47	TTT - 1808.90 TTE - 865.60 J - 2.48	2.51
SL	S	0	S	TTT - 1928.90 TTE - 1030.24 J - 31.87	TTT - 1753.89 TTE - 836.89 J - 68.55	TTT - 1753.89 TTE - 836.89 J - 68.55	TTT - 1874.58 TTE - 977.55 J - 28.52	2.03
SL	S	20	S	TTT - 1904.90 TTE - 1006.24 J - 30.42	TTT - 1741.89 TTE - 826.39 J - 65.74	TTT - 1741.89 TTE - 826.39 J - 65.74	TTT - 1850.58 TTE - 953.55 J - 26.91	2.12
SL	N	100	S	TTT - 1688.71 TTE - 709.31 J - 46.35	TTT - 1612.53 TTE - 633.89 J - 47.29	TTT - 1612.53 TTE - 633.89 J - 47.29	TTT - 1808.24 TTE - 800.52 J - 23.90	1.62
2h	N	0	N (31 / 32)	-	-	-	-	<0.1
2h	N	20	N (31 / 32)	-	-	-	-	<0.1
2h	N	100	N (31 / 32)	-	-	-	-	<0.1
2h	N	0	N (30 / 32)	-	-	-	-	<0.1
2h	N	20	N (30 / 32)	-	-	-	-	<0.1
2h	N	100	N (30 / 32)	-	-	-	-	<0.1
30m	N	0	N (14 / 28)	-	-	-	-	<0.1
30m	N	20	N (14 / 28)	-	-	-	-	<0.1
30m	N	100	N (14 / 28)	-	-	-	-	<0.1
30m	N	0	N (14 / 28)	-	-	-	-	<0.1
30m	N	20	N (14 / 28)	-	-	-	-	<0.1
30m	N	100	N (14 / 28)	-	-	-	-	<0.1

Tabela 4.12: Tamanho Médio - Teste 6

TN1 (15%) | TN2 (15%) | TTS (15%) | TTC (55%)								
MTE	CC	AJT	Sol	Sol-1	Sol-TTT	Sol-TTE	Sol-J	T(m)
SL	N	0	N (38 / 39)	-	-	-	-	<0.1
SL	N	20	N (38 / 39)	-	-	-	-	<0.1
SL	N	100	N (38 / 39)	-	-	-	-	<0.1
SL	S	0	N (38 / 39)	-	-	-	-	<0.1
SL	S	20	N (38 / 39)	-	-	-	-	<0.1
SL	N	100	N (38 / 39)	-	-	-	-	<0.1
2h	N	0	N (38 / 39)	-	-	-	-	<0.1
2h	N	20	N (38 / 39)	-	-	-	-	<0.1
2h	N	100	N (37 / 39)	-	-	-	-	<0.1
2h	N	0	N (34 / 39)	-	-	-	-	<0.1
2h	N	20	N (34 / 38)	-	-	-	-	<0.1
2h	N	100	N (33 / 39)	-	-	-	-	<0.1
30m	N	0	N (11 / 30)	-	-	-	-	<0.1
30m	N	20	N (14 / 31)	-	-	-	-	<0.1
30m	N	100	N (18 / 28)	-	-	-	-	<0.1
30m	N	0	N (14 / 28)	-	-	-	-	<0.1
30m	N	20	N (18 / 28)	-	-	-	-	<0.1
30m	N	100	N (14 / 28)	-	-	-	-	<0.1

Tabela 4.13: Tamanho Médio - Teste 7

TN1 (15%) | TN2 (15%) | TTS (55%) | TTC (15%)									
M | T	MTE	CC	AJT	Sol	Sol-1	Sol-TTT	Sol-TTE	Sol-J	T(m)
50 | 50	SL	N	0	S	TTT - 4975.16 TTE - 3919.69 J - 27.09	TTT - 4796.24 TTE - 3728.31 J - 44.90	TTT - 4891.36 TTE - 3826.20 J - 27.32	TTT - 4975.16 TTE - 3919.69 J - 27.09	TO
66.66 | 33.33	SL	N	20	S	TTT - 3268.12 TTE - 2334.29 J - 95.10	TTT - 3268.12 TTE - 2334.29 J - 95.10	TTT - 3268.12 TTE - 2334.29 J - 95.10	TTT - 3268.12 TTE - 2334.29 J - 95.10	TO

Tabela 4.14: Tamanho Médio - Teste 8

TN1 (80%) | TN2 (20%) | TTS (0%) | TTC (0%)									
TR	MTE	CC	AJT	Sol	Sol-1	Sol-TTT	Sol-TTE	Sol-J	T(m)
C	SL	N	20	S	TTT - 6488.75 TTE - 513.49 J - 15.51	TTT - 6482.99 TTE - 511.89 J - 15.36	TTT - 6491.51 TTE - 507.45 J - 15.57	TTT - 6569.78 TTE - 595.41 J - 3.30	TO
P	SL	N	20	N (50 / 51)	-	-	-	-	<0.1

carga de trabalho atribuída iria ter quase o dobro de tempo de trabalho relativamente ao trabalhador com menos tempo de trabalho.

No outro teste (Tabela 4.14) verificou-se que quando os auxiliares se deslocam a pé, o algoritmo pode não ser concluído, ou poderia existir um aumento do tempo total de trabalho, relativamente ao uso de carro (carrinha) como meio de transporte. Neste caso específico ficou apenas uma tarefa por realizar, o que é suficiente para a interrupção do algoritmo.

Para casos desta complexidade, já foi atingido o tempo limite (60 minutos) pelo que o algoritmo acabou por ser interrompido. Mesmo sendo interrompido, são devolvidas as soluções geradas pela procura extensiva dentro do tempo limite. Todo este tempo se deve a um único dia, o que nos leva a pensar que futuramente poderá existir uma melhoria do algoritmo, visto que este tem dificuldades em certas situações. Neste caso específico, a solução actual só iria permitir optimizar um único dia. Ou seja, para casos desta complexidade, com por exemplo, 30 dias, aquilo que iria acontecer é que o algoritmo iria encontrar soluções para os 30 dias numa questão de segundos. Após esta fase, iria ser começada a optimização no primeiro dia, onde poderiam haver melhorias, mas o algoritmo não iria terminar a procura extensiva, o que levaria a que os restantes 29 dias não tivessem qualquer optimização.

4.4 Dimensão Elevada

Resultados

Para finalizar este capítulo realizámos um último teste, em condições "leves", de modo a comprovar que a nossa solução está habilitada a resolver problemas de larga escala. Todas as tarefas consideradas foram do tipo 1, portanto cada tarefa seria realizada por um trabalhador; cada paciente iria ser visitado duas vezes por dia: uma de manhã, outra de tarde, sendo que o poderia ser por qualquer trabalhador. Foram considerados 30 dias, sendo que as tarefas seriam exactamente as mesmas para cada dia. Cada tarefa

teria a duração de 5 minutos, com uma janela temporal de 2 horas; foi ainda definido que não existiria limite de tempo de espera entre a realização de duas tarefas; não foram considerados almoços; considerou-se um alargamento das janelas temporais de 20 %; por cada 5 tarefas, foi considerado 1 auxiliar (200 tarefas para 40 auxiliares no teste efectuado, sendo que cada paciente estava associado a 2 tarefas, pelo que foram considerados 100 pacientes). Não foi realizada a procura exaustiva, visto que o objectivo aqui passava apenas por perceber quanto tempo demoraria o algoritmo a encontrar uma solução admissível para um problema de grande dimensão.

Tabela 4.15: Tamanho Elevado - Teste 1

TN1 (100%) | TN2 (0%) | TTS (0%) | TTC (0%)									
D	MTE	CC	AJT	Sol	Sol-1	Sol-TTT	Sol-TTE	Sol-J	T(m)
30	SL	N	20	S	TTT - 18827.23 TTE - 16861.67 J - 2.99	-	-	-	2.34

Análise

Na Tabela 4.15 podemos então concluir que, utilizando *benchmarks* relativamente simples, a nossa solução não tem problemas em resolver casos de larga escala, tendo encontrado uma solução para 200 tarefas diárias, considerando 40 trabalhadores e 30 dias, em 2.34 minutos. Os dados apresentados na coluna relativa à primeira solução referem-se a um único dia. Os restantes 29 dias são iguais, porque como já foi referido, as tarefas de cada dia eram exactamente iguais.

4.5 Conclusões Finais

A principal conclusão retirada é que performance e o tempo de execução do algoritmo estará sempre mais dependente da complexidade envolvida do problema a resolver, a nível de restrições ou variedade de tipos de tarefas, e não tanto em termos de dimensão da instância.

Em relação à variação de parâmetros, em cada teste, os resultados foram os esperados: um aumento das janelas temporais, permite em geral obter melhores soluções. Limitar o tempo máximo de espera entre tarefas, pode levar à não conclusão das mesmas. Exigir 30 minutos de tempo máximo de espera entre tarefas leva à não conclusão do algoritmo. Não considerámos particularmente preocupante, pois deve-se sobretudo às condições específicas dos casos de estudo, e a maioria das tarefas que ficaram por atribuir devem-se a tarefas destinadas ao período de almoço.

Capítulo 5

Plataforma Web

Neste capítulo apresentamos a aplicação web, que desenvolvemos com o intuito de facilitar a utilização do algoritmo desenhado. Todo o trabalho desenvolvido encontra-se disponível no repositório `https://bitbucket.org/ordep94c/thesis2019_pedrocolaco/src/master/`. A estrutura da aplicação segue o modelo cliente-servidor, onde o cliente é executado no browser, sendo que existe um servidor que suporta os serviços e os recursos a partilhar com os clientes. O capítulo está, então, dividido em três secções: IFML (*URL - IFML*), Cliente e Servidor.

Foi implementada uma arquitectura MVC através da framework Java Spring (*URL - Java Spring*). MVC (Model-View-Controller ou, em português, Modelo-Vista-Controlador) é uma arquitectura que permite dividir o desenvolvimento de uma aplicação em três componentes interconectadas. É possível, então, desenvolver paralelamente estas três partes. A Vista, que é a representação da saída de dados e das respostas que são dadas pelo sistema está exposta na secção relativa ao Cliente; enquanto o Modelo e o Controlador estão na secção do Servidor. O Modelo representa os dados da aplicação e as suas operações; o Controlador é a componente da lógica de recepção e envio de pedidos entre a Vista e o Modelo.

5.1 IFML

IFML (*Interaction Flow Modeling Language*) é uma linguagem de modelação, que permite perceber toda a interação possível que um utilizador poderá efetuar numa aplicação,

independentemente do tipo de dispositivo ostentado, ou da tecnologia de desenvolvimento usada.

Tendo isto em conta, foi considerado fundamental produzir um esquema IFML, antes de se passar ao desenvolvimento da plataforma, de maneira a que o desenvolvimento fosse, no fundo, a consequência de um plano pré-existente. O esquema apresenta-se na Figura 5.1.

Segue-se a lista das principais interacções com a aplicação:

1. Login / Registo - A página web inicial contém uma mensagem de boas-vindas, e uma pequena descrição da plataforma. O utilizador pode agora registar-se ou efectuar login, no caso de já estar registado. Para se registar no sistema, basta ao utilizador introduzir um nome de utilizador não utilizado, para além de ter de introduzir a password a usar uma segunda vez, de maneira a confirmar. Se os valores inseridos nos dois campos não coincidirem, é mostrada uma mensagem de erro, e o utilizador terá de voltar a tentar. Posteriormente, se o login for efectuado com sucesso, o utilizador entrará automaticamente na sua página pessoal, onde está contida a sua informação e uma lista de planos existentes.

2. Adicionar Plano - O utilizador pode agora adicionar quantos planos quiser à sua conta. Nesta fase inicial será pedido um nome do plano; escolher entre "Sim" ou "Não" dependendo irão ser considerados almoços para os auxiliares no planeamento; o número de dias do horizonte de planeamento; e para finalizar um modo de transporte: a pé (1km percorrido em 10 minutos), de carro (1km percorrido em 2 minutos). No caso da opção escolhida ser "De carro", o utilizador introduzirá o número de lugares disponíveis para cada veículo. Se todas as informações forem preenchidas, um novo plano será então adicionado ao sistema, e o utilizador poderá prosseguir para página do plano gerado;

3. Editar / Remover Plano - Na página referente ao plano, está exposta a informação que o utilizador introduziu na criação do mesmo. Aqui, o utilizador terá a oportunidade de editar algum dos parâmetros (nome, transporte, dias, almoços). Existe também a possibilidade de remover o plano do sistema;

4. Adicionar Auxiliares e Pacientes - Nesta página estão também contidas as listas de auxiliares e de pacientes associadas ao plano. O utilizador poderá adicionar o número de auxiliares e de pacientes que pretender. Para cada auxiliar, terá de introduzir obrigatoriamente um nome, e opcionalmente um contacto telefónico, um endereço de e-mail, uma fotografia, e ainda, qualquer informação adicional

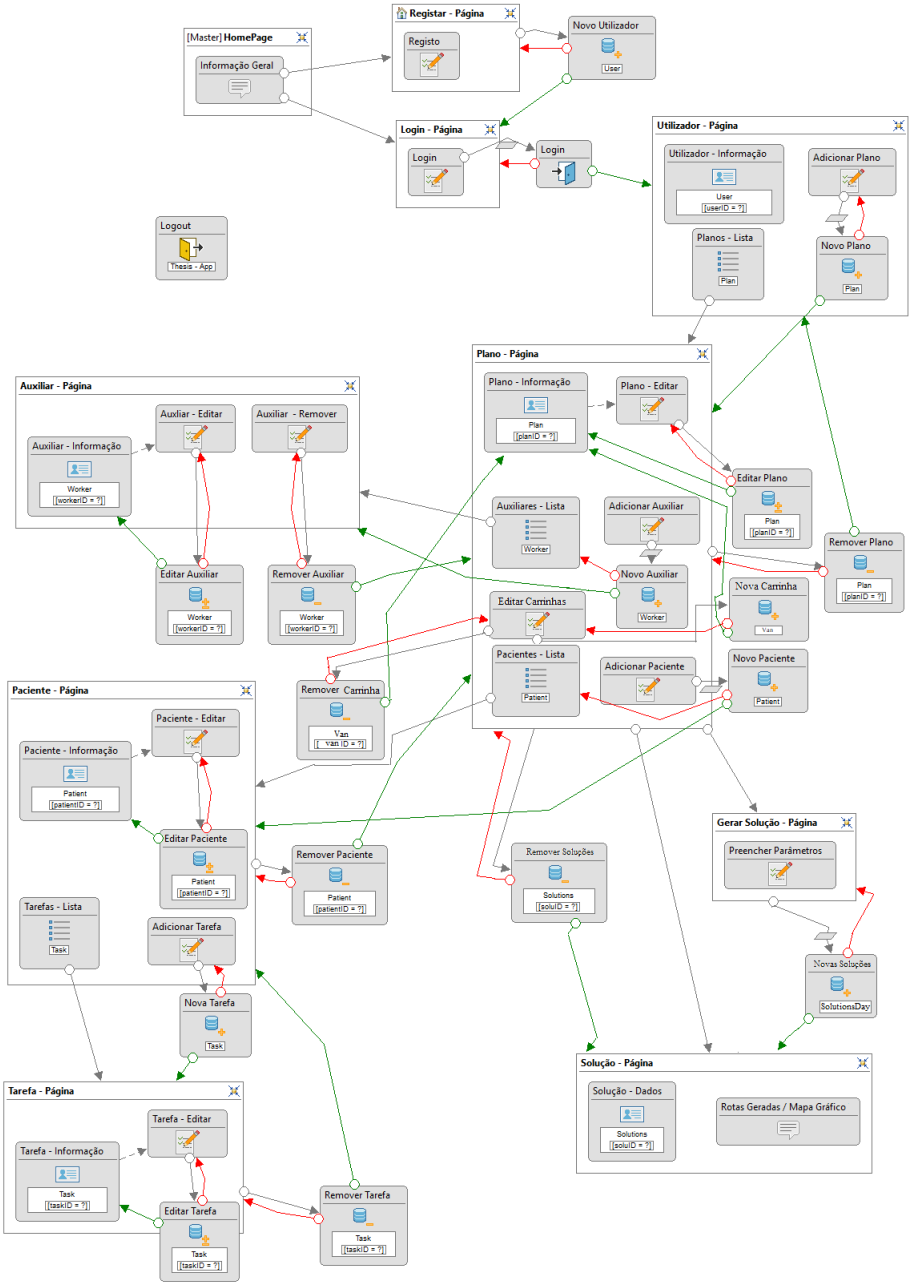

Figura 5.1: Esquema IFML

que considerar relevante. Já no caso de um paciente, o utilizador terá de introduzir as suas coordenadas geográficas (latitude e longitude). As coordenadas não têm de ser introduzidas manualmente, visto que o utilizador tem a possibilidade de as gerar automaticamente, ao premir um ponto do mapa. Estas serão essenciais à produção do planeamento das rotas;

5. Editar Carrinhas - Aqui o utilizador tem a oportunidade de alterar as carrinhas associadas ao plano. Introduzirá um novo número de carrinhas, sendo que para cada uma destas irá também introduzir o número de lugares disponíveis;

6. Adicionar Tarefas - As tarefas ficarão diretamente ligadas a um paciente. Para adicionar uma nova tarefa terá de ser seleccionado um conjunto de dados:

    - Seleccão do número de auxiliares necessários - um ou dois auxiliares;
    - Seleccão do tipo de tarefa - tarefa normal, tarefa de transporte para o centro sem prioridade, transporte para o centro com prioridade; transporte para casa sem prioridade ou transporte para casa com prioridade;
    - Seleccão do ínicio da janela temporal - 08:00, 08:30, 09:00, (...) 20:00;
    - Seleccão do fim da janela temporal - 08:00, 08:30, 09:00, (...) 20:00;
    - Seleccão da duração prevista da tarefa - 5 minutos, 10 minutos, (...) 120 minutos;
    - Seleccão da possibilidade da janela temporal poder ser alargada? - sim ou não;
    - Seleccão do dia no qual a tarefa será realizada - seleccionar algum dos dias do plano;
    - Introdução de informação adicional - qualquer informação extra que o utilizador considerar relevante.

7. Editar / Remover Auxiliares, Pacientes ou Tarefas - Nas páginas relativas a um Auxiliar, Paciente, ou Tarefa a ideia geral é a mesma: existe a possibilidade de ver, editar e remover a informação existente;

8. Ver / Descartar / Gerar Solução - Depois de introduzida toda a informação referente aos auxiliares, pacientes e tarefas a realizar, o utilizador pode agora gerar a solução. Se a solução já tiver sido gerada, o utilizador pode acedê-la directamente e/ou removê-la. Na Figura 5.2 apresenta-se a página que precede a exibição da

solução. O utilizador irá preencher os últimos dados relativos ao plano a solucionar:

- Coordenadas geográficas do centro de onde partirão todos os auxiliares. As coordenadas podem ser geradas automaticamente, ao indicar um ponto no mapa;
- Trabalho diário máximo de cada auxiliar, em horas;
- Tempo máximo de espera entre tarefas, em minutos;
- Hora de almoço;
- Tempo limite que o programa irá ser executado;
- Seleccionar se um paciente será sempre visitado, ou não, pela mesma equipa;
- Seleccionar se as janelas temporais serão alargadas. Se sim, inserir o valor, em percentagem;
- Seleccionar se irá existir procura exaustiva.

O utilizador pode agora gerar a solução. Após a solução ser gerada, poderá escolher se quer visualizar a solução onde o tempo total das rotas é minimizado; a solução onde o tempo total de espera entre tarefas é minimizado ou a solução mais justa.

Depois, o utilizador poderá visualizar toda a informação referente à solução gerada, bem como, um mapa ilustrativo das rotas a serem efectuadas por cada auxiliar.

5.2 Cliente

Nesta secção vamos abordar a parte relativa ao cliente. Para desenvolver a interface da aplicação web foi utilizada a framework React (*URL - React*). Em React é possível criar interfaces interactivas, através de um encapsulamento de componentes, que gerem o próprio estado. Isto significa que uma única página web poderá ter diversas vistas, dependendo dos dados que forem alterados. No fundo, não é necessário ter de programar diferentes páginas, para diferentes vistas, visto que o React actualizará com eficiência e processará apenas os componentes certos quando existirem alterações de estado.

Esta técnica foi utilizada no desenvolvimento da interface diversas vezes. Na Figura 5.3 podemos ver um exemplo disso mesmo. Aqui, na página relativa à adição de um novo plano, a vista é alterada automaticamente ao ser premido algum dos botões existentes. O utilizador preenche os dados do novo plano a acrescentar à sua lista de planos, e

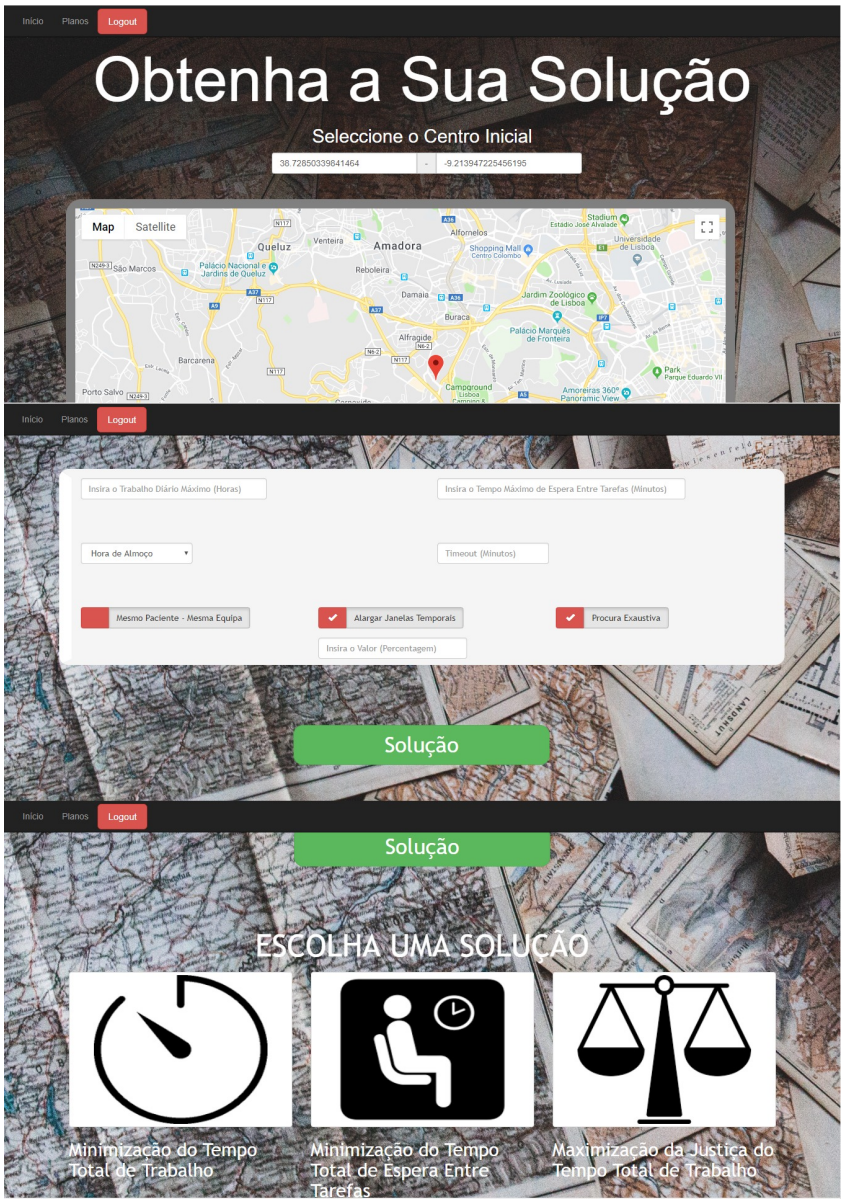

Figura 5.2: Página web referente à geração de uma solução

o conteúdo da página vai variando à medida que o utilizador prossegue ou retrocede no processo. No final, é visualizada uma mensagem, no caso do pedido ter sido bem efectuado.

A lógica dos componentes, em React, é desenvolvida em Javascript (*URL - Javascript*), sendo que cada componente tem um método *render()* que receberá um conjunto de dados de entrada, e retornará a vista a exibir, através de código HTML (*URL - HTML*).

Para além das tecnologias já referidas, foi ainda usado CSS (*URL - CSS*), de maneira a puder adicionar estilo, seja através de cores, tamanhos, ou espaçamentos às páginas web; e tornar a aplicação responsiva.

Uma técnica essencial, para tornar a aplicação responsiva para diferentes dispositivos, foi a utilização de *querys @media*, de modo a puder alterar o estilo para diferentes tipos de dispositivos. Na Listagem 5.1 está apresentado um pequeno excerto do ficheiro referente ao códido CSS, onde o espaçamento da página alusiva ao auxiliar é modelado de maneira diferente para quatro tipos de dispositivos: telemóvel (*mobile*), *tablet*, computador portátil (*laptop*), e ainda ecrãs de grandes dimensões (*desktop*).

Houve, portanto, o cuidado de tornar a plataforma o mais responsiva possível à variação do tamanho de ecrã usado. Na Figura 5.4 é possível observar o conteúdo da página referente a um auxiliar, com um diferente estilo para cada dispositivo. Neste caso específico, a maior diferença, no estilo, entre os diferentes dispositivos, é precisamente no telemóvel, onde a secção que contém a informação do auxiliar é exposta de uma maneira completamente diferente em relação aos restantes dispositivos: a fotografia aparece no topo; o botão das opções surge no final; mas mais importante, é a barra de navegação, que ao invés de ser exposta com as três opções (Início, Planos, Logout), de maneira adjacente, é exibida de maneira a não ocupar espaço desnecessário, sendo que o utilizador pode aceder, e visualizar as opções, no caso de ser necessário (Figura 5.5).

Foi também usado Bootsrap (*URL - Bootsrap*), que é uma *framework* de *front-end* precisamente vocacionada para componentes de interface, baseado em modelos de design. Esta framework tem um sistema de "grelhas" que permite dividir uma página web em linhas e colunas, de maneira a puder alinhar completamente o conteúdo da página, de maneira responsiva. Na Figura 5.6 apresenta-se um exemplo do seu uso. Na página alusiva a um plano, decidimos dividir o conteúdo da página em três linhas: a primeira é toda ela ocupada com as informações básicas referente ao plano; na segunda, dividimos em duas colunas, sendo que a primeira ocupa um espaço de 2/3, contendo as listas de auxiliares, pacientes e carrinhas, enquanto que na coluna que ocupa o restante espaço está a *tab*, na qual o utilizador poderá escolher se pretende visualizar a lista de auxiliares, a lista de pacientes, ou lista de carrinhas existentes; na última linha são exibidos

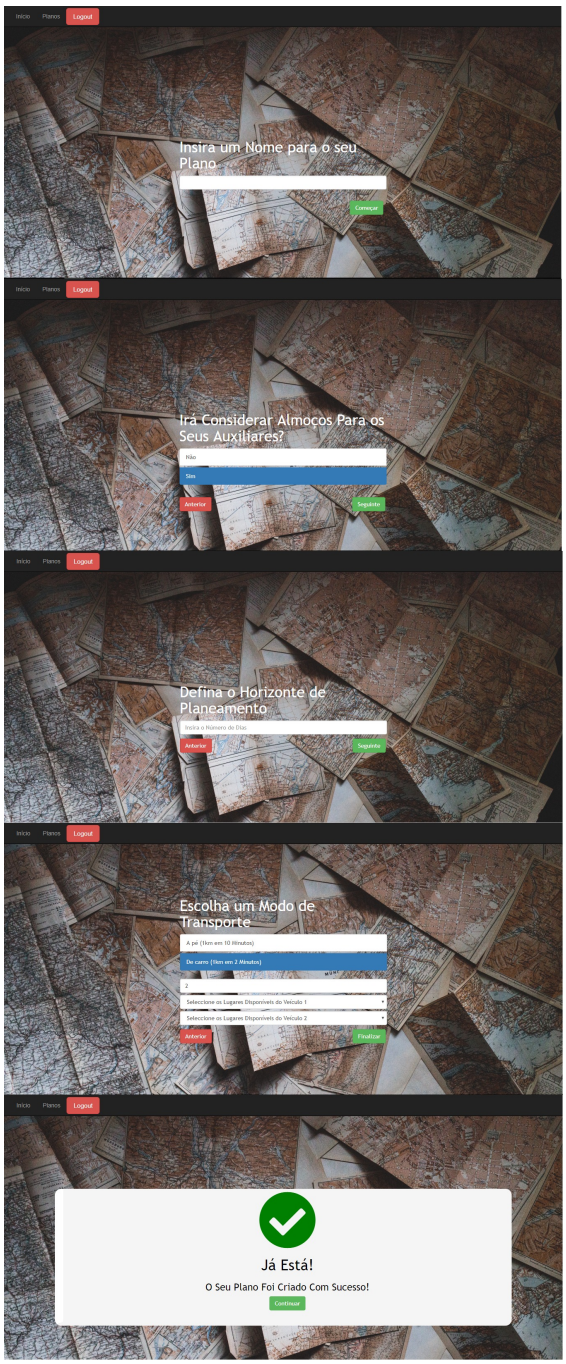

Figura 5.3: Diferentes vistas, relativa à página de adição de um novo plano

```
@media (max-width: 500px) and (max-height: 500px) { /*Mobile Size*/
  .workerPage{
  padding-top: 20%;
  padding-bottom: 20%;
  }
}

@media (max-width: 850px) and (min-height: 900px) { /*Tablet Size*/
  .workerPage{
  padding-top: 35%;
  padding-bottom: 50%;
  }
}

@media (min-width: 750px) and (max-height: 900px) { /*Laptop Size*/
  .workerPage{
  padding-top: 5%;
  padding-bottom: 20%;
  }
}

@media (min-width: 900px) and (min-height: 900px) { /*Desktop Size*/
  .workerPage{
  padding-top: 10%;
  padding-bottom: 15%;
  }
}
```

Listagem 5.1: Código CSS, com querys @media

dois botões, em colunas de igual tamanho, onde o utilizador poderá ver a solução (no caso de ela já existir), ou de gerar uma nova.

5.3 Servidor

Nesta secção será abordada a forma como foi desenvolvida a aplicação, naquilo que diz respeito ao lado do servidor. A secção subdivide-se em Controlador e Modelo.

5.3.1 Controlador

Os controladores, na nossa arquitectura, seguem os princípios REST (*URL - REST*), pelo que foi programada uma Restfull (*URL - Restfull Api*), de maneira a que os clientes a possam manipular e aceder, através da interface da aplicação, a informação guardada na base de dados.

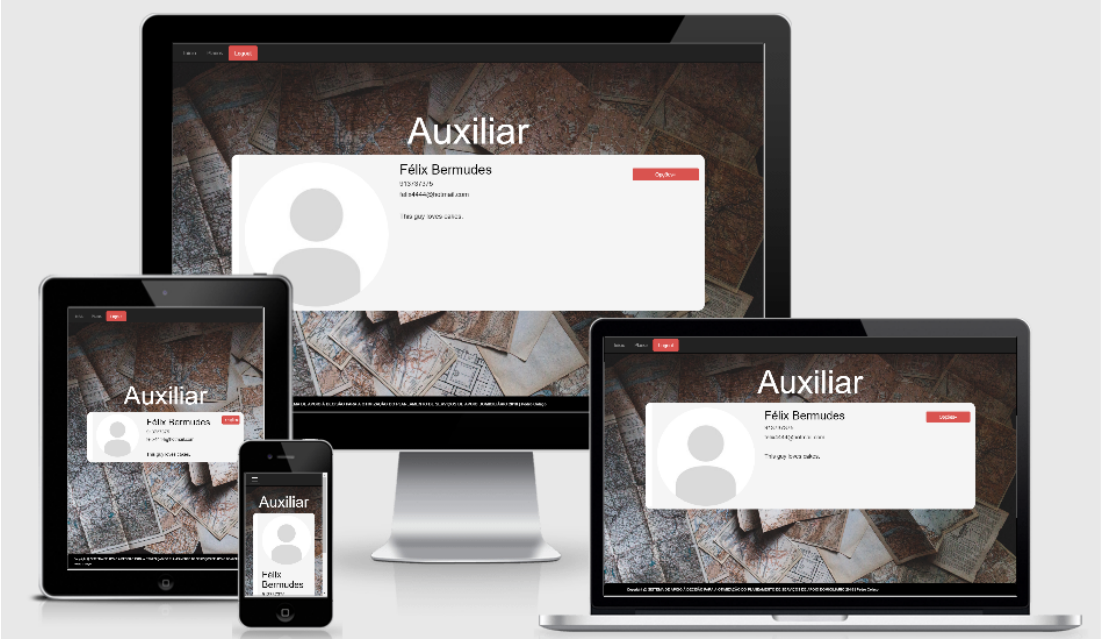

Figura 5.4: Responsividade da plataforma, em diferente dispositivos

Figura 5.5: Barra de navegação, telemóvel

Figura 5.6: Página web de um plano, onde foi usado *Bootsrap*

REST (Representational State Transfer ou, em português, Transferência de Estado Representacional) é um conjunto de princípios de arquitectura, que permitem a criação de um sistema distribuído onde se aproveita a infraestrutura web existente. Foi adoptado como o modelo a ser utilizado na evolução da arquitetura do protocolo HTTP (*URL - HTTP*). O HTTP é o principal protocolo de comunicação para sistemas Web, existente há mais de 20 anos.

Para tentar contextualizar a importância deste modelo no nosso trabalho, é importante referir que a nossa aplicação gere e coordena um conjunto vasto de informação, sejam utilizadores, planos, trabalhadores, tarefas, etc. Esta informação pode ser definida como recursos da aplicação, e um dos princípios REST é precisamente garantir que cada um desses dados é identificado como um recurso único, de maneira a que este possa ser manipulado individualmente. A identificação de cada recurso é feito através de um URI (Uniform Resource Identifier ou, em português, Identificador de Recursos Universal). URI une o Protocolo (http://) a localização do recurso (URL - tesePedro.com) e o nome do recurso (URN - /users/1/) de maneira a ser possível aceder a qualquer informação da aplicação.

Quando um cliente faz um pedido HTTP para o sistema, este terá de identificar qual o(s) recurso(s) que pretende aceder e/ou manipular através do seu URI. É ainda necessário que o cliente especifique o tipo de manipulação que deseja realizar. Este tipo de manipulação é identificado por métodos do protocolo HTTP. De seguida segue-se a lista dos principais métodos existentes:

- GET - Obter os dados de um recurso;
- POST - Adicionar um novo recurso;
- PUT - Actualizar os dados de um recurso;
- DELETE - Apagar um recurso.

Foi feito, então, no Spring um mapeamento de URNs com pedidos HTTP. Dando um exemplo prático, um cliente ao aceder à sua página pessoal na aplicação, irá fazer, automaticamente, um pedido HTTP GET para o URI "(...)/idUser/plans". A API recebe o pedido, e irá retornar a lista de planos desse utilizador, depois de confirmar que o mesmo existe na base de dados. A resposta ao cliente é dada em JSON (*URL - JSON*), que é um formato simples, de trocas de informações e dados entre sistemas. Agora, o cliente já tem acesso à lista dos seus planos, na página pessoal.

A Tabela 5.1 revela todos os controladores.

Tabela 5.1: Controladores do Servidor

Método HTTP	**URN**	**Descrição**
GET	/users	Devolve todos os utilizadores
GET	/users/{idUser}	Devolve o utilizador identificado pelo {idUser}
POST	/users	Cria um novo utilizador
GET	/users/auth/{userName}/{password}	Devolve o utilizador identificado pelo {userName} e {password}, no caso de este existir
GET	/users/{idUser}/plans	Devolve todos os planos associados ao utilizador identificado pelo {idUser}
POST	/users/{idUser}/plans	Cria um novo plano, associado ao utilizador identificado pelo {idUser}
GET	/users/{idUser}/plans/{idPlan}	Devolve o plano identificado pelo {idPlan}, do utilizador {idUser}
DELETE	/users/{idUser}/plans/{idPlan}	Remove o plano identificado pelo {idPlan}, do utilizador {idUser}
PUT	/users/{idUser}/plans/{idPlan}	Actualiza o plano identificado pelo {idPlan}, do utilizador {idUser}
POST	/users/{idUser}/plans/{idPlan}/ workers	Cria um novo auxiliar, associado ao plano {idPlan} do utilizador {idUser}

Table 5.1 continued from previous page

Método HTTP	URN	Descrição
GET	/users/{idUser}/plans/{idPlan}/ workers	Devolve todos os auxiliares, associados ao plano {idPlan} do utilizador {idUser}
GET	/users/{idUser}/plans/{idPlan}/ workers/{idWorker}	Devolve o auxiliar {idWorker}, associado ao plano {idPlan} do utilizador {idUser}
DELETE	/users/{idUser}/plans/{idPlan}/ workers/{idWorker}	Remove o auxiliar {idWorker}, associado ao plano {idPlan} do utilizador {idUser}
PUT	/users/{idUser}/plans/{idPlan}/ workers/{idWorker}	Actualiza o auxiliar {idWorker}, associado ao plano {idPlan} do utilizador {idUser}
POST	/users/{idUser}/plans/{idPlan}/ patients	Cria um novo paciente, associado ao plano {idPlan} do utilizador {idUser}
GET	/users/{idUser}/plans/{idPlan}/ patients	Devolve todos os pacientes, associados ao plano {idPlan} do utilizador {idUser}
GET	/users/{idUser}/plans/{idPlan}/ patients/{idPatient}	Devolve o paciente {idPatient}, associado ao plano {idPlan} do utilizador {idUser}
DELETE	/users/{idUser}/plans/{idPlan}/ patients/{idPatient}	Remove o paciente {idPatient}, associado ao plano {idPlan} do utilizador {idUser}
PUT	/users/{idUser}/plans/{idPlan}/ patients/{idPatient}	Actualiza o paciente {idPatient}, associado ao plano {idPlan} do utilizador idUser}
GET	/users/{idUser}/plans/{idPlan}/ vans	Devolve todas as carrinhas associadas ao plano {idPlan} do utilizador {idUser}
POST	/users/{idUser}/plans/{idPlan}/ vans	Adiciona carrinhas ao plano, removendo todas as que estavam antes associadas ao mesmo {idPlan} do utilizador {idUser}
GET	/users/{idUser}/plans/{idPlan}/ patients/{idPatient}/tasks	Devolve todas as tarefas do paciente {idPatient}, associado ao plano {idPlan} do utilizador {idUser}
GET	/users/{idUser}/plans/{idPlan}/ patients/{idPatient}/tasks/{idTask}	Devolve a tarefa {idTask} do paciente {idPatient}, associado ao plano {idPlan} do utilizador {idUser}
POST	/users/{idUser}/plans/{idPlan}/ patients/{idPatient}/tasks	Cria uma nova tarefa associada ao paciente {idPatient} do plano {idPlan} do utilizador {idUser}

Table 5.1 continued from previous page

Método HTTP	URN	Descrição
DELETE	/users/{idUser}/plans/{idPlan}/ patients/{idPatient}/tasks/{idTask}	Remove a tarefa {idTask} associada ao paciente {idPatient} do plano {idPlan} do utilizador {idUser}
PUT	/users/{idUser}/plans/{idPlan}/ patients/{idPatient}/tasks/{idTask}	Actualiza a tarefa {idTask} associada ao paciente {idPatient} do plano {idPlan} do utilizador {idUser}
PUT	/users/{idUser}/plans/{idPlan}/ generate	Gera a solução e actualiza o plano identificado pelo {idPlan}, do utilizador {idUser}
GET	/users/{idUser}/plans/{idPlan}/ solus	Devolve as soluções do plano {idPlan} associado ao utilizador {idUser}
DELETE	/users/{idUser}/plans/{idPlan}/ solus	Remove as soluções do plano {idPlan} associado ao utilizador {idUser}

5.3.2 Modelo

Os dados relativos à aplicação estão guardados em MySql (*URL - MySql*), um sistema de base de dados relacional da Oracle (*URL - Oracle*), que utiliza linguagem SQL. Estes dados são criados no Spring, num sistema orientado a objectos, pelo que existe a necessidade de conectar e aceder à base de dados, e de fazer um mapeamento entre os objectos Java e a base de dados relacional. A este mapeamento dá-se a designação de ORM (Object-Relational Mapping ou, em português, Mapeamento Objeto-Relacional (*URL - ORM*)). O Spring é capaz de integrar facilmente uma interface JPA (Java Persistance API (*URL - JPA*)), que é a interface que fornece um modelo de persistência para o ORM, enquanto que Hibernate (*URL - Hibernate*) é a sua implementação. Modelo de persistência, neste caso, significa um modelo onde os dados podem ser armazenados de maneira persistente, ou seja, os dados são retidos, mesmo após a energia do dispositivo de armazenamento ser desligada. Um armazenamento não volátil, portanto.

De maneira a criar e guardar os dados, as classes dos objectos Java são registadas como entidades, através da anotação @Entity, de modo a puderem ser criadas tabelas na base de dados. Para cada entidade será especificada uma chave primária, através da anotação @ID, enquanto que a anotação @Generated Value permite especificar que a gestão dos valores da chave primaria é efectuada pelo Hibernate, não sendo portanto responsabilidade do nosso código. De maneira a puder fazer o mapeamento de relacionamentos entre entidades, usámos anotações suportadas pelo JPA. O JPA contém as

associações One-to-one (Um para Um); One-to-Many (Um para Muitos); Many-to-one (Muitos para Um) e Many-to-Many (Muitos para Muitos).

Na Figura 5.7 estão dois exemplos de duas entidades, onde foi programado um mapeamento entre as mesmas. Neste caso, cada auxiliar tem um plano associado, sendo que cada plano contém uma lista de trabalhadores. Como é uma relação de muitos para um, será guardado na tabela do auxiliar um atributo extra, com a chave primária do plano a que está associado.

```java
@Entity
public class Plan {

    @Id
    @GeneratedValue
    private int planID;

    private String namePlan;
    private boolean transpCar;
    private int days;
    private boolean lunch;

    private int maxTime;
    private int maxWaitTimeBetweenTasks;
    private int lunchTime;
    private int lunchConsideredToTeamsStartingWorkingBeforeEqual;

    private boolean sameTeam_samePatient_constraint;
    private int tasksTWresize;
    private int extensiveSearch;

    @ManyToOne(cascade=CascadeType.MERGE)
    private User userPlan;

    @OneToMany(cascade=CascadeType.ALL, mappedBy = "workerPlan")
    @JsonIgnore
    List<Worker> workers;

    @OneToMany(cascade=CascadeType.ALL, mappedBy = "patientPlan")
    @JsonIgnore
    List<Patient> patients;

    @OneToMany(cascade=CascadeType.ALL, mappedBy = "planSolu")
    @JsonIgnore
    List<SolutionsDay> solus;

    public Plan(){

    }
```

(a) Plano

```java
@Entity
public class Worker {

    @Id
    @GeneratedValue
    private int IDworker;

    private String name;
    private int contact;
    private String email;
    private String info;

    @ManyToOne(cascade=CascadeType.MERGE)
    private Plan workerPlan;


    public Worker(){

    }
```

(b) Auxiliar

Figura 5.7: Objectos Plano e Auxiliar registados como Entidades

Depois das tabelas serem criadas, já é possível guardar informação. Na Listagem 5.2 é criado um objecto Utilizador, com um nome de usuário “eu” e uma password “a”. Este utilizador será guardado no repositório users, que é uma extensão do repositório JPA - CrudRepository (Listagem 5.3). CRUD é o acrónimo de Create, Read, Update e Delete, e representam as quatro operações básicas utilizadas em bases de dados relacionais. A interface CrudRepository espera dois tipos genéricos: o tipo de objecto que queremos manipular, e o tipo de atributo que representa a chave primária do objecto. No caso apresentado, os dois tipos são, respectivamente, User e Integer.

```
1 User u1 = new User("eu","a");
2
3 users.save(u1);
```

Listagem 5.2: Criação de um utilizador

```
1 public interface UserRepository extends CrudRepository<User,Integer>{
2
3   User findByuserName(String userName);
4
5 }
```

Listagem 5.3: Repositório de utilizadores

Capítulo 6

Conclusão

Tendo em conta as projecções relativas ao grande aumento do índice de envelhecimento nos próximos anos, vários desafios irão surgir consequentemente. Esta situação, leva à necessidade de existência de instituições capazes de servir estas populações.

O objectivo desta dissertação passava precisamente pelo desenvolvimento de uma plataforma web que integrasse uma solução, potencialmente inovadora, do problema de planeamento do serviço de apoio domiciliário, que pudesse ser usufruida por essas mesmas instituições, de maneira a facilitar o seu trabalho.

Consideramos que os objectivos foram cumpridos, pois tendo em conta os bons resultados obtidos, relativos aos estudos e testes efectuados ao algoritmo desenvolvido, pudemos apresentar uma solução que permite aos utilizadores interessados a possibilidade de variarem diversos parâmetros de entrada; consideraram diferentes tipos de tarefas; e, sobretudo, a oportunidade de escolher entre três diferentes tipos de soluções geradas.

A nível de performance, a principal conclusão retirada é que o tempo de execução do algoritmo estará sempre mais dependente da complexidade envolvida do problema a resolver, a nível de restrições ou variedade dos diferentes tipos de tarefas existentes, e não necessariamente da dimensão da instância do problema.

Como trabalho futuro, poderá ser equacionado a introdução de parâmetros e restrições adicionais, tais como considerar a possibilidade de num mesmo plano, puderem

existir auxiliares que só tenham disponibilidade para trabalhar em certos dias específicos; ou dividir, de maneira justa, o tempo limite de optimização da primeira solução encontrada por todos os dias do plano. A nível da plataforma web, poderão ser adicionados mecanismos que garantam uma maior segurança à aplicação.

Bibliografia

Bachouch, R., A. Guinet e S. Hajri-Gabouj (2011). "A decision-making tool for home health care nurses planning". Em: Supply Chain Forum: Int J 2011; 12(1):14-20.

Begur, S., D. Miller e J. Weaver (1997). "An integrated spatial DSS for scheduling and routing home-health-care nurses". Em: Interfaces 1997; 27(4):35–48.

Braekers, K., R. Hartl, S. Parragh e F. Tricoire (2016). "A bi-objective home care scheduling problem: analyzing the trade-off between costs and client inconvenience". Em: Eur J Oper Res; 248(2):428-43.

Decerle, J., O. Grunder, E. Hassani e O. Barakat (2018). "A memetic algorithm for a home health care routing and scheduling problem". Em: Operations Research for Health Care; 16:59-71.

Erdem, M. e S. Bulkan (2017). "A Two-Stage Solution Approach For The Large-Scale Home Healthcare Routing And Scheduling Problem". Em: The South African Journal of Industrial Engineering; 28(4):133-149.

Erdoğan, G. (2016). "An open source Spreadsheet Solver for Vehicle Routing Problems". Em: Computers Operations Research; 84:62-72.

Fikar, C. e P. Hirsch (2017). "Home health care routing and scheduling: A review". Em: Computers Operations Research; 77:86-95.

Lin, C.-C., L.-P. Hung, W.-Y. Liu e M.-C. Tsai (2017). "Jointly rostering, routing, and rerostering for home health care services: A harmony search approach with genetic, saturation, inheritance, and immigrant schemes". Em: Computers Industrial Engineering; 115:151-166.

Mankowska, D., F. Meisel e C. Bierwirth (2014). "The home health care routing and scheduling problem with interdependent services". Em: Health Care Manag Sci; 17(1):15-30.

Projeções de População Residente em Portugal (2017). URL: https://www.ine.pt/xportal/xmain?xpid=INE&xpgid=ine_destaques&DESTAQUESdest_boui=277695619&DESTAQUESmodo=2&xlang=pt.

Shao, Y., J. Bard e A. Jarrah (2012). "The therapist routing and scheduling problem". Em: IIETransactionsonOperationsEngineering Analysis; 44(11):868-893.

Shao, Y., J. Bard e A. Jarrah (2013). "A sequential GRASP for the therapist routing and scheduling problem". Em: J Sched; 17(2):109-33.

Trautsamwieser, A. e P. Hirsch (2014). "A branch-price-and-cut approach for solving the medium-term home health care planning problem". Em: Networks 2014; 64(3):143-59.

URL - Bootsrap. URL: https://getbootstrap.com/.

URL - CSS. URL: https://www.w3schools.com/css/.

URL - Hibernate. URL: http://hibernate.org/.

URL - HTML. URL: https://www.w3schools.com/html/.

URL - HTTP. URL: https://woliveiras.com.br/posts/url-uri-qual-diferenca/.

URL - IFML. URL: https://www.ifml.org/.

URL - Java Spring. URL: https://spring.io/.

URL - Javascript. URL: https://www.javascript.com/.

URL - JPA. URL: https://www.devmedia.com.br/introducao-a-jpa-java-persistence-api/28173.

URL - JSON. URL: https://www.w3schools.com/whatis/whatis_json.asp.

URL - MySql. URL: https://www.mysql.com/.

URL - Oracle. URL: https://www.oracle.com/index.html.

URL - ORM. URL: https://www.devmedia.com.br/orm-object-relational-mapper/19056.

URL - React. URL: https://reactjs.org/.

URL - REST. URL: https://www.codecademy.com/articles/what-is-rest.

URL - Restfull Api. URL: https://becode.com.br/o-que-e-api-rest-e-restful/.

URL - Sat4j. URL: http://www.sat4j.org.

URL - SMT. URL: https://en.wikipedia.org/wiki/Satisfiability_modulo_theories.

URL - SMT Interpol. URL: https://ultimate.informatik.uni-freiburg.de/smtinterpol/.

Anexo I

Annex 1

I.1 Solução - Abordagem Inicial

O problema tratado nesta tese é composto por muitas restrições e especificações próprias. Assim, antes de se passar à modelação do mesmo, decidiu-se trabalhar sobre uma versão relaxada, o Traveling Salesman Problem.

Como mencionado anteriormente o TSP é uma simplificação do verdadeiro problema a que se pretende estudar. Uma vez que não são consideradas restrições como as janelas temporais ou a continuidade de cuidados, optou-se por estudar inicialmente uma versão mais simples, pois acredita-se que a melhor abordagem para a resolver, poderá ser também a melhor para o verdadeiro problema.

O Traveling Salesman Problem (TSP) é um problema de otimização NP-difícil. É um problema que, a partir de um ponto inicial e de conjunto de cidades, tem como objetivo determinar a melhor rota, de maneira a que essas mesmas cidades, sejam percorridas uma, é só uma vez, e se se regresse ao ponto inicial. A melhor rota, objetivamente, é a rota na qual é efetuado o menor percurso possível medido, por exemplo, em tempo, em distância, em custos ou noutra métrica adequada.

O TSP pode ser formalizado com recurso a um grafo onde um nó representa um local a ser visitado e um arco representa, por exemplo, a distância entre dois nós. Para uma solução ser considerada admissível, todos os nós têm de ser visitados. A solução terá que ser composta por um conjunto de arcos, de modo a que exista uma entrada e

uma saída de cada nó. A solução ótima representa a solução admissível, na qual a soma das distâncias dos arcos é minimizada.

Como já foi dito, foram investigadas algumas abordagens de resolução do TSP, das quais nenhuma foi encontrada na literatura. De acordo com o que foi revisto, a melhor abordagem seria através da criação de uma meta-heurística, pois pela literatura os métodos exatos não são uma boa solução para problemas de média/larga escala. Mas era importante ter a certeza, e perceber se alguma delas poderia ser a melhor estratégia para atacar o verdadeiro problema.

A ideia inicial passava por integrar um solver SAT. Nas subsecções seguintes é feita uma análise e conclusão de resultados das restantes abordagens. Foram testadas três instâncias do problema, variando o número de nós (5,40,100). Foi feito uma média de três testes para cada instância.

Todos os testes que seguem foram executados num computador pessoal, num Windows de 64 bits; processador Intel i5 em 2.20 GHz; e 8GB de memória RAM.

I.1.1 Abordagem SAT

Boolean Satisfiability Problem (SAT) é um problema de decisão, que determina se existe uma solução que satisfaça uma fórmula booleana, formada por conjunções ($\wedge$), disjunções ($\vee$) e negações ($\neg$). Esta fórmula apresenta-se na denominada Conjunctive Normal Form (CNF) sendo formada por uma conjunção de cláusulas. Cada cláusula é uma disjunção de literais em que cada literal é uma variável booleana, ou a sua negação. A solução da fórmula é a atribuição de 0's e 1's às variáveis existentes, de modo a que a expressão seja verdadeira.

Exemplo: $(\neg x1 \vee x2 \vee x3) \wedge (\neg x2 \vee \neg x3 \vee x4) \wedge (x3 \vee x1 \vee x4)$
Solução: $x1 = 0; x2 = 0; x3 = 1; x4 = 1$

Na versão modelada, cada variável representa um arco. Foram considerados apenas 6 nós, porque a formulação do problema não foi dinâmica e teve de ser explicitada num ficheiro de texto (ver figura I.1). Para além das restrições que determinam que todos os nós têm de ser visitados, foram adicionadas outras restrições. Estas últimas impedem a geração de uma solução com subciclos, de maneira a que a solução gerada represente um caminho conexo. Isto é, um caminho em que todos os nós são visitados uma e uma só vez.

Foi então criado um ficheiro, no qual é representado o problema, em CNF. Para determinar se existe uma solução admissível, foi usado o SAT Solver disponibilizado

pela biblioteca Sat4j (*URL - Sat4j*). A Figura I.2 permite-nos analisar os resultados obtidos.

```
c
c start with comments
c cada arco(i,j) -> variavel
c no inicial -> 1 / no final -> 6
c nos intermediários têm necessariamente uma chegada e uma saída
c restrições para formar um caminho satisfazivel
c 12 -> 1  / 13 -> 2  / 14 -> 3  / 15 -> 4  /
c 23 -> 5  / 24 -> 6  / 25 -> 7  / 26 -> 8  /
c 32 -> 9  / 34 -> 10 / 35 -> 11 / 36 -> 12 /
c 42 -> 13 / 43 -> 14 / 45 -> 15 / 46 -> 16 /
c 52 -> 17 / 53 -> 18 / 54 -> 19 / 56 -> 20 /
c
p cnf 20 78
1 2 3 4 0
5 6 7 8 0
9 10 11 12 0
13 14 15 16 0
17 18 19 20 0
1 9 13 17 0
2 5 14 18 0
3 6 10 19 0
4 7 11 15 0
8 12 16 20 0
-2 -5 0
-3 -6 0
-4 -7 0
-1 -9 0
-3 -10 0
-4 -11 0
-1 -13 0
-2 -14 0
-4 -15 0
-1 -17 0
-2 -18 0
-3 -19 0
-5 -2 0
-6 -3 0
-7 -4 0
```

PostScript file

Figura I.1: Ficheiro com parâmetros de entrada do problema a ser resolvido pelo Solver SAT

```
Linha de comandos
c #vars     20
c #constraints  78
c constraints type
c org.sat4j.minisat.constraints.cnf.OriginalBinaryClause => 68
c org.sat4j.minisat.constraints.cnf.OriginalWLClause => 10
c starts                : 1
c conflicts             : 0
c decisions             : 7
c propagations          : 20
c inspects              : 54
c shortcuts             : 0
c learnt literals       : 0
c learnt binary clauses : 0
c learnt ternary clauses        : 0
c learnt constraints    : 0
c ignored constraints   : 0
c root simplifications  : 1
c removed literals (reason simplification)      : 0
c reason swapping (by a shorter reason) : 0
c Calls to reduceDB     : 0
c speed (assignments/second)    : 1538.4615384615386
c non guided choices    7
c learnt constraints type
c constraints type
c org.sat4j.minisat.constraints.cnf.OriginalBinaryClause => 68
c org.sat4j.minisat.constraints.cnf.OriginalWLClause => 10
s SATISFIABLE
v -1 -2 3 -4 -5 -6 -7 8 -9 -10 11 -12 -13 14 -15 -16 17 -18 -19 -20 0
c Total wall clock time (in seconds) : 0.027
```

Figura I.2: Resultados do Solver SAT

Os resultados permitem-nos perceber que a solução encontrada pelo solver é satisfazível. As variáveis que entram na solução são: 3; 8; 11; 14; 17. O que significa que o caminho satisfazível é: $1 \rightarrow 4 \rightarrow 3 \rightarrow 5 \rightarrow 2 \rightarrow 6$.

A rapidez de resolução (0.027 segundos) poderia levar-nos a pensar que o uso de um SAT Solver seria uma boa abordagem para o problema do apoio domiciliário. Contudo, o nosso é um problema de otimização, e não de admissibilidade. Na instância testada, não são consideradas, por exemplo, distâncias entre nós. Representá-las numa expressão CNF não seria uma tarefa impossível, mas levaria a um aumento bastante considerável de variáveis e restrições, o que iria pôr em causa a eficiência da sua resolução.

Conclui-se que resolver o problema através de um Solver SAT não é a melhor das abordagens. Ainda assim, ficou claro que este tipo de solvers são bastante poderosos na resolução de outros tipos de problemas.

I.1.2 Abordagem SMT

Solvers de Satisfability Modulo Theories (*URL - SMT*) permitem verificar a satisfatibilidade de fórmulas lógicas de primeira ordem. Estes são uma extensão dos SAT solvers que permitem representar um peso (distância) nos arcos, sendo assim possível ultrapassar a limitação encontrada nos SAT solvers. Foi feito, em Java, um pequeno programa, no qual é integrado um solver SMT, disponibilizado pela biblioteca SmtInterpol (*URL - SMT Interpol*).

Apesar de ser um solver de satisfatibilidade, decidiu-se estudá-lo, pois seria possível usar as suas potencialidades para resolver problemas de otimização. Isto poderia ser feito, por exemplo, executando constantemente o solver e ir acrescentado clausulas ao problema, de modo a que a nova solução fosse melhor que a anterior (processo iterativo).

Tal como na abordagem referente ao solver Pseudo-Boolean, apenas se considerou, como restrições, que cada nó teria obrigatoriamente uma entrada e uma saída, sendo que o nó representativo do nó inicial não tem entrada; tal como o nó que representa o nó final, onde termina o caminho, não tem saída.

Da análise dos tempos computacionais (Tabela I.1) efetuados, podemos verificar que os resultados foram piores que as abordagem estudadas nas subsecções seguintes.

Tabela I.1: SMT Solver - resultados

	5 nós	40 nós	100 nós
Solução Admissível	0.214 segundos	3.231 segundos	30.104 segundos

I.1.3 Abordagem Google Optimization Tools

Nesta subsecção apresentamos o estudo de uma solução tecnológica, com recurso a um maior poder de computação.

O Google disponibiliza um conjunto de ferramentas para solucionar problemas de programação linear. Basta instalar um add-on; criar uma folha de cálculo no Google Docs; preencher a folha com as restrições e a função objetivo que representam o problema; clicar "Solve", e a solução do problema é-nos mostrada. Para podermos testar a ferramenta nas mesmas condições das abordagens anteriores, foi criado um programa em Java que permite o envio de informação para a folha de cálculo criada, a partir do seu ID. Assim, podemos preencher as células da folha de cálculo automaticamente, podendo testar o comportamento do solver, variando o número de nós.

Os resultados (Tabela I.2) foram melhores que as anteriores abordagens, provavelmente devido ao facto de se estar a usar o poder computacional disponibilizado pela Google e não um computador pessoal, usado anteriormente. Apesar da melhoria, os resultados não são, ainda assim, satisfatórios no que diz respeito ao uso desta tecnologia na solução final. Pelo menos, de maneira direta, uma vez que problemas de larga escala fazem ultrapassar o limite máximo de células existente na folha de cálculo.

Tabela I.2: Google Optimization Tools Solver - resultados

	5 nós	40 nós	100 nós
Solução Ótima	2.91 segundos	21.14 segundos	Ultrapassa o limite de 2000000 células disponibilizado pela folha de cálculo

I.1.4 Abordagem Pseudo-Boolean

A biblioteca referida anteriormente permite também a resolução de problemas Pseudo-Boolean. Um solver deste tipo não é tão limitado. A sua utilização permitiria atribuir um peso às variáveis booleanas, e definir uma função objetivo, de modo a podermos retornar uma solução otimizada.

Foi codificado um pequeno programa, em Java, no qual é integrado um solver Pseudo-Boolean. Como referido anteriormente, este é um solver de otimização, pelo que permitiria ultrapassar a limitação identificada relativamente ao Solver SAT.

Tal como para o SAT, apenas se considerou, como restrições, que cada nó teria obrigatoriamente uma entrada e uma saída, sendo que o nó representativo do nó inicial

não tem entrada; tal como o nó que representa o nó final, onde termina o caminho, não tem saída. Foram testadas três instâncias do problema, variando o número de nós. O objetivo era analisar o tempo que o solver demora a determinar uma solução admissível, bem como, a solução ótima.

Na Tabela I.3 podemos observar os resultados obtidos. Foi curioso verificar que para instâncias de média/grande dimensão foram encontradas soluções admissíveis em menos de um segundo. Contudo, o tempo limite definido (2 horas) foi excedido na procura da solução ótima, mesmo com tão poucas restrições consideradas.

Tabela I.3: Pseudo Boolean Solver - resultados

	5 nós	40 nós	100 nós
Solução Admissível	0.0097 segundos	0.049 segundos	0.196 segundos
Solução Ótima	0.119 segundos	Tempo limite de 7200 segundos foi excedido	Tempo limite de 7200 segundos foi excedido

I.1.4.1 Aprofundamento

Tendo em consideração os resultados obtidos, procurou-se explorar o solver Pseudo-Boolean, de modo a perceber se seria uma abordagem viável para a solução do problema de apoio domiciliário.

Foi feito, em Java, um programa, no qual é integrado um solver da biblioteca Sat4j, e que resolve uma variante do planeamento de serviços de apoio domiciliário para um horizonte temporal de um dia. Depois, compararam-se os resultados computacionais com outros artigos que também consideram métodos exatos como método de resolução.

O programa desenvolvido não está preparado para considerar todas as restrições relativas ao nosso caso de estudo específico. Trabalhou-se, portanto, sobre uma versão um pouco mais generalizada, até de modo a poder fazer melhores comparações com resoluções apresentadas noutros artigos.

O problema é estruturado através de um grafo G (N, A), onde N é um conjunto de nós e A é um conjunto de arcos – pares de nós. Um nó representa uma tarefa a ser realizada. É constituído por uma posição geográfica, uma janela de tempo na qual a mesma deve ser realizada, e a sua duração prevista. Um arco é um par de nós percorridos por uma determinada equipa. A solução é um conjunto de arcos, que representam as rotas a ser efetuadas pelas equipas existentes.

Mais à frente será explicado como são modeladas as restrições neste solver, e porque não é possível modelar as restrições relativas às janelas temporais diretamente no solver. Por não ser possível, tentou-se considerá-las na função objetivo:

$$minS_{xy}A^t_{xy}$$

A^t_{xy} é um arco percorrido pela equipa t entre os nós x e y. S_{xy} é um valor associado a cada arco, que representa o melhor tempo possível a que a tarefa a ser realizada no nó y pode ter início, tendo em conta a sua janela temporal e a distância percorrida desde o nó x.

Dando um exemplo prático, podemos considerar um arco A^t_{xy}, no qual o nó x tem uma janela temporal de 90 a 150 (9h30 até 10h30), com a tarefa a ter uma duração de 30 min; um nó y, de chegada, com uma janela temporal compreendida entre 120 a 220 (10h00 até 11h40). Sabendo ainda que a distância entre os nós x e y é de 10 km, o que equivale a cerca de 20 minutos percorridos de carro. A melhor hora possível a que a tarefa do nó y pode ser inicializada, a partir do nó x, é de 140 (início da tarefa em x + duração da tarefa em x + duração da viagem entre x e y, ou seja, 90+30+20 =140).

A ideia passava por calcular a melhor hora possível a que uma tarefa possa ser inicializada a partir de qualquer nó. No caso de não ser possível, ou seja, 18 quando a melhor hora possível excede a janela temporal, atribuiu-se um tempo exageradamente elevado, pelo que muito penalizador da função objetivo (ex:1000).

O grande problema aqui, é que esta estratégia serve apenas para modelar a situação inicial. Com o decorrer das rotas a serem percorridas, a melhor hora possível que uma tarefa possa ser inicializada pode ser alterada, pois está dependente do percurso efetuado até então pelos outros nós. A solução passava, então, por verificar no final da execução do solver, se as rotas geradas satisfazem as janelas temporais de todos os nós.

As restrições são representadas por um conjunto de cláusulas. No solver, foram usadas essencialmente duas funções: at least(i,arc[]) e at most(i,arc[]), que significam respetivamente que, dentro de um conjunto de arcos, pelo menos i elementos têm de constar na solução final; e no segundo caso, no máximo i elementos podem fazer parte da solução.

As restrições modeladas foram:

- Todos os nós têm uma saída - Garantir que um, e só um, arco com o nó inicial x entra na solução;

- Todos os nós têm uma entrada - Garantir que um, e só um, arco com o nó final y entra na solução;
- Impedir que a mesma rota não seja efetuada no caminho inverso - garantir que dos arcos com os nós x → y e y → x, só um entra na solução;
- Formar caminho satisfazível para todas as equipas - garantir que a entrada e saída do nó é feito pela mesma equipa;
- Equilibrar o trabalho a ser efetuado pelas equipas – garantir um equilíbrio de visitas por parte das equipas. Por exemplo, numa situação em que haja 3 equipas, e 15 nós a serem visitados, esta restrição vai obrigar a que cada equipa visite 5 nós. Este tipo de modelação pode resultar numa solução "não justa" no sentido em que apesar de existir um equilíbrio de nós visitados, as rotas a serem efetuadas tenham distâncias bastante diferentes, o que leva a um desequilíbrio a nível de tempo gasto para cumprir o trabalho proposto para cada equipa.

Tendo em conta que não é possível modelar completamente o problema pretendido através das clausulas do solver, visto que a estratégia modelada apenas serve para a situação inicial do problema, a estratégia passou a ser: guardar soluções, à medida que estas vão sendo encontradas pelo solver, e em pós-processamento, analisá-las à parte. Sendo assim, no fim do solver ser executado, é verificado, quais destas soluções não têm sub-ciclos, e quais cumprem as janelas temporais previamente definidas. A melhor é retornada.

Comparação de Resultados

Foram encontrados seis artigos referentes ao problema do planeamento de serviços de apoio domiciliário, nos quais é considerado um único dia de planeamento, em que são disponibilizados um conjunto de benchmarks, disponibilizados em anexo. Destes seis artigos, quatro utilizam meta-heurísticas. Sendo assim, analisou-se apenas dois artigos: Braekers et al. 2016 e Mankowska et al. 2014. Em anexo está contida toda a informação relativa aos mesmos: janelas temporais dos casos de estudo; matrizes de distâncias entre nós e resultados obtidos.

Braekers et al. 2016:

Os autores consideraram um conjunto de restrições e especificações que não se aplicam no nosso problema, nomeadamente, níveis de habilitação de cada funcionário, o que pressupõe que cada tarefa não pode ser efetuada por qualquer funcionário. Também são considerados diferentes modos de transporte. As instâncias usadas por nós,

como comparação, consideram apenas carro como o meio de deslocação entre nós. Ao contrário do nosso caso, não existe um "centro" de onde partem todos os funcionários. De maneira a poder fazer integrar estes benchmarks no programa desenvolvido, considerou-se que todas os funcionários estariam localizados nas coordenadas relativas ao primeiro funcionário mencionado no artigo. Neste artigo, tentou-se resolver os casos pequenos (até cerca de 25 nós) através de métodos exatos. Os restantes são resolvidos com recurso a meta-heurísticas.

Caso nº1: 10 Tarefas / 2 Equipas

A solução do artigo Braekers et al. 2016 demorou menos de 60 segundos para encontrar o ótimo. A nossa solução encontrou o ótimo em cerca de 62 segundos. Foram encontradas 9 possíveis soluções. A melhor cumpria as restrições das janelas temporais.

Caso nº2: 15 Tarefas / 4 Equipas

A solução do artigo Braekers et al. 2016 demorou menos de 60 segundos para encontrar o ótimo. A nossa solução não encontrou o ótimo ao fim de 5 minutos. Foram encontradas 10 possíveis soluções. A melhor cumpria as restrições das janelas temporais.

Caso nº3: 20 Tarefas / 5 Equipas

No artigo Braekers et al. 2016, a solução ótima foi encontrada em cerca de 45 minutos. No nosso caso, o solver encontrou 12 possíveis soluções no espaço de 1 hora. Nenhuma satisfazia as condições das janelas temporais.

Caso nº4: 25 Tarefas / 7 Equipas

No artigo Braekers et al. 2016, a solução ótima não foi encontrada ao fim de dois dias. No nosso caso, o solver encontrou 4 possíveis soluções em 36 horas. Nenhuma satisfazia as condições das janelas temporais.

Mankowska et al. 2014:

Novamente, existem algumas diferenças em relação à modelação do problema. Este artigo apresenta não só o tempo computacional que demora a ser encontrada a solução ótima, mas também, nos casos em que o ótimo não é encontrado, se a solução é boa ou não. Na prática, se a solução encontrada está, ou não, perto da solução ótima. Contudo, como a função objetivo é diferente da nossa proposta, torna-se complicado, logo à priori, poder fazer comparações justas.

Neste artigo, também não se considera que todos os funcionários estão aptos a realizar qualquer tarefa.

Caso nº1: 10 Tarefas / 3 Equipas

A solução do artigo Mankowska et al. 2014 demorou menos de 60 segundos para encontrar o ótimo. A nossa solução não encontrou o ótimo ao fim de 10 minutos. Foram encontradas 9 possíveis soluções (a última ao fim de 4 segundos). A melhor cumpria as restrições das janelas temporais.

Caso nº2: 25 Tarefas / 5 Equipas

No artigo Mankowska et al. 2014 não foi encontrado o ótimo ao fim de 10 horas. Foram encontradas soluções admissíveis. A nossa solução não encontrou o ótimo ao fim de 16 horas, sendo que foram encontradas 25 possíveis soluções. A sexta mais promissora foi provada como admissível.

Podemos concluir que os resultados ficaram um pouco aquém do esperado. Tendo em conta as dificuldades de modelação do problema do planeamento de serviços de apoio domiciliário, e do elevado tempo computacional que demorou a serem resolvidos os casos já explicitados, decidiu-se não prosseguir, nem aprofundar mais esta abordagem, passando então para o desenvolvimento de uma meta-heurística.