RAFAEL HENRIQUE VARETO

Orientador: Haroldo Gambini Santos

PROGRAMAÇÃO INTEIRA E HEURÍSTICAS PARA O PROBLEMA DE ESCALONAMENTO DE ENFERMEIROS

Ouro Preto

Setembro de 2013

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

PROGRAMAÇÃO INTEIRA E HEURÍSTICAS PARA O PROBLEMA DE ESCALONAMENTO DE ENFERMEIROS

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

RAFAEL HENRIQUE VARETO

Ouro Preto
Setembro de 2013

UNIVERSIDADE FEDERAL DE OURO PRETO

FOLHA DE APROVAÇÃO

PROGRAMAÇÃO INTEIRA E HEURÍSTICAS PARA O PROBLEMA DE ESCALONAMENTO DE ENFERMEIROS

RAFAEL HENRIQUE VARETO

Monografia defendida e aprovada pela banca examinadora constituída por:

Dr. Haroldo Gambini Santos – Orientador
Universidade Federal de Ouro Preto

Dr. Luiz Henrique de Campos Merschmann
Universidade Federal de Ouro Preto

Dr. Marcone Jamilson Freitas Souza
Universidade Federal de Ouro Preto

Ouro Preto, Setembro de 2013

Resumo

Ao contrário da maioria das empresas onde os turnos são durante a semana, hospitais trabalham vinte e quatro horas por dia e sete dias por semana. Por esse e muitos outros motivos existe uma grande dificuldade em gerar escalas trabalhistas de boa qualidade para problemas deste contexto.

Nesta monografia são apresentados os detalhes do estudo de Programação Inteira do clássico Problema de Escalonamento de Enfermeiros, o qual leva em consideração a distribuição da carga horária dos enfermeiros em turnos em um determinado horizonte de planejamento.

Neste trabalho é abordado inicialmente uma solução compacta e simples gerada por um algoritmo guloso que produz soluções válidas mas não de boa qualidade. Em seguida são utilizados métodos de refinamento executados concorrentemente – estas estratégias são: geração de subproblemas através de fixações e heurísticas – que melhoram os resultados de forma expressiva.

Os experimentos computacionais realizados durante o período de confecção da Monografia, e que possuem as técnicas apresentadas nesta abordagem, alcançaram algumas soluções interessantes, mesmo não obtendo a melhor solução para todas as instâncias. Dentre essas soluções podemos destacar o alcance de resultados já obtidos por outros pesquisadores utilizando uma ferramenta de código aberto.

Palavras-chave: Escalonamento de Enfermeiros, Programação de Horários, Programação Inteira Mista.

Abstract

Unlike most of the companies which have shifts during the week, hospitals work twenty-four hours a day and seven days a week. This is one of the many reasons why it is very difficult to make up good quality work schedules due to all kinds of problems.

The details of the study of the classic problem of timetabling nurses, are presented in this dissertation, which takes into consideration the workload distribution of the nurses in shifts in a specific planning environment.

In this work a compact and simple solution is initially approached, generated by a greedy algorithm that comes up with valid solutions but not good quality ones, then concurrent refinement methods are used from the initial solution – these strategies are: creation of sub-problems through heuristics and neighbourhood fixations – which significantly improve the results.

The computational experiments carried out while the dissertation was being made, and which contain the techniques presented in this approach, achieved some interesting results, although the best solution was not obtained in all instances. From these solutions we can highlight results which have already been obtained by other researchers using an open-source tool.

Keywords: Nurse Rostering Problem, Timetabling, Mixed Integer Programming.

Dedico este trabalho aos meus pais, Antônio José Vareto e Josenete Braga Vareto, e à minha família. Os maiores exemplos de minha vida.

Agradecimentos

Temo que nesta lista de agradecimentos haja a ausência de uma ou outra pessoa que teve uma participação, ainda que pequena, na conclusão de um trabalho como este. Independentemente, registro aqui os meus agradecimentos, tendo em mente que ainda são insuficientes para expressar toda a minha gratidão.

Primeiramente agradeço a Deus, pois me deu a vida e tem me dado força, dia após dia, para continuar.

Agradeço também aos meus pais Antônio e Josenete pelo seu incessável amor, apoio, carinho e direção. À minha irmã Suelen, um exemplo vivo de determinação, pela amizade e pela ajuda.

Ao meu orientador Haroldo, pela paciência e conhecimento transmitido, possibilitando transformar este projeto em realidade. À todos os professores da UFOP, devo honrá-los, pois me ensinaram grande parte de todo o meu conhecimento.

Aos meus amigos Samuel Souza, Marco Túlio e Gabriel Resende, pela ajuda e esclarecimento de dúvidas em momentos antecedentes aos exames.

Aos meus amigos Filipe Eduardo, Thaís Gonçalves, Sabrina Braga, Carlos Henrique, Tainá Mendonça, Isabela Aredes e Matheus Assis, por deixarem que eu fizesse parte de suas vidas e me confortarem em momentos difíceis.

À todos aqueles que oram por mim e desejam a mim sucesso.

"Lá vem aquele sonhador!", diziam uns aos outros. – Gênesis 37:19

Sumário

1 Introdução **1**

1.1 Motivação 2

1.2 Justificativa 2

1.3 Objetivos 3

1.3.1 Objetivos Específicos 3

1.4 Estrutura do Trabalho 3

2 Revisão Bibliográfica **5**

2.1 Heurísticas 5

2.2 Metaheurísticas 7

2.2.1 Multi-Start 8

2.2.2 Variable Neighborhood Search 9

2.2.3 Relaxation Induced Neighborhood Search 10

2.3 Programação com Multi-Processos 11

3 Descrição do Problema Abordado **12**

3.1 Restrições 12

3.2 Formulação 14

3.2.1 Dados de entrada 14

3.2.2 Variáveis de Decisão 16

3.2.3 Função Objetivo 16

3.2.4 Restrições 17

4 Desenvolvimento **19**

4.1 Representação de uma Solução 20

4.2 Solução Inicial 21

4.3 Estrutura de Vizinhança 23

4.3.1 Fixação de Dias 24

4.3.2 Fixação de Enfermeiros 25

4.3.3 Fixação Mista 25

4.3.4 Fixação de valores Idênticos 26
4.4 Algoritmo Proposto 26

5 Experimentos Computacionais **30**
5.1 Instâncias INRC 30
5.1.1 Formato do arquivo de entrada 33
5.1.2 Compatibilidade com CBC 34
5.2 Experimentos 36
5.3 Resultados 37

6 Conclusões **42**

Referências Bibliográficas **46**

Lista de Figuras

4.1 *Fixação dos Dias* de uma vizinhança com $p = 3$ 24
4.2 *Multi-Start*(a) e *Best-Start*(b) multi-processados 29

5.1 Gráfico comparativo com instâncias *Sprint* 40
5.2 Gráfico comparativo com instâncias *Medium* 41
5.3 Gráfico comparativo com instâncias *Long* 41

Lista de Tabelas

3.1 Índices das restrições fracas de faixa e binárias 15

4.1 Exemplo reduzido de uma escala trabalhista 20
4.2 Exemplo de escala trabalhista reduzida representada por algarismos 21
4.3 Exemplo de aplicação do movimento Adiciona/Remove turno 22

5.1 Detalhes das instâncias Sprint 31
5.2 Detalhes das instâncias Medium e Long 32
5.3 Tempo de geração de arquivos LP em segundos 35
5.4 Distribuição do tempo de execução do algoritmo MS e BS 37
5.5 Resultado das intâncias Sprint 38
5.6 Resultado das intâncias Medium e Long 39

6.1 Exemplo de solução para instâncias *sprint_hidden* 44
6.2 Exemplo de solução para instâncias *medium_hidden* 45

Lista de Algoritmos

2.1 Heurística Construtiva 6
2.2 Heurística de Refinamento 7
2.3 Multi-Start 8
2.4 Variable Neighborhood Search 9
2.5 Variable Neighborhood Descent 10

4.1 GeraSoluçãoInicial 21
4.2 Estrutura de Vizinhança 24
4.3 FixaçãoRINS 26
4.4 Versão Inicial 27
4.5 Versão Multi-Start 27
4.6 Versão Best-Start 28

Capítulo 1

Introdução

Este documento trata do Problema de Escalonamento de Enfermeiros. Neste problema o objetivo é confeccionar uma tabela operacional, ou seja, uma escala de trabalho válida para os enfermeiros satisfazendo várias restrições trabalhistas, operacionais e, se ainda for possível, as preferências pessoais.

A grande maioria dos hospitais trabalham vinte e quatro horas por dia, assim sendo, é necessário produzir uma escala de serviços para as equipes de enfermagem periodicamente. Caso essa escala seja elaborada manualmente, tem-se, então, uma tarefa muito difícil demandando uma grande quantidade de tempo.

Para tornar mais fácil a boa elaboração das atividades de todos os enfermeiros, sobretudo, é essencial que os mesmos tenham uma escala de trabalho (planejamento dos turnos) equilibrada ao longo de todo o mês, fazendo o possível para satisfazer requisições pessoais e respeitando todas as normas trabalhistas. É muito importante que a escala de trabalho proposta para cada enfermeiro esteja de acordo com as legislações do país e as normas internas de cada hospital, e que toda a demanda interna seja suprida.

Para resolução deste problema, todo o trabalho é fundamentado nas descrições da International Nurse Rostering Competition (INRC) criada em 2010 (Haspeslagh et al., 2010). Um dos objetivos da INRC é motivar o estudo e a elaboração de novas abordagens para os problemas adjuntos, e se possível elaborar um *benchmark*, atraindo pessoas de todas as áreas de pesquisa.

O problema dos enfermeiros em questão envolve a atribuição de turnos levando em conta algumas restrições. Em geral, devemos considerar dois tipos básicos de restrições (Causmaecker e Berghe, 2003): Restrições Fortes e Restrições Fracas. Restrições fortes são aquelas que devem ser satisfeitas de qualquer maneira. Por outro lado, restrições fracas são um conjunto de restrições que devem ser satisfeitas sempre que possível, mas sabendo que nem todas poderão ser atendidas. Desta maneira, a demanda e o número de turnos a serem "cobertos" por dia são consideradas restrições fortes (*hard*), já as preferências pessoais dos enfermeiros são caracterizadas como restrições fracas (*soft*). Uma solução validada é aquela onde todas

as restrições fortes são obrigadamente atendidas. Consequentemente, a qualidade da solução é medida de acordo com o número de restrições fracas respeitadas.

Geralmente, na prática, é suficiente encontrar uma "boa" solução para o problema em tempo viável, não precisando ser o ótimo global que necessariamente demanda um grande esforço computacional e uma taxa de tempo muito maior.

1.1 Motivação

Problemas de escalonamento de horários (*timetabling*) são comumente encontrados em citações bibliográficas, tal popularidade foi obtida graças a dificuldade de garantir que a solução obtida é a melhor solução possível.

Algumas considerações são comuns na maioria dos trabalhos encontrados na literatura: os enfermeiros, em geral, preferem trabalhar uma certa sequência de dias no mesmo horário/turno. Devido a isso é necessário uma carga horária mais flexível aos empregados, principalmente com a escassez de enfermeiros (Mueller et al., 1990). Por isso, a gerência do hospital deve levar em conta e analisar as preferências pessoais dos enfermeiros e também os pedidos de dias de folga.

Geralmente em hospitais, bastante tempo é gasto tentando determinar escalas que são viáveis e de qualidade para os enfermeiros. Apesar do Problema de Escalonamento de Enfermeiros ser um problema combinatório de difícil otimização, ele tem ganhado uma ampla atenção nos últimos anos, provavelmente pelo aumento crescente da demanda hospitalar por enfermeiros resultante da grande quantidade de pacientes.

A construção manual de quadros de horários, a cada dia, vem se tornando uma tarefa mais complicada e trabalhosa: além das restrições básicas, um bom quadro de horários deve levar em conta muitos outros requerimentos, considerando necessidades institucionais e pessoais (relacionadas à equipe de enfermeiros). De fato, um dos critérios que define a qualidade, e mesmo a viabilidade, de um quadro de horários é bastante específico e dependente do sistema hospitalar do país.

1.2 Justificativa

Hospitais são instituições sem fins lucrativos, mas isto não significa que estas instituições não se importam com déficits. Pelo fato de ser um transtorno bastante comum, e por possuir significância econômica e dificuldade de solução, o problema tem atraído a comunidade de pesquisa operacional.

Como os serviços de enfermagem possuem um dos maiores componentes de custo nos orçamentos de hospitais, é essencial que se desenvolva um bom planejamento. Em particular, é muito importante utilizar o tempo e esforço eficientemente, tanto para distribuir o mais unifor-

memente possível a carga horária entre os enfermeiros, quanto para atender suas preferências pessoais.

Para a instituição, a geração de quadros de horários otimizados permite a construção de escalas com maior qualidade, garantindo a cobertura correta de profissionais ao longo do período e diminuindo o montante necessário de horas extras exigidas dos funcionários, o que implica em uma significativa redução de custos. Para a equipe, obtém-se como benefício uma melhor qualidade de vida para os funcionários, visto que a escala gerada permite uma distribuição mais uniforme dos benefícios como, por exemplo, as folgas em feriados e fins de semana. A melhoria no ambiente de trabalho permite que se ofereça um melhor atendimento à população.

1.3 Objetivos

O objetivo, em âmbito geral do projeto, é desenvolver um sistema para gerenciar a escala operacional de enfermeiros de forma automática e otimizada.

1.3.1 Objetivos Específicos

- Desenvolver o sistema utilizando as linguagens de programação C/C++;
- Utilizar a biblioteca *COIN-OR Open Solver Interface*;
- Implementar técnicas de concorrência que possam acelerar a busca no espaço de soluções;
- Permitir a geração automática da escala operacional dos enfermeiros, de acordo com a necessidade individual de cada um deles;
- Equilibrar a carga horária trabalhada por cada um dos enfermeiros, de forma que nenhum deles fique sobrecarregado ou desocupado;
- Realizar testes com diferentes tipos de instâncias disponibilizadas pela INRC.

1.4 Estrutura do Trabalho

A monografia está dividida em seis capítulos, incluindo esta Introdução.

O Capítulo 1 apresenta uma breve descrição do universo dos problemas *timetabling*, além de apresentar o problema e a motivação necessária para alcançar os objetivos pretendidos. No Capítulo 2 temos uma revisão bibliográfica sobre variados métodos amplamente utilizados na resolução de problemas de escalonamento de enfermeiros e afins.

A definição do Problema de Escalonamento de Enfermeiros juntamente com sua formulação são apresentadas no Capítulo 3. No Capítulo 4 é elaborado o algoritmo proposto, baseado em diferentes fixações de variáveis, responsáveis em gerar diferentes vizinhanças.

O Capítulo 5 expõe os resultados computacionais obtidos durante um longo período de experimentos. Os mesmos são analisados para verificar a validade das estratégias propostas e implementadas.

Por fim, no Capítulo 6, são apresentadas as conclusões e apontadas as sugestões para trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

A Pesquisa Operacional é uma área bastante focada por pesquisadores. Este universo abrange uma gama muito grande de problemas, para citar alguns temos: Escalonamento de tripulações aéreas (Souza et al., 2006), Sequenciamento de tarefas (Lenstra et al., 1977), Roteamento de veículos (Mine et al., 2010), Escalonamento de salas de aula (Silva et al., 2005), entre outros. Estes problemas são geralmente resolvidos por artifícios de Programação Linear Inteira. Esta técnica é geralmente usada para elaborar e solucionar muitos problemas práticos de Otimização Combinatória.

A Programação Linear Inteira (Nepomuceno, 2006) se baseia em alguns conceitos de Programação Linear, em que um problema é explícito em termos de variáveis contínuas e um conjunto de restrições lineares sobre essas variáveis. Acontece que, em alguns problemas, as variáveis precisam assumir valores inteiros, e nunca contínuos. Consequentemente, se algum destes problemas incorporar uma restrição estabelecendo que todas as suas variáveis têm de apresentar valores inteiros, temos então um problema que precisa ser resolvido pela Programação Linear Inteira.

Os estudos recentes disponíveis na literatura têm mostrado que algoritmos baseados em heurísticas, se forem bem elaborados, podem resolver com bastante eficiência vários problemas de escalonamento de horários. Estas heurísticas podem ser aplicadas em diferentes tipos de problemas, entretanto, para cada um deles é necessário propor um método eficiente específico. Para o Problema de Escalonamento de Enfermeiros, as heurísticas precisam atender todas as restrições, aumentando o aproveitamento dos recursos envolvidos e tentando respeitar as opções preferenciais da equipe envolvida.

2.1 Heurísticas

Segundo Souza (2011), as heurísticas são técnicas geralmente inspiradas em métodos intuitivos que buscam uma boa solução em um custo computacional aceitável. Infelizmente estas técnicas não garantem a otimalidade da solução, tampouco asseguram se está próxima da solução ótima.

Existem dois tipos de heurísticas: heurísticas construtivas e heurísticas de refinamento. As heurísticas construtivas iniciam-se de uma "solução vazia". Ao escolher os elementos, esta faz uma análise destes elementos de acordo com a função de avaliação adotada. De modo repetitivo, este método estende a solução corrente até que uma solução completa é construída. Uma boa heurística construtiva é aquela que, além de gerar uma solução inicial válida, também constrói soluções iniciais de qualidade.

Dentre o espaço de heurísticas construtivas, podemos citar, entre outros, (Souza, 2011): heurística de Construção Gulosa, do Vizinho mais Próximo, de Bellmore e Nemhauser e de Inserção mais Barata.

A seguir, no algoritmo 2.1, temos um pseudocódigo ilustrando superficialmente o funcionamento de heurísticas construtivas. Nesta heurística existe a função de avaliação $f(.)$ e a solução s, inicialmente vazia. O método *seleciona* é o elemento determinante da heurística, pois através dele podemos rotular a heurística como uma das mencionadas no parágrafo anterior.

Algoritmo 2.1: Heurística Construtiva

Input: $f(.)$, s
$s \leftarrow \emptyset$;
Inicializa o conjunto C de elementos candidatos;
while $C \neq \emptyset$ **do**
  $c_{escolhido} \leftarrow seleciona\{c \in C\}$;
  $s \leftarrow s \cup \{c_{escolhido}\}$;
  Atualiza o conjunto C de elementos candidatos;
return s

Os mecanismos de busca local, previamente referenciados como heurísticas de refinamento, se constituem de técnicas baseadas em "soluções vizinhas". Uma vizinhança, segundo Almeida et al. (2008), é um conjunto de soluções (vizinhos) muito próximos da solução corrente, isto é, soluções com pouca diferenciação. Em outras palavras, seja S um espaço de busca de soluções e f a função objetivo a otimizar e dado que $s \in S$, o conjunto de vizinhos $N(s) \subseteq S$ contém um número determinado de soluções s'. Cada $s' \in N(s)$ é chamada de vizinho de s e são obtidos através de operações chamadas de movimento.

As técnicas de refinamento são geralmente executadas após o processo de obtenção da solução inicial, ou seja, depois que já se tenha obtido uma solução competente. De forma mais clara, estas heurísticas partem de uma solução inicial s_0 qualquer e percorrem, por meio de movimentos em cada iteração, da solução atual para uma vizinha conforme a estrutura de vizinhança adotada. O objetivo de todo este percurso é encontrar melhores resultados, tanto para problemas de maximização quanto para minimização.

O segredo dos métodos de refinamento reside na definição da vizinhança, ou seja, de uma solução s existente no espaço de soluções deve sempre ser possível atingir as outras soluções

utilizando um número finito de movimentos.

As heurísticas de refinamento mais presentes na literatura são: método da Descida, da Subida, de Primeira Melhora, e principalmente, Descida em Vizinhança Variável, entre outros.

O algoritmo 2.2 demonstra os passos mínimos necessários para se construir um mecanismo de busca local. O método *selecionaMelhor*, na grande maioria dos casos de uso, encontra um conjunto de vizinhos tendo características muito semelhantes com a solução corrente. Após encontrar um novo vizinho, a próximo etapa é avaliá-lo e determinar se o mesmo possui melhor valor na função objetivo. Caso se obtenha melhor valor, o vizinho se torna a nova solução corrente e o algoritmo continua por mais uma vez. Se o mesmo não acontecer, o algoritmo para e retorna o ótimo local.

Algoritmo 2.2: Heurística de Refinamento

Input: $f(.)$, s_0
$s \leftarrow s_0$;
repeat
  $s^* \leftarrow$ selecionaMelhor$(s' \mid s' \in N(s))$;
  if $f(s^*)$ *é melhor que* $f(s)$ **then**
    $s \leftarrow s^*$;
    $f(s) \leftarrow f(s^*)$;
until $f(s^*) \geq f(s)$;
return s

2.2 Metaheurísticas

Metaheurísticas são procedimentos de alto nível (Rosenberg et al., 2010) desenvolvidas para encontrar ou gerar soluções ainda melhores do que aquelas geradas por simples heurísticas. Estes métodos são usados, principalmente, quando não se incorpora todas as características do problema real devido à um conjunto incompleto de informações ou capacidade computacional limitada.

Contrariamente às heurísticas convencionais, Souza (2011) relata que as metaheurísticas são de natureza universal armadas de recursos para tentar escapulir de ótimos locais que ainda possuem valores distantes dos ótimos globais.

As metaheurísticas combinam métodos heurísticos básicos para explorar o espaço de busca eficientemente, e são utilizados para resolver uma variedade muito grande de problemas. Entretanto, não existe uma forma coerente para provar sua eficácia, e assim como acontece nas heurísticas, metaheurísticas também não garantem que soluções ótimas globais sejam encontradas para o problema em questão. Metaheurísticas geralmente encontram boas soluções com menor esforço computacional (Wikipedia, 2011) que métodos iterativos e simples heurísticas.

As principais propriedades de uma metaheurística são os mecanismos usados para fugir de emboscadas de ótimos locais. A princípio, elas se dividem em busca local e busca populacional.

Em metaheurísticas de trajetória, ou de busca local, a análise do espaço de busca é feita através de movimentos e/ou trocas executadas em cada iteração na solução atual, resultando em outra solução promitente. Dentre os vários métodos de trajetória existentes, destacam-se: *Multi-start* (Almeida et al., 2008), *Iterated Local Search* (Mine et al., 2010), Busca Tabu (Burke et al., 1998), *Simulated Annealing* (Hadwan e Ayob, 2010) e *Variable Neighborhood Search* (Burke et al., 2004).

Por sua vez, metaheurísticas de busca populacional trabalham com combinações de soluções a fim de que se obtenha uma nova "geração" de soluções. O ideal é que a cada geração haja também uma evolução, ou seja, formação de soluções ainda melhores. Em cada geração ocorre a avaliação de algumas soluções, este é o momento em que algumas delas são selecionadas para serem modificadas ou recombinadas e formarem um novo conjunto de soluções (população). Algoritmos Genéticos, *Scatter Search*, Colônia de Formigas e Algoritmos Meméticos são exemplos de metaheurísticas de busca populacional.

2.2.1 Multi-Start

De acordo com Almeida et al. (2008) e Souza (2011), o algoritmo *Multi-Start* baseia-se em fazer amostragens do espaço de busca. Uma forma de se obter diversificações é reiniciar o procedimento de melhora a partir de uma nova solução visto que uma região foi completamente explorada sem obter grandes melhoras. Sua principal vantagem é sua fácil implementação.

Cada iteração produz uma solução e o algoritmo só termina quando o critério de parada adotado é satisfeito. No final, a saída do algoritmo é a solução com melhor resultado dentre as demais.

Algoritmo 2.3: Multi-Start

```
Input: f(.) , N(.)
f* ← ∞ {Valor associado à s*};
while Critério de parada não é atendido do
    s ← heurísticaConstrutiva();
    s ← heurísticaRefinamento(s);
    if f(s) < f(s*) then
        s* ← s;
        f* ← f(s);
s ← s*;
return s
```

No algoritmo 2.3 as diversificações são obtidas através da geração de soluções aleatórias, resultando em possibilidades de escapar de ótimos locais. Métodos *Multi-Start* possuem dois momentos: o primeiro é aquele onde a solução é gerada e o segundo momento é onde a solução

é refinada. Na grande maioria das vezes a primeira etapa é executada por um algoritmo heurístico construtivo e na segunda etapa tem sido cada vez mais comum elaborar metaheurísticas complexas que podem ou não aprimorar os resultados.

2.2.2 Variable Neighborhood Search

A Busca em Vizinhança Variável (*Variable Neighborhood Search*, VNS) é um método de refinamento que explora o espaço de soluções por meio de trocas de estruturas de vizinhança, ou seja, a forma de se obter novos vizinhos é alterada a partir do momento que o problema não obtém melhores resultados com a vizinhança corrente.

Podemos constatar que o VNS (Souza, 2011) não segue uma trajetória constante de vizinhos. Na verdade, ele explora, de forma sucessiva, vizinhanças mais "remotas" em relação a solução corrente e assume uma nova solução se, e somente se, um movimento de melhora é realizado.

Dado que existe um conjunto de estruturas de vizinhança E, este algoritmo parte de uma solução inicial s_0 e a cada iteração obtém-se um vizinho s' na vizinhança N^e da solução corrente. Este vizinho é então submetido a uma busca local. Se a nova solução s'' for melhor que a solução corrente s, a busca continua de s'' recomeçando da primeira estrutura de vizinhança, caso contrário, a busca continua a partir de outro tipo de estrutura de vizinhança.fixa

Algoritmo 2.4: Variable Neighborhood Search

```
Input: E, s_0
s ← s_0;
while Critério de parada não é atendido do
    for e ∈ E do
        s' ←pertubação(s, N^e);
        s'' ←buscaLocal(s');
        if f(s'') < f(s) then
            s ← s'';
            Reiniciar laço de repetição for;
return s
```

Variable Neighborhood Descent

A Descida em Vizinhança Variável (*Variable Neighborhood Descent, VND*) é uma estratégia de busca local (Hu e Raidl, 2006) e é geralmente utilizada como um método subordinado ao VNS, *Multi-Start* e outras metaheurísticas. A ideia por trás do VND é basicamente explorar soluções por diferentes estruturas de vizinhança.

Iniciando pela primeira estrutura de vizinhança N^1, o VND executa a busca local, escolhendo melhores vizinhos, até que nenhuma melhora seja mais possível. A partir deste ótimo

local, a busca continua com a estrutura de vizinhança N^2. Se alguma melhora é encontrada com esta estrutura, o VND reinicia a busca utilizando a estrutura de vizinhança N^1 novamente. Caso contrário, a nova estrutura será N^3.

Algoritmo 2.5: Variable Neighborhood Descent

```
Input: E, s
while Critério de parada não é atendido do
    for e ∈ E do
        s' ←melhorVizinho(s, N^e);
        if f(s') < f(s) then
            s ← s';
            Reiniciar laço de repetição for;
return s
```

2.2.3 Relaxation Induced Neighborhood Search

Em muitos problemas de otimização há mais de um tipo de solução: Uma delas é a solução incumbente, ou seja, uma solução inteira factível que atende aos requisitos de integralidade, mas não é a melhor. Dentre várias maneiras de se obter uma solução inteira rapidamente, a utilizaçâo de algoritmos construtivos é bastante utilizada. Todavia, as soluções geradas por estes algoritmos são geralmente ruins no quesito satisfação de restrições fracas.

A outra solução é fracionária, resultante de relaxações, e não satisfaz a integralidade das variáveis. Entretanto, seu valor na função objetivo é sempre melhor que o da solução inteira. A solução fracionária é obtida pela relaxação (Causmaecker e Berghe, 2003) de todas as variáveis e, consequentemente, não é uma solução válida pois não é uma solução inteira.

Danna et al. (2005) perceberam que, enquanto algumas variáveis possuem valores totalmente diferentes na solução inteira e na solução relaxada, muitas variáveis possuem os mesmos valores. O método *Relaxation Induced Neighborhood Search* induz que a instanciação das variáveis idênticas em ambas soluções forma uma solução parcial com grande possibilidade de alcançar tanto a integralidade das variáveis quanto um bom valor objetivo. Daí, propuseram um algoritmo simples, baseado nos seguintes passos:

- Fixar as variáveis que possuem os mesmos valores na solução incumbente e na solução fracionária relaxada.
- Estabelecer um corte baseado no valor da função objetivo da solução incumbente.
- Resolver o subproblema restante, ou seja, trabalhar com as variáveis que não foram fixadas.

O subproblema, gerado pelas variáveis que não foram fixadas, também é difícil de resolver. Por isso, quando sua exploração termina e caso seja encontrada uma solução inteira melhor do que a solução incumbente, esta última então é atualizada com os novos valores.

2.3 Programação com Multi-Processos

O modelo de programação com multi-processos, também conhecido como *multithreading*, permite que várias *threads* existam dentro de um único processo. As *threads* compartilham os recursos de um processo, como o endereço de memória, sendo também capazes de processá-los independentemente.

Existem algumas APIs que facilitam bastante a utilização de múltiplas *threads*. Em C/C++ as principais são **OpenMP** (Dagum e Menon, 1998) e **Pthreads** (Sakamoto et al., 1999). Ambas são APIs que suportam programação multi-processada com compartilhamento de memória e basicamente se consistem de um conjunto de diretivas de compilação e rotinas que influenciam a execução.

Para melhorar o desempenho de suas heurísticas, Ribas et al. (2006) expõe um algoritmo paralelo alicerçado na metaheurística *Iterated Local Search* (ILS) para resolver um problema que carece de velocidade na geração de uma solução. Eles propuseram uma abordagem desenvolvida em uma arquitetura com vários núcleos de processamento e foram observadas melhorias na qualidade da solução.

Hansen et al. (2008) propuseram diversas versões do método VNS, e uma delas é definida como *Parallel VNS*. Em seu artigo, eles afirmam que uma das maneiras mais eficientes de se obter bons resultados com o VNS paralelo é atribuir diferentes vizinhanças para cada núcleo de processamento e interromper a execução assim que um progresso é obtido na solução.

Capítulo 3

Descrição do Problema Abordado

Os dados de entrada para o Problema de Escalonamento de Enfermeiros (*Nurse Rostering Problem*, NRP) foram disponibilizados pela INRC e uma descrição mais detalhada do problema foi escrita por Haspeslagh et al. (2010). A descrição fornece detalhes fundamentais para uma ampla compreensão do problema, pois facilitam a interpretação dos arquivos de entrada. Estes arquivos especificam os detalhes de cada contrato, a aptidão de cada enfermeiro, dentre outras propriedades de cada problema.

Na grande maioria dos problemas, o trabalho da equipe de enfermagem é distribuído em quatro turnos – manhã, tarde, noite e madrugada. Algumas instâncias possuem um turno adicional exclusivo ao enfermeiro chefe.

Todos os meses cada enfermeiro é informado do seu horário em um planejamento de 28 dias, isto é, 4 semanas. Nestes dias, cada turno é preenchido por um único enfermeiro, entretanto há um número exato de enfermeiros a serem alocados para cada turno diariamente. Dependendo do dia e do horário, existe uma demanda diferente a ser garantida a fim de se assegurar a qualidade do serviço ao longo do tempo.

3.1 Restrições

Existem basicamente dois tipos de restrições que devem ser levadas em consideração quando se trata da atribuição de turnos aos enfermeiros. São elas:

- **Restrições Fortes:** Do inglês, *Hard Constraints*, são restrições que devem ser satisfeitas de qualquer maneira e, por isso, estão diretamente ligadas ao sucesso ou fracasso de uma solução;
- **Restrições Fracas:** Também do inglês, *Soft Constraints*, é um conjunto de restrições que devem ser respeitadas quando possível, contudo há grandes chances que nem todas serão atendidas.

Uma solução factível é uma solução que não viola nenhuma restrição forte. A especificação da INRC define duas restrições fortes:

1. Um enfermeiro pode trabalhar no máximo um turno por dia;
2. Existe uma demanda exata de cada turno por dia que deve ser satisfeita durante todo o planejamento.

No problema descrito pela INRC, cada enfermeiro trabalha rigorosamente de acordo com seu contrato. Nele estão definidas a grande maioria das restrições fracas. São elas:

1. mínimo/máximo número de turnos atribuídos aos enfermeiros;
2. mínimo/máximo número de dias de trabalho consecutivos;
3. mínimo/máximo número de dias de folga consecutivos;
4. máximo número de fins-de-semana trabalhados consecutivamente;
5. máximo número de fins-de-semana trabalhados em um período de quatro semanas;
6. mínimo número de dias de folga após uma série de turnos noturnos;
7. nenhum turno noturno antes de um final de semana sem trabalhar;
8. fim-de-semana completo: enfermeiro precisa trabalhar ambos os dias do final de semana, senão é registrada uma penalidade;
9. fim-de-semana idêntico: enfermeiro tem de trabalhar em turnos idênticos no mesmo fim-de-semana para que não seja caracterizada uma violação;
10. trabalha dia sim/não: requisição de um enfermeiro para trabalhar ou não em determinado dia. Se a restrição não for respeitada, compromete a qualidade da solução;
11. trabalha turno sim/não: solicitação de um enfermeiro para trabalhar ou não no turno t em um dia específico. Também pode comprometer a solução se não for satisfeita;
12. sequência indesejada: sequência de atribuições de dias ou turnos que vai contra as preferências de um enfermeiro baseado em seu contrato;
13. enfermeiros sub-qualificados: um enfermeiro com pouco conhecimento não pode trabalhar em um turno que exija maior destreza e capacidade técnica.

Algumas sequências indesejadas podem ocorrer em qualquer dia do horizonte de planejamento. Por exemplo:

- o turno noturno em uma sexta-feira não pode ser alocado a um enfermeiro se o mesmo terá o fim-de-semana livre;
- se um enfermeiro trabalha no fim-de-semana, ele também deve trabalhar na sexta-feira;
- um enfermeiro provavelmente não gostaria de ser alocado em um turno noturno tendo no dia seguinte uma atribuição no turno da manhã.

São as restrições fracas que determinam a qualidade de uma solução. Muitas restrições fracas violadas indicam que a solução não é de boa qualidade e isso não é difícil de acontecer em problemas com inúmeras restrições.

3.2 Formulação

Esta formulação, elaborada por meu orientador, busca representar de forma fiel as relações existentes entre a função objetivo, as variáveis de decisão e as restrições. A modelagem auxilia na descrição do problema, pois abrange as restrições que compõem as instâncias da INRC.

3.2.1 Dados de entrada

Aqui temos a lista dos tipos de dados que caracterizam o problema:

N conjunto de enfermeiros

C conjunto de contratos

$\tilde{c}_n$ contrato do enfermeiro n

S conjunto de turnos

$\tilde{S}$ conjunto de turnos noturnos

D conjunto de dias com elementos sequencialmente enumerados a partir de 1

Π conjunto de todos os pares ordenados $(d_1, d_2) \in D \times D : d_1 \leq d_2$ representando janelas no período de planejamento

$\tilde{W}_c$ conjunto dos finais-de-semana no horizonte de planejamento de acordo com a definição de finais de semana no contrato c, com elementos enumerados de 1 até $\tilde{w}_c$

$\tilde{D}_{ic}$ conjunto de dias no i-ésimo fim-de-semana do contrato c

$\tilde{r}_{ds}$ número de enfermeiros necessários no dia d e turno s

$\acute{P}_c$ conjunto de padrões de turnos de trabalho indesejáveis para o contrato c

$\hat{P}_c$ conjunto de padrões de dias de trabalho indesejáveis para o contrato c

São os contratos que determinam a configuração das restrições, pois são eles que atribuem um peso quando há violação de uma restrição fraca.

As restrições fracas são divididas em dois grupos: *Restrições Fracas de Faixa* (RFF) e *Restrições Fracas Binárias* (RFB). As RFF são aquelas que estabelecem um intervalo de valores inteiros válidos. Valores fora desta faixa são penalizados proporcionalmente à distância do valor aceitável mais próximo. RFB possuem natureza binária: são ou não são satisfeitas.

Para diminuir a superabundância de informação nas seções que seguem, cada restrição está associada a um índice da tabela 3.1. Estes índices informam o limite mínimo e máximo de uma dada RFF i e um contrato c, respectivamente denotados por $\underline{\gamma}_c^i$ e $\overline{\gamma}_c^i$, enquanto a penalização destas restrições são expressas por $\underline{\omega}_c^i$ e $\overline{\omega}_c^i$. Já em RFB, o peso da penalização é definido por ω_c^i.

	Restrições Fracas de Faixa
1	número total de atribuições
2	dias de trabalho consecutivos
3	dias de folga consecutivos
4	fins-de-semana trabalhados em um período de quatro semanas
5	fins-de-semana trabalhados consecutivos
6	número de dias de folga após uma série de turnos noturnos

	Restrições Fracas Binárias
7	fim-de-semana completo
8	nenhum turno noturno antes de um final de semana sem trabalhar
9	fim-de-semana idêntico
10	enfermeiros sub-qualificados
11	padrão de turnos de trabalho indesejáveis
12	padrão de dias de trabalho indesejáveis
13	turnos e dias indesejáveis

Tabela 3.1: Índices das restrições fracas de faixa e binárias

As variáveis de folga, denotadas por $\underline{\alpha}_n^i$,$\overline{\alpha}_n^i$ e α_n^i, também estão associadas ao índices da tabela 3.1. Elas correspondem à violação dos limites inferiores e superiores das RFF e RFB para um enfermeiro n. Índices adicionais às variáveis α_n^i podem ser utilizados, por exemplo: α_{nj}^7 está diretamente relacionada à violação da restrição correspondente ao final de semana completo para o enfermeiro n no j-ésimo fim-de-semana.

3.2.2 Variáveis de Decisão

Um dos trabalhos mais árduos é determinar as variáveis do problema e, principalmente, decidir qual valor cada variável deve receber. Neste problema, a variável de decisão principal é:

$$x_{nsd} = \begin{cases} 1 & \text{se o enfermeiro } n \text{ está alocado no turno } s \text{ no dia } d \\ 0 & \text{caso contrário} \end{cases}$$

E ainda foram utilizadas mais quatro variáveis auxiliares:

$$y_{ni} = \begin{cases} 1 & \text{se o enfermeiro } n \text{ trabalha no fim-de-semana } i \\ 0 & \text{caso contrário} \end{cases}$$

$$w_{nd_1d_2} = \begin{cases} 1 & \text{se o enfermeiro } n \text{ trabalha do dia } d_1 \text{ até o dia } d_2 \\ 0 & \text{caso contrário} \end{cases}$$

$$r_{nd_1d_2} = \begin{cases} 1 & \text{se o enfermeiro } n \text{ folga do dia } d_1 \text{ até o dia } d_2 \\ 0 & \text{caso contrário} \end{cases}$$

$$z_{ni_1i_2} = \begin{cases} 1 & \text{se o enfermeiro } n \text{ trabalha do fim-de-semana } i_1 \text{ até o fim-de-semana } i_2 \\ 0 & \text{caso contrário} \end{cases}$$

3.2.3 Função Objetivo

A função objetivo é uma equação, recheada de restrições, a ser otimizada. Na função objetivo do problema especificado pela INRC denota-se como $w_{nd_1d_2}$ e $r_{nd_1d_2}$, respectivamente, as variáveis referentes à seleção de uma janela de trabalho ou folga de um conjunto Π.

O peso de todas as violações que ocorrem em um período de trabalho ou folga para enfermeiros de contrato c, iniciando no dia d_1 e terminando no dia d_2, são denotadas por $\sigma_{cd_1d_2}$ e τ_{cd_2}. O cálculo deste peso é obtido de acordo com um parâmetro de entrada que especifica a dimensão de uma violação de certa restrição multiplicado pelo número de violações.

As restrições fracas 10 e 13 são diretamente penalizadas na variável x_{nsd}, com os coeficientes ν_{nsd}. De semelhante modo, a restrição fraca 5 é penalizada nas variáveis $z_{ni_1i_2}$ com coeficientes ψni_1i_2.

Minimize:

$$\sum_{n\in N}\left[\begin{array}{l}\sum\limits_{(d_1d_2)\in\Pi}(\sigma_{\tilde{c}_n d_1 d_2} w_{n d_1 d_2} + \tau_{\tilde{c}_n d_1 d_2} r_{n d_1 d_2}) + \\ \sum\limits_{s\in S}\sum\limits_{d\in D}\nu_{nsd}x_{nsd} + \overline{\omega}^1_{\tilde{c}_n}\overline{\alpha}^1_n + \underline{\omega}^1_{\tilde{c}_n}\underline{\alpha}^1_n + \\ \sum\limits_{i\in\{1\ldots\tilde{w}_c\}}(\omega^4_{\tilde{c}_n}\alpha^4_{ni} + \underline{\omega}^8_{\tilde{c}_n}\underline{\alpha}^8_{ni} + \underline{\omega}^9_{\tilde{c}_n}\underline{\alpha}^9_{ni}) + \\ \sum\limits_{i_1,i_2\in\tilde{W}_{\tilde{c}_n}:i_2\geq i_1}\psi_{ni_1i_2}z_{ni_1i_2} + \\ \sum\limits_{d\in D}\underline{\omega}^6_{\tilde{c}_n}\underline{\alpha}^6_n + \sum\limits_{\acute{p}\in\acute{P}_{\tilde{c}_n}}\alpha^{11}_{n\acute{p}} + \sum_{\hat{p}\in\hat{P}_{\tilde{c}_n}}\alpha^{12}_{n\hat{p}}\end{array}\right]$$

3.2.4 Restrições

As restrições 3.1 e 3.2 modelam as restrições fortes do problema. Restrições 3.3 e 3.4 vinculam as ativações das variáveis x juntamento com a ativação das variáveis y. Das restrições 3.5 até 3.9 são atestados que cada ativação da janela de trabalho (w) é seguido pela ativação de uma variável r e vice-versa, estabelecendo períodos de trabalho e folga contínuos com diferentes tamanhos.

O restaste das restrições são somente restrições fracas, implicando que não há a obrigatoriedade delas serem satisfeitas. Nas formulações, as restrições são interpretadas como fraca desde que seja adicionado uma variável de folga (α) que serão punidas na função objetivo quando ativadas.

As restrições 3.10 determinam o número máximo e mínimo de dias trabalhados no planejamento, enquanto 3.11 especificam o número máximo de finais de semana trabalhados em 4 semanas. As restrições 3.12 garantem que haja um limite no número máximo de finais de semana trabalhados consecutivamente. Uma sequência de dias de folga após trabalhar uma série de turnos noturnos são determinadas pelas restrições 3.13. Restrições 3.14 asseguram que uma enfermeira não vai trabalhar a noite em uma sexta-feira antecedendo um fim-de-semana com folga. Caso um enfermeiro trabalhe um fim-de-semana, foram criadas as restrições 3.15 e 3.16 para garantir que os turnos sejam os mesmos. Os padrões indesejados que, preferencialmente, devem ser evitados são modelados pelas restrições 3.17 e 3.18.

$$\sum_{n\in N} x_{nsd} \;=\; \tilde{r}_{ds}\ \forall d\in D, s\in S \tag{3.1}$$

$$\sum_{s\in S} x_{nsd} \;\leq\; 1\ \forall n\in N, d\in D \tag{3.2}$$

$$y_{ni} \;\geq\; \sum_{s\in S} x_{nsd}\ \forall n\in N, i\in \tilde{W}_{\tilde{c}_n}, d\in \tilde{D}_{i\tilde{c}_n} \tag{3.3}$$

$$y_{ni} \;\leq \sum_{s\in S, d\in \tilde{D}_{i\tilde{c}_n}} x_{nsd}\ \forall n\in N, i\in \tilde{W}_{\tilde{c}_n} \tag{3.4}$$

$$\sum_{s\in S} x_{nsd} \;= \sum_{(d_1,d_2)\in\Pi\,:\,d\in\{d_1,\ldots,d_2\}} w_{nd_1d_2}\ \forall n\in N, d\in D \tag{3.5}$$

$$\sum_{s\in S} x_{nsd} \;=\; 1-\Big(\sum_{(d_1,d_2)\in\Pi\,:\,d\in\{d_1,\ldots,d_2\}} r_{nd_1d_2}\Big)\ \forall n\in N, d\in D \tag{3.6}$$

$$\sum_{(d_1,d_2)\in\Pi\,:\,d\in\{d_1\ldots d_2\}} (w_{nd_1d_2}+r_{nd_1d_2}) \;=\; 1\ \forall n\in N, d\in D \tag{3.7}$$

$$\sum_{d'\in\{1,\ldots,d\}} w_{nd'd} + \sum_{d''\in D: d''\,\geq\, d+1} w_{n,d+1,d''} \;\leq\; 1\ \forall n\in N, d\in D \tag{3.8}$$

$$\sum_{d'\in\{1,\ldots,d\}} r_{nd'd} + \sum_{d''\in D: d''\geq d+1} r_{n,d+1,d''} \;\leq\; 1\ \forall n\in N, d\in D \tag{3.9}$$

$$\underline{\gamma}^1_{\tilde{c}_n} - \underline{\alpha}^1_n \;\leq \sum_{s\in S, d\in D} x_{nsd} \;\leq\; \overline{\gamma}^1_{\tilde{c}_n} + \overline{\alpha}^1_n\ \forall n\in N \tag{3.10}$$

$$\sum_{i'\in\{i,\ldots,i+3\}} y_{ni'} \;\leq\; \overline{\gamma}^4_{\tilde{c}_n} + \overline{\alpha}^4_{ni}\ \forall n\in N, i\in\{1,\ldots,\tilde{w}_{\tilde{c}_n}-3\} \tag{3.11}$$

$$\sum_{i'\in\{i,\ldots,i+\overline{\gamma}^5_{\tilde{c}(n)}\}} y_{ni'} \;\leq\; \overline{\gamma}^5_{\tilde{c}_n} + \overline{\alpha}^5_{ni}\ \forall n\in N, i\in\{1,\ldots,\tilde{w}_{\tilde{c}_n}-\overline{\gamma}^5_{\tilde{c}_n}\} \tag{3.12}$$

$$\sum_{s'\in\tilde{S}} \underline{\gamma}^6_{\tilde{c}_n} x_{ns'd} + \sum_{s\in S\setminus\tilde{S}, d'\in\{d+1\ldots,d+\underline{\gamma}^6_{\tilde{c}(n)}\}} x_{nsd'} \;\leq\; \underline{\gamma}^6_{\tilde{c}_n} + \underline{\alpha}^6_{nd}\ \forall n\in N, d\in D: d\leq |D| - \underline{\gamma}^6_{\tilde{c}(n)} \tag{3.13}$$

$$\sum_{s\in\tilde{S}}\ \sum_{d\in\tilde{W}_{i\tilde{c}_n}: d\geq 2\wedge d\leq d'\,\forall d'\in\tilde{W}_{i\tilde{c}_n}} x_{n,s,d-1} + y_{ni} \;\leq\; 1+\alpha^8_{ni}\ \forall n\in N, i\in\tilde{W}_{i\tilde{c}_n} \tag{3.14}$$

$$\alpha^9_n \;\geq\; x_{nsd_1} - x_{nsd_2}\ \forall n\in N, s\in S, i\in\tilde{W}_{\tilde{c}_n}, d_1,d_2\in\tilde{D}_{i\tilde{c}(n)}:\, d_1<d_2 \tag{3.15}$$

$$\alpha^9_n \;\geq\; x_{nsd_2} - x_{nsd_1}\ \forall n\in N, s\in S, i\in\tilde{W}_{\tilde{c}_n}, d_1,d_2\in\tilde{D}_{i\tilde{c}(n)}:\, d_1<d_2 \tag{3.16}$$

$$\sum_{j\in\{1,\ldots,\tilde{s}(\acute{p})\}} x_{n,\acute{p}[1],d+j-1} \;\leq\; \tilde{s}(\acute{p}) + \alpha^{11}_{n\acute{p}}\ \forall n\in N, \acute{p}\in\acute{P}_{\tilde{c}_n}, d\in\{1,\ldots,|D|-\tilde{s}(\acute{p})+1\} \tag{3.17}$$

$$\sum_{s\in S}\ \sum_{j\in\{1,\ldots,\tilde{s}(\hat{p}):\hat{p}[j]\geq 1\}} x_{n,s,\hat{p}[j]} + \sum_{j\in\{1,\ldots,\tilde{s}(\hat{p}):\hat{p}[j]\leq -1\}} (1-\sum_{s\in S} x_{n,s,-\hat{p}[j]}) \;\leq\; \tilde{s}(\hat{p}) + \alpha^{12}_n\ \forall n\in N, \hat{p}\in\hat{P}_{\tilde{c}_n} \tag{3.18}$$

Capítulo 4

Desenvolvimento

O presente capítulo trata dos detalhes de implementação do Nurse Rostering Problem Solver (NRPS), um sistema desenvolvido para gerar boas escalas trabalhistas para equipes de enfermagem.

O NRPS foi desenvolvido utilizando a linguagem C/C++. Tal linguagem foi escolhida devido à existência da API da biblioteca COIN-OR Open Solver Interface (OSI) que permite criar aplicações usando diferentes resolvedores (Saltzman et al., 2004). A OSI é uma classe abstrata para resolvedores de programação linear genéricos e possui subclasses específicas para vários resolvedores(CBC, CLP, CPLEX, GLPK, dentre outros).

O resolvedor de programação inteira utilizado se chama COIN-OR Branch-and-Cut (CBC), é *open-source* e foi escrito em C++. O CBC foi inicialmente projetado para ser utilizado como uma biblioteca (Forrest e Lougee-Heimer, 2005), entretanto, também é possível usá-lo como a aplicação em si, pois funciona muito bem como resolvedor independente. O COIN-OR Branch-and-Cut faz parte da iniciativa Computational Infrastructure for Operations Research (COIN-OR) e vincula-se ao COIN-OR CLP, um outro resolvedor de programação inteira, para resolver subproblemas.

O CBC foi um recurso essencial para o projeto, ele foi utilizado juntamente com outras técnicas de programação. São apresentadas nas seções que se seguem os principais métodos e escolhas tomadas, baseadas na pesquisa bibliográfica apresentada no capítulo anterior e na experiência de meu orientador.

Primeiramente, na seção 4.1, é apresentado de forma clara como uma solução para o problema dos enfermeiros é representada. O algoritmo e métodos propostos para gerar uma solução inicial são mostrados na seção 4.2. Já na seção 4.3 são descritas as estruturas de vizinhança e as fixações utilizadas para gerar subproblemas e facilitar a exploração do espaço de soluções. A seção 4.4 apresenta o algoritmo proposto para refinar a solução inicial e gerar boas solução. Este algoritmo é composto pelas estruturas de vizinhança, responsáveis por uma grande melhora das soluções iniciais.

4.1 Representação de uma Solução

Para representar o problema de escalonamento de enfermeiros, foi adotada uma matriz $M_{|N|\times|D|}$ tendo as linhas simbolizando os enfermeiros e as colunas simbolizando os dias. Tal notação foi adotada, pois $|N|$ é o número de enfermeiros disponíveis e $|D|$ representa o número de dias do planejamento em que o escalonamento está sendo executado.

Cada célula $m_{i,j}$ da matriz pode ser preenchida por um turno $t \in T$. O número de turnos a serem executados por cada enfermeiro varia de acordo com a capacidade técnica e a formação de cada um deles.

A escala trabalhista, tabela 4.1, demonstra uma possível solução, com um pequeno número de enfermeiros e dias, para o problema de escalonamento em questão. Note que o traços "−" representam dias de folga, **D** (*days*) são turnos executados durante a manhã, **E** (*evening*) são os turno da tarde, **L** (*late*) são turnos noturnos, enquanto **N** (*night*) são aqueles turnos alocados durante a madrugada.

N_i/D	**Sun**	Mon	Tue	Wed	Thur	Fri	**Sat**	**Sun**	Mon	Tue
N_1	N	L	L	L	N	N	L	L	L	N
N_2	-	L	-	L	L	-	L	D	-	D
N_3	-	-	L	-	D	D	-	-	-	E
N_4	-	-	L	L	E	-	-	D	D	D
N_5	-	N	-	D	D	D	E	-	-	N
N_6	D	D	-	-	-	E	E	-	-	E
N_7	D	E	E	L	-	-	L	N	N	-
N_8	N	E	-	-	E	-	N	-	-	E
N_n										

Tabela 4.1: Exemplo reduzido de uma escala trabalhista

Para facilitar o processamento computacional, principalmente se houver turnos com nomenclaturas longas, os turnos podem ser representados por algarismos. Assim, ao invés da representação adotada na tabela 4.1, que permite o usuário entender com muito mais clareza e rapidez, são utilizados valores numéricos para representar cada turno. Veja na tabela 4.2 como ficam a distribuição dos turnos.

Com a representação adotada, cada número representa exclusivamente um único turno, isto é, o número 0 representa os dias de folga, o número 1 substitui o turno **D**, o número 2 está no lugar do turno **E** e o número 3 está para o turno **L** assim como o número 4 está para o turno **N**.

Esta representação, seja preenchida por siglas ou algarismos, só pode ser adotada quando existe no máximo um turno alocado para cada enfermeiro no mesmo dia. Como o Capítulo 3 define o problema impedindo que qualquer enfermeiro trabalhe dois turnos ou mais no mesmo dia, esta representação foi devidamente escolhida.

N_i/D	**Sun**	Mon	Tue	Wed	Thur	Fri	**Sat**	**Sun**	Mon	Tue
N_1	4	3	3	3	4	4	3	3	3	4
N_2	0	3	0	3	3	0	3	1	0	1
N_3	0	0	3	0	1	1	0	0	0	2
N_4	0	0	3	3	2	0	0	1	1	1
N_5	0	4	0	1	1	1	2	0	0	4
N_6	1	1	0	0	0	2	2	0	0	2
N_7	1	2	2	3	0	0	3	4	4	0
N_8	4	2	0	0	2	0	4	0	0	2
N_n										

Tabela 4.2: Exemplo de escala trabalhista reduzida representada por algarismos

4.2 Solução Inicial

A princípio, o mais importante é que todas as restrições fortes sejam atendidas, o que já nos fornece uma solução válida e provavelmente bem distante de uma solução ótima.

Como ponto inicial do projeto, foi implementado um algoritmo construtivo guloso para a elaboração inicial das escalas. Ao construir as escalas ocorre a verificação da demanda diária, ou seja, o algoritmo é executado enquanto houver turnos a serem ocupados pelos enfermeiros.

O método 4.1 constrói uma matriz de alocação $M_{|N|\times|D|}$, inicializando todas as células m_{ij} da tabela com dias de folga (*days off*). Esta matriz totalmente preenchida representa a solução inicial e neste método é identificada como sendo a solução s.

Algoritmo 4.1: GeraSoluçãoInicial

```
Input: f(.)
s ← ∅;
for d ∈ D do
    while demandaRestante(d) > 0 do
        t ← turnoNaoAlocado(d);
        C ← enfermeirosCandidatos(d, t);
        if |C| > 0 then
            n ← selecionaAleatoriamente{c ∈ C};
            s_{n,d} ← t;
        else
            C ← enfermeirosComPenalidade(d, t);
            n ← selecionaMenorPenalidade{c ∈ C};
            s_{n,d} ← t;
s ← equilibraCargaHoraria(s);
return s
```

Este algoritmo percorre de forma crescente todos os dias do planejamento e só avança para o dia seguinte quando toda a demanda do dia corrente for suprida. Para suprir a demanda é

criado um conjunto com a seleção dos enfermeiros aptos e que ainda não foram alocados no dia d, ou seja, enfermeiros que ainda estão livres. Os enfermeiros deste conjunto são selecionados aleatoriamente e são alocados para trabalhar no turno t no dia d.

Caso não haja mais enfermeiros candidatos disponíveis para o turno e o dia atual, é criado um conjunto com enfermeiros tecnicamente capazes de trabalhar no turno t, mas se houver alguma atribuição também haverá penalidades. Deste conjunto são escolhidos aqueles enfermeiros que pioram o mínimo possível o valor da solução para receberem as atribuições.

Balanceamento da carga-horária

Após gerar a solução inicial, é iniciada a etapa de trocas que busca equilibrar o número de turnos trabalhados por cada enfermeiro. Existem enfermeiros que trabalham poucos turnos durante o período e existem outros que trabalham em um número muito maior de turnos dentro do mesmo período. Estas trocas (*swaps*) não podem modificar turnos que estejam em dias diferentes e ocorrem preferencialmente entre enfermeiros com poucas atribuições e enfermeiros com muitas atribuições.

Estes movimentos (veja a tabela 4.3) não buscam melhorar a solução, na verdade, a única verificação realizada é para garantir que após cada *swap*, a solução continue válida. Estes *swaps* permitem que mais movimentos de troca possam ser feitos durante a resolução dos subproblemas, aumentando ainda mais a probabilidade de se alcançar uma melhor solução.

Movimento Adiciona/Remove Turno

Este movimento consiste em dar folga para um dado enfermeiro n_1 em um dia d que está trabalhando no turno t e escolher outro enfermeiro, n_2, para trabalhar em seu lugar. O movimento é feito entre turnos do mesmo dia e o mesmo não pode ser realizado se n_2 não for qualificado o suficiente para trabalhar no turno t.

N_i/D	**Sun**	Mon	Tue	Wed	Thur	Fri	**Sat**	**Sun**	Mon	Tue
N_1	**4**	3	3	3	4	4	3	3	**3**	4
N_2	**0**	3	0	3	3	0	3	1	**0**	1
N_n										
N_1	**0**	3	3	3	4	4	3	3	**0**	4
N_2	**4**	3	0	3	3	0	3	1	**3**	1
N_n										

Tabela 4.3: Exemplo de aplicação do movimento Adiciona/Remove turno

A tabela 4.3 exemplifica o funcionamento do movimento de Adiciona/Remove turno. Note que há duas trocas entre os enfermeiros N_1 e N_2, uma no domingo (*Sunday*) e outra numa segunda-feira (*Monday*). Também é possível perceber que, inicialmente, o enfermeiro N_1 não

possui nenhuma folga, enquanto N_2 possui quatro dias livres. Após o balanceamento, ambos os enfermeiros possuem dois dias livres.

Neste movimento uma célula m_{ik} da matriz de solução tem seu valor alterado de algum valor x para 0 (dia de folga) enquanto uma outra célula m_{jk} tem seu valor alterado de 0 para x tal que $1 \leq x \leq |T|$.

4.3 Estrutura de Vizinhança

O problema de escalonamento de enfermeiros possui um número muito grande de restrições, e com isso muitos métodos já se mostraram ineficazes (Qu e He, 2009). Existem vários fatores que dificultam a solução destes problemas, como a capacidade técnica da equipe, a necessidade de equilibrar a carga de trabalho entre os enfermeiros, e também considerar as preferências pessoais de cada um deles.

Recentemente muitos pesquisadores tem investido seus esforços na resolução de subproblemas. Subproblemas são geralmente utilizados para acelerar a execução da busca local, pois parte do problema original é "descartado", isto é, o problema a ser resolvido torna-se temporariamente menor.

Neste trabalho os subproblemas são obtidos através de fixações de variáveis. Estas fixações dividem todo o problema em duas partes: uma parte possui somente variáveis fixadas e na outra todas as variáveis continuam livres.

As variáveis que são fixadas não permitem que seus valores sejam alterados. Isto significa que resolvedores, como o CBC, ou outros métodos de refinamento só serão aplicados sobre as variáveis livres. As variáveis que permaneceram livres no problema original são exatamente as variáveis que constituem o subproblema. Para determinar quais variáveis permanecem livres são aplicadas estruturas de vizinhança sobre uma solução pré-existente.

Cada estrutura de vizinhança agrupa subproblemas de diferentes maneiras (a forma como estes subproblemas são arranjados são descritas nos próximos tópicos). Todos estes agrupamentos são limitados pelo parâmetro p da vizinhança. Este parâmetro determina o tamanho exato do subproblema e as estruturas de vizinhança garantem que tal seja sempre menor que o problema original.

Veja no algoritmo 4.2 que vários subproblemas, de tamanho p, são elaborados e imediatamente armazenados em E. Em cada iteração, as variáveis do problema original que não estão contidas no subproblema e são fixadas, e todas aquelas que pertencem a e são mantidas livres para que o processo de otimização do CBC tenha influência sobre elas. Quando há alguma melhora na solução o parâmetro p retorna para seu valor inicial e todo o processo, desde a geração de subproblemas, é reiniciado mantendo a nova solução como solução corrente. Se não for constatada nenhuma melhora depois de percorrer todo o conjunto E, o algoritmo aumenta o valor de p e repete todo o procedimento.

Algoritmo 4.2: Estrutura de Vizinhança

```
Input: f(.), s, p_0
p ← p_0;
while p < tamanho máximo da estrutura de vizinhança do
    E ← geraSubproblemas(p);
    for e ∈ E do
        s' ← fixaVariáveis(s, e);
        s' ← resolveSubproblema(s');
        if f(s') < f(s) then
            s ← s';
            p ← p_0;
            Reiniciar laço de repetição while;
    p ← p + x;
return s
```

Cada uma das estruturas de vizinhança, descritas a seguir, funcionam de forma idêntica ao VND, explorando as vizinhança até que seus mínimos locais sejam encontrados.

4.3.1 Fixação de Dias

Nesta estrutura de vizinhança é feita a fixação de todas as variáveis pertencentes aos enfermeiros que não trabalham em um intervalo de dias. A quantidade de dias, ou melhor, o tamanho do subproblema, altera a todo momento que não há mais melhoras na função objetivo.

Existem quatro formas diferentes para gerar subproblemas. A primeira a ser executada percorre o problema continuamente sem deixar lacunas, formando uma espécie de janela. Sejam d_i e d_f, respectivamente, o início e o fim de uma janela na qual os dias pertencentes a ela permanecem livres. Para se determinar o tamanho da janela, dado que se conhece d_i, basta fazer a seguinte operação: $d_f = d_i + p - 1$.

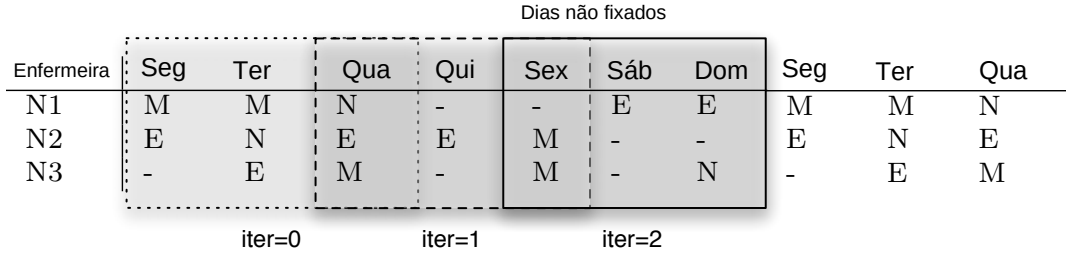

Figura 4.1: *Fixação dos Dias* de uma vizinhança com $p = 3$

A figura 4.1, extraída de (Santos et al., 2012), ilustra muito bem como a busca contínua no espaço de soluções é realizada pela estrutura de vizinhança de fixação dos dias. Perceba

que esta janela percorre, em várias iterações, todo o horizonte de planejamento, do primeiro até o último dia.

Outras duas formas de se gerar uma janela consistem em escolher apenas os dias pares ou ímpares. Por exemplo: Dado que $p = 4$ e o subproblema atual abrange não mais que os dias ímpares, a primeira iteração é constituída dos dias 1, 3, 5 e 7. Consequentemente, a segunda iteração é composta por 3, 5, 7 e 9, continuando sucessivamente. O mesmo procedimento acontece, se ao invés de números ímpares forem números pares.

A quarta, e última, forma de escolher os dias não está associado com a condição do dia ser par ou ímpar. De fato, o fator determinante é a soma total do custo de todas as alocações em um dia d. Os p dias que somaram mais violações são deixados livres e resolvidos pelo CBC.

4.3.2 Fixação de Enfermeiros

Esta estrutura de vizinhança é muito semelhante com a estrutura 4.3.1, só que agora são realizadas as fixações de todos as variáveis do planejamento que pertencem à um grupo de enfermeiros. De semelhante forma, o tamanho do subproblema também é alterado toda vez que não há mais melhora no valor da função objetivo.

Nesta estrutura também há quatro formas de selecionar janelas e são quase idênticas aos modos apresentados na fixação de dias. Para cada valor de p obtêm-se os p enfermeiros que sofreram o maior número de penalizações, uma variedade de janelas com p enfermeiros exclusivamente pares ou ímpares em cada, e por último janelas contínuas com p enfermeiros.

O parâmetro p determina o tamanho do subproblema. Valores muito pequenos de p podem criar subproblemas que não possuem nenhuma solução melhor e valores muito grandes podem criar subproblemas intratáveis. Quanto menor o valor, maior é o número de diferentes subproblemas diferentes na vizinhança.

4.3.3 Fixação Mista

Esta estrutura é um pouco mais complexa que as duas anteriores, na verdade, se trata da interseção de ambas. Aqui os subproblemas são gerados pela fixação de todas as alocações de um conjunto de enfermeiros que trabalham em determinados dias do mês.

Digamos que temos um conjunto $D = \{7, 8, 9, 10, 11\}$ e outro conjunto $N = \{3, 5, 7\}$. As únicas alocações que permanecerão livres serão aquelas que pertencem simultaneamente aos dias contidos em D e aos enfermeiros abrangidos por N. Portanto, qualquer alocação atribuída a um enfermeiro de N fora dos dias contidos em D também será fixada.

Na fixação mista o número de células em cada janela é geralmente menor que em 4.3.1 e 4.3.2. Esta estrutura possui dois parâmetros: p_d e p_n. O parâmetro p_d determina quantos dias serão englobados pelo subproblema, enquanto p_n define o número de enfermeiros que participarão do mesmo.

4.3.4 Fixação de valores Idênticos

Esta estrutura de vizinhança, inspirada no *Relaxation Induced Neighborhood Search* (RINS) e previamente descrita na seção 2.2.3, também produz subproblemas, mas de forma diferente das três citadas anteriormente. O subproblema na heurística RINS é constituído pelas variáveis que possuem valores distintos na solução incumbente e na solução fracionária relaxada.

Assim como a adaptação proposta por Gomes et al. (2013), também foram feitas modificações na metaheurística RINS. Esta adaptação consiste em variar o número de variáveis fixadas e verificar em qual delas se encontra a melhor solução, pois quando há muitas variáveis fixas o espaço de busca acaba ficando reduzido e pouco diversificado. Se poucas são fixadas o espaço fica extenso, demandando mais tempo. A ideia desta adaptação é explorar vários tamanhos de problema através de diversos percentuais de fixação.

Perceba que no algoritmo 4.3 há vários parâmetros: $f(.)$ é a função de avaliação, s_i é a incumbente gerada pelo algoritmo *GeraSoluçãoInicial*, s_f é a solução fracionária relaxada e p_0 é o parâmetro que indica a porcentagem das variáveis idênticas que serão inicialmente fixadas.

Algoritmo 4.3: FixaçãoRINS

Input: $f(.)$, s_i, s_f, p_0
$p \leftarrow p_0$;
$f(s) \leftarrow f(s_i)$;
$s \leftarrow fixaIgualdades(s_i, s_f, p)$;
while $p < 100\%$ **do**
  $s' \leftarrow resolveSubproblema(s)$;
  if $f(s') < f(s)$ **then**
    $s \leftarrow s'$;
    $p \leftarrow p_0$;
  else
    $p \leftarrow p + x\%$;
return s

Neste método, se p é igual a 30%, significa que o algoritmo fixará as variáveis idênticas até atingir este percentual. Se alguma melhora na solução for obtida, p receberá o valor inicial p_0, caso contrário o percentual de variáveis fixadas aumenta em x%.

4.4 Algoritmo Proposto

O algoritmo proposto combina as estruturas de vizinhança detalhadas neste capítulo, inclusive a adaptação da metaheurística RINS. Os subproblemas são criados por estas estruturas que realizam fixações. O valor resultante s é atualizado por elas somente se resultados ainda melhores forem obtidos. Durante o processo de desenvolvimento, houve pelo menos três modificações consideráveis.

A primeira delas, e consequentemente mais simples, foi desenvolvida no início do projeto e ainda não possuía a estrutura de vizinhança baseada no método RINS. Nesta etapa o NRPS se comportava de forma semelhante ao *Variable Neighborhood Descent*. A principal diferença desta versão para o VND está no momento em que acontece uma melhora na solução. No VND, quando há uma melhora, a busca é reiniciada utilizando a primeira estrutura de vizinhança. Já no algoritmo 4.4, a busca é reiniciada dentro da própria estrutura de vizinhança.

Algoritmo 4.4: Versão Inicial

```
Input: f(.), x, p_d, p_n
f(s) ← ∞;
s' ← geraSoluçãoInicial(f(.));
while f(s') < f(s) e Tempo limite não for atingido do
    s' ← fixaçãoDias(f(.), s, p_d);
    s' ← fixaçãoEnfermeiros(f(.), s', p_n);
    s' ← fixaçãoMista(f(.), s', p_d, p_n);
    if f(s') < f(s) then
        s ← s';
return s
```

A segunda versão do algoritmo foi inspirada na metaheurística *Multi-Start* (MS), no qual são geradas um determinado número de soluções pelo método *geraSoluçãoInicial*. O pseudo-código do método proposto baseado no *Multi-Start* está detalhado no algoritmo 4.5. Neste algoritmo, x indica o número máximo de iterações, v indica o percentual padrão de fixações do método *fixaçãoRINS* enquanto p_d e p_n determinam o tamanho do subproblema para as estruturas restantes.

Algoritmo 4.5: Versão Multi-Start

```
Input: f(.), x, v, p_d, p_n
f(s) ← ∞;
repeat
    s_i ← geraSoluçãoInicial(f(.));
    s_f ← solução fracionária relaxada;
    while f(s') < f(s) e Tempo limite não for atingido do
        s' ← fixaçãoRINS(f(.), s_i, s_f, v);
        s' ← fixaçãoDias(f(.), s', p_d);
        s' ← fixaçãoEnfermeiros(f(.), s', p_n);
        s' ← fixaçãoMista(f(.), s', p_d, p_n);
        if f(s') < f(s) then
            s ← s';
until atingir x iterações;
return s
```

A outra variante possui os mesmos parâmetros de entrada do algoritmo 4.5, todavia foi

chamado de *Versão Best-Start* (BS), pois elabora várias soluções iniciais antes de realizar qualquer processo de refinamento. Observe que no primeiro laço de repetição (*repeat*) são elaboradas x soluções iniciais, e de todas elas é escolhida unicamente a melhor. No algoritmo 4.6 a primeira seção de refinamento é executada isoladamente pela metaheurística RINS adaptada. Quando o RINS chega ao fim, é dado início à execução concorrente das estruturas de fixação dos dias, dos enfermeiros e de fixação mista.

Algoritmo 4.6: Versão Best-Start

```
Input: f(.), x, v, p_d, p_n
f(s_i) ← ∞;
repeat
    s ← geraSoluçãoInicial(f(.));
    if f(s) < f(s_i) then
        s_i ← s;
        f(s_i) ← f(s);
until atingir x iterações;
s_f ← solução fracionária relaxada;
s' ← fixaçãoRINS(f(.), s_i, s_f, v);
while f(s') < f(s) e Tempo limite não for atingido do
    s' ← fixaçãoDias(f(.), s', p_d);
    s' ← fixaçãoEnfermeiros(f(.), s', p_n);
    s' ← fixaçãoMista(f(.), s', p_d, p_n);
    if f(s') < f(s) then
        s ← s';
return s
```

Execução Concorrente

Dentre as três versões desenvolvidas, a primeira foi a única a não utilizar recursos *multithreading*. Em compensação, as versões *Multi-Start* e *Best-Start* foram paralelizadas conforme Hansen et al. (2008), ou seja, cada vizinhança é executada por um núcleo de processamento independente.

Ao final da execução de todas as *threads* é feita a seleção da melhor solução retornada por elas. Enquanto o tempo limite não for estourado e enquanto a solução alcançar melhorias o algoritmo re-executa o processo de otimização em paralelo.

A figura 4.2 mostra, de forma bem objetiva, o funcionamento dos algoritmos *Multi-Start* e *Best-Start* em paralelo. Em (a) todas as estruturas de vizinhança são executadas concorrentemente, mas em (b) há o processamento inicial por meio do algoritmo RINS.

Neste projeto foi escolhida a API *OpenMP* ao invés da *Pthread*. Ao trabalhar com laços de repetição, o *OpenMP* é muito mais fácil de utilizar, além de ocultar grande parte do código de infraestrutura. A principal justificativa da escolha se dá à facilidade de se adaptar um código

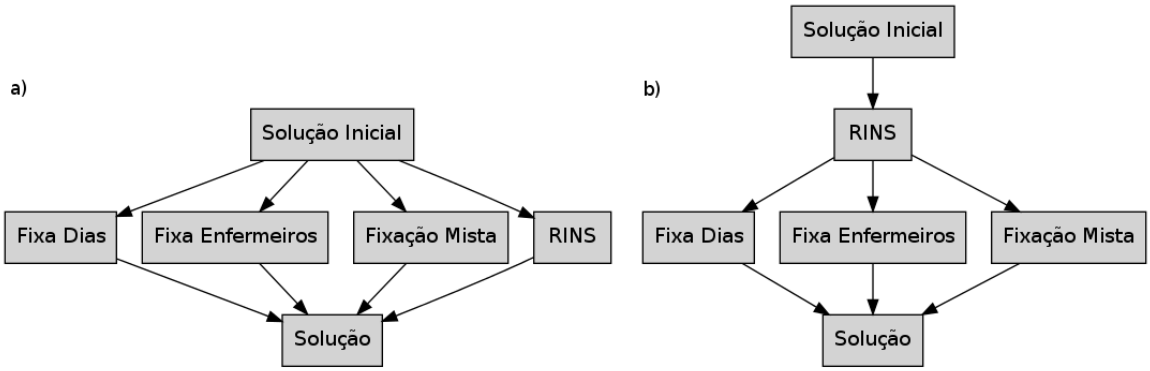

Figura 4.2: *Multi-Start*(a) e *Best-Start*(b) multi-processados

sequencial em paralelo utilizando este modelo se comparado ao *Pthread*, necessitando apenas de algumas alterações.

Como quase todos os métodos já tinham sido desenvolvidos, o *OpenMP* facilitou o processo de adaptação do algoritmo. Com a paralelização da geração de subproblemas foi possível manter o mesmo tempo de execução e desempenhar muito mais processos de refinamento, além de explorar com maior eficiência os recursos da máquina.

Capítulo 5

Experimentos Computacionais

Aqui é apresentado as características das instâncias, os procedimentos tomados durante os experimentos e os resultados obtidos com os métodos que foram descritos. Os experimentos foram realizados utilizando os computadores do Grupo de Otimização e Algoritmos (GOAL)[1].

5.1 Instâncias INRC

As instâncias foram liberadas no início da INRC em 2010 e, no final, os competidores submeteram seus melhores resultados encontrados no intervalo de tempo permitido para cada instância. Apesar da competição ter ocorrido alguns anos atrás, os resultados ainda são divulgados para que pesquisadores possam contrastar suas soluções.

As instâncias da INRC estão divididas em três categorias: *Sprint*, *Medium* e *Long*. As instâncias *Sprint* caracterizam problemas contendo um menor número de enfermeiros, exatamente 10, e são instâncias geralmente resolvidas em menor tempo. As instâncias *Medium* possuem 30 ou 31 enfermeiros e já apresentam certa dificuldade para serem resolvidas. Já as instâncias *Long* apresentam o maior número de enfermeiros, 50 ao todo, e necessitam de um intervalo maior de tempo para que seja possível alcançar bons resultados. Estas instâncias se diferem principalmente na demanda diária, no número de contratos, no número de padrões indesejados e principalmente nas preferências pessoais de cada enfermeiro.

As tabelas 5.1 e 5.2 descrevem, com maiores detalhes, os atributos e a dificuldade imposta por cada um destes problemas. A coluna |N| indica o número total de enfermeiros, |D| os dias do planejamento, |S| o número de turnos, |C| a quantidade de contratos, |P| são os números de padrões indesejados e |DOR| e |SOR| representam, respectivamente, as requisições para não trabalhar em determinados dias e em determinados turnos.

[1] http://www.goal.ufop.br/

Instância			|N|	|D|	|S|	|C|	|P|	|DOR|	|SOR|
Sprint	Early	01	10	28	4	4	3	100	50
		02	10	28	4	4	3	100	50
		03	10	28	4	4	3	100	50
		04	10	28	4	4	3	100	50
		05	10	28	4	4	3	100	50
		06	10	28	4	4	3	100	50
		07	10	28	4	4	3	100	50
		08	10	28	4	4	3	100	50
		09	10	28	4	4	3	100	50
		10	10	28	4	4	3	100	50
	Hidden	01	10	28	3	3	4	100	50
		02	10	28	3	3	4	100	50
		03	10	28	4	3	8	150	70
		04	10	28	4	3	8	100	50
		05	10	28	3	3	8	100	50
		06	10	28	4	3	4	200	100
		07	10	28	3	3	4	200	100
		08	10	28	4	3	8	250	120
		09	10	28	4	3	8	200	100
		10	10	28	4	3	8	200	100
	Late	01	10	28	4	3	8	100	50
		02	10	28	3	3	4	100	39
		03	10	28	4	3	8	100	50
		04	10	28	4	3	8	100	50
		05	10	28	4	3	8	100	50
		06	10	28	4	3	0	100	50
		07	10	28	4	3	0	100	50
		08	10	28	4	3	0	0	0
		09	10	28	4	3	0	0	0
		10	10	28	4	3	0	100	50

Tabela 5.1: Detalhes das instâncias Sprint

Instância			|N|	|D|	|S|	|C|	|P|	|DOR|	|SOR|
Medium	Early	1	31	28	4	4	0	93	310
		2	31	28	4	4	0	93	310
		3	31	28	4	4	0	93	310
		4	31	28	4	4	0	93	310
		5	31	28	4	4	0	93	310
	Hidden	1	30	28	5	4	9	0	0
		2	30	28	5	4	9	0	0
		3	30	28	5	4	9	0	0
		4	30	28	5	4	9	0	0
		5	30	28	5	4	9	0	0
	Late	1	30	28	4	4	7	90	300
		2	30	28	4	3	7	90	300
		3	30	28	4	4	0	90	300
		4	30	28	4	3	7	90	300
		5	30	28	5	4	7	90	300
Long	Early	1	49	28	5	3	3	490	245
		2	49	28	5	3	3	490	245
		3	49	28	5	3	3	490	245
		4	49	28	5	3	3	490	245
		5	49	28	5	3	3	490	245
	Hidden	1	50	28	5	3	9	0	0
		2	50	28	5	3	9	0	0
		3	50	28	5	3	9	0	0
		4	50	28	5	3	9	0	0
		5	50	28	5	4	9	0	0
	Late	1	50	28	5	3	9	0	0
		2	50	28	5	4	9	0	0
		3	50	28	5	3	9	0	0
		4	50	28	5	4	9	0	0
		5	50	28	5	3	9	0	0

Tabela 5.2: Detalhes das instâncias Medium e Long

5.1.1 Formato do arquivo de entrada

Todos os arquivos de entrada foram disponibilizados no formato *XML* e texto. O conteúdo de uma instância, seja ela *XML* ou texto é exatamente o mesmo. Cada instância contém os seguintes atributos:

- *SchedulingPeriod*: representa um NRP com um número identificador único;
- *StartDate*: data do inicio do período de planejamento;
- *EndDate*: data limitando o fim do horizonte de planejamento;
- *Skills*: a capacidade e aptidão que um enfermeiro possui;
- *ShiftTypes*: cada turno possui um identificador (*ID*), horário inicial (*StartTime*), horário final (*EndTime*), uma pequena descrição e uma lista de habilidades necessárias;
- *Patterns*: são sequências indesejadas que devem ser evitadas;
- *Contracts*: possui regras individuais de trabalho de um enfermeiro. Neste elemento estão descritos a grande maioria das restrições fracas presentes no Capítulo 3;
- *Employees*: lista dos enfermeiros que estão disponíveis. Cada enfermeiro está associado a um contrato e possui uma determinada habilidade;
- *Day of week cover*: especificação do número de enfermeiros necessários por turno em cada dia da semana;
- *Date specific cover*: determinação do número de enfermeiros exigidos para um determinado dia e turno;
- *DayOffRequests*: para cada requisição de folga é informado o ID do enfermeiro e o dia;
- *DayOnRequests*: para cada requisição de trabalho é dado o ID do enfermeiro e o dia;
- *ShiftOffRequests*: em cada requisição para não trabalhar em um turno é fornecido o ID do enfermeiro, o turno e o dia;
- *ShiftOnRequests*: na requisição para trabalhar em um turno específico é fornecido o ID do enfermeiro, o turno e o dia;

5.1.2 Compatibilidade com CBC

Para a resolução dos problemas foi necessário utilizar as instâncias no formato texto para gerar a solução inicial válida. Ao introduzir o resolvedor CBC, foi estrutural utilizar instâncias no formato CPLEX LP[2]. Os arquivos compatíveis com estes resolvedores recebem como rótulo a extensão LP e são muito utilizados, uma vez que facilitam a representação do problema na forma algébrica.

Observando as instâncias da INRC, percebe-se que elas não são compatíveis com resolvedores de programação inteira mista. Uma das formas de gerar estes arquivos é através do GNU Linear Programming Kit (GLPK) (Makhorin, 2001). O GLPK é um resolvedor de problemas lineares e de programação inteira com abundante documentação. Além do formato LP, o GLPK recebe como entrada arquivos modelados em MathProg[3]: uma linguagem capaz de descrever modelos de programação matemática linear. Para se modelar um problema em MathProg são necessários dois arquivos:

- Um arquivo principal com extensão MOD em que um problema é representado em termos de conjuntos, parâmetros, variáveis, restrições e função objetivo. Este arquivo contém a formulação do problema, e é único para todas as instâncias da INRC.

- Um arquivo auxiliar com extensão DAT, muito utilizado quando há diferentes dados de entrada para serem resolvidos usando o mesmo modelo. Arquivos DAT são opcionais e contém detalhes de cada instância em particular.

Os arquivos DAT são facilmente adaptados a partir das instâncias disponibilizadas no formato texto, já o arquivo MOD foi gerado em conformidade com a formulação do problema descrita no capítulo 3. Com ambos os arquivos, o GLPK efetua a conversão para o formato LP. A conversão se faz necessária visto que o CBC é incompatível com estes formatos, que por outro lado, são mais fáceis de serem gerados.

Dependendo da quantidade de detalhes do problema descritos no DAT, o GLPK consome mais de alguns segundos para fazer a conversão. Por este motivo, durante o trabalho, também foi desenvolvido em C++ um mini-sistema para gerar arquivos LP direto e exclusivamente dos arquivos DAT abordados. Este sistema abrange somente os problemas concedidos pela INRC, tanto que os testes foram realizados com todas as 60 instâncias previamente descritas neste capítulo.

Veja na tabela 5.3 a comparação do tempo gasto, em segundos, pelo GLPK e pelo mini-sistema ao executar as conversões. Os resultados mostraram que o sistema criado gera arquivos no formato CPLEX LP mais rapidamente que o GLPK sem a necessidade do arquivo MOD, que foi, de certa forma, integrado ao sistema.

[2]http://lpsolve.sourceforge.net/5.5/CPLEX-format.htm
[3]http://lpsolve.sourceforge.net/5.5/MathProg.htm

Instâncias			GLPK	Interpret	Instâncias			GLPK	Interpret
Sprint	Early	1	4.452	0.114	Long	Early	1	7.887	0.523
		2	4.526	0.111			2	7.714	0.510
		3	2.036	0.105			3	8.049	0.520
		4	2.043	0.105			4	7.658	0.569
		5	2.027	0.107			5	7.621	0.512
		6	2.028	0.104		Hidden	1	8.456	0.592
		7	2.032	0.105			2	8.478	0.579
		8	2.028	0.107			3	8.620	0.601
		9	2.025	0.112			4	8.673	0.572
		10	2.040	0.117			5	8.673	0.586
	Hidden	1	2.798	0.106		Late	1	20.341	0.573
		2	2.219	0.105			2	9.841	0.572
		3	3.780	0.114			3	9.617	0.581
		4	2.207	0.123			4	9.671	0.572
		5	2.118	0.120			5	9.601	0.574
		6	2.088	0.104	Medium	Early	1	5.345	0.300
		7	3.511	0.106			2	5.389	0.305
		8	2.474	0.113			3	5.341	0.294
		9	3.501	0.112			4	5.346	0.301
		10	2.214	0.111			5	11.035	0.305
	Late	1	2.096	0.113		Hidden	1	12.467	0.338
		2	2.541	0.103			2	5.820	0.342
		3	2.102	0.114			3	5.803	0.338
		4	2.104	0.114			4	5.817	0.348
		5	2.091	0.114			5	5.957	0.347
		6	2.006	0.101		Late	1	5.636	0.324
		7	1.994	0.111			2	7.861	0.326
		8	2.055	0.105			3	10.338	0.298
		9	2.003	0.103			4	6.620	0.327
		10	2.000	0.101			5	6.716	0.331

Tabela 5.3: Tempo de geração de arquivos LP em segundos

5.2 Experimentos

Experimentos computacionais foram realizados para verificar o desempenho dos algoritmos desenvolvidos com todas as 60 instâncias disponibilizadas pela competição. Os experimentos, tanto para a versão serial ou paralela, foram realizados em uma máquina com as seguintes configurações:

- Sistema Operacional Ubuntu 12.04 Server 64bits
- Kernel Linux 3.2.0-23-generic
- Processador Intel Core i5-650 CPU @ 3.20GHz – 16Gb RAM
- Linguagem: C/C++ GCC 4.6.3
- CBC MIP Solver 2.8.3
- OpenMP 3.1

Foram utilizadas 4 threads simultaneamente na execução da versão *Multi-Start*. Para a versão *Best-Start* do algoritmo foram utilizadas somente 3 threads já que o método RINS não foi executado concorrentemente às outras estruturas de vizinhança.

Ao pesquisar outros trabalhos na literatura que também utilizaram instâncias da INRC (Burke e Curtois, 2012), foi possível observar que o tempo de execução das instâncias variam entre alguns minutos nas instâncias *Sprint* e podem chegar até a 10 horas, como foi o caso dos problemas da categoria *Long*.

Para os experimentos realizados no laboratório GOAL foi estipulado o tempo máximo de execução para cada categoria das instâncias. As instâncias do tipo *Sprint* não poderiam ultrapassar o limite máximo de 15 minutos, enquanto as instâncias *Medium* não poderiam ser executadas por mais de 60 minutos. Por último, e não menos importante, foram executados testes com as instâncias *Long* com tempo máximo de 180 minutos.

Na versão *Multi-Start*, os tempos estipulados no parágrafo anterior correspondem exatamente ao tempo total executado. Porém o tempo total de execução é dividido pelo número de iterações do algoritmo MS. Assim, dado que o tempo de execução de uma instância é 15 minutos e são realizadas três iterações, o tempo de execução de cada iteração é de 5 minutos.

No algoritmo *Best-Start*, a execução foi um pouco diferente. Como pode ser observado na figura 4.2, o método RINS não foi executado ao mesmo tempo das outras estruturas, pois estas vizinhanças foram executadas como uma espécie de pós-processamento.

A tabela 5.4 demonstra, respectivamente, o número de iterações realizada no MS e o tempo de execução de cada iteração, o tempo gasto na metaheurística adaptada RINS e o tempo total gasto pelas estruturas de vizinhança no BS, além do tempo total de execução. A unidade de tempo é estabelecida em minutos.

Categoria	Multi-Start		Best-Start		Total
	N. Iterações	***T. Iterações***	***RINS***	***Vizinhança***	
Sprint	3	5	10	5	15
Medium	3	20	45	15	60
Long	3	60	130	50	180

Tabela 5.4: Distribuição do tempo de execução do algoritmo MS e BS

Tais procedimentos foram adotados quando foi constatado que a utilização de todas as quatro estruturas de vizinhanças concorrentemente poderia acarretar em uma alta utilização da memória RAM. Este uso, principalmente em instâncias *Long*, chegou ao ponto de utilizar toda a memória RAM e ainda preencher boa parte da memória virtual. Como memórias virtuais se encontram na memória secundária, um dispositivo de certa forma lento, o desempenho dos algoritmos caem drasticamente.

5.3 Resultados

Com o propósito de comparação, foram adicionados os melhores resultados conhecidos pelos administradores da INRC[4], além dos resultados obtidos por Gomes (2012).

Os resultados da versão inicial do NRPS, aquele método de execução sequencial, não foram expostos na tabela. Seus testes só foram realizados com instâncias da categoria *Sprint* tendo o limite de tempo preservado. Quase todos os resultados alcançados pela versão inicial foram superados pelos métodos executados em paralelo, em algumas instâncias foram obtidos resultados idênticos aos algoritmos BS e/ou MS.

Os resultados podem ser observados nas tabelas 5.5 e 5.6. A coluna INRC indica os resultados, extraídos no dia 20 de agosto de 2013, do site da competição internacional. Na coluna GOMES estão os resultados obtidos por outro pesquisador, todavia, seus experimentos foram utilizados utilizando o resolvedor CPLEX. Já a coluna COIN-OR CBC aponta os resultados obtidos pela utilização do resolvedor de forma independente, sem a integração de métodos externos. As colunas NRPSMS e NRPSBS indicam, respectivamente, os resultados obtidos pelas duas versões paralelas desenvolvidas: a versões *Multi-Start* e *Best-Start*. Os campos da tabela preenchidos com SNE indicam que nenhuma solução foi encontrada com o tempo máximo utilizado.

[4]http://www.kuleuven-kulak.be/nrpcompetition/instances-results

Instância			INRC	GOMES	Gap	Coin-Or CBC	Gap	NRPSMS	Gap	NRPSBS	Gap
Sprint	Early	01	56	56	0.00	56	0.00	56	0.00	56	0.00
		02	58	58	0.00	58	0.00	59	1.72	58	0.00
		03	51	51	0.00	51	0.00	52	1.96	51	0.00
		04	59	59	0.00	59	0.00	60	1.69	61	3.39
		05	58	58	0.00	58	0.00	58	0.00	58	0.00
		06	54	54	0.00	54	0.00	54	0.00	54	0.00
		07	56	56	0.00	56	0.00	58	3.57	56	0.00
		08	56	56	0.00	56	0.00	56	0.00	57	1.79
		09	55	55	0.00	55	0.00	56	1.82	55	0.00
		10	52	52	0.00	52	0.00	53	1.92	52	0.00
	Hidden	01	32	32	0.00	33	3.13	37	15.63	37	15.63
		02	32	32	0.00	32	0.00	47	46.88	33	3.13
		03	62	62	0.00	62	0.00	81	30.65	62	0.00
		04	66	66	0.00	72	9.09	70	6.06	68	3.03
		05	59	59	0.00	59	0.00	75	27.12	60	1.69
		06	130	130	0.00	130	0.00	157	20.77	132	1.54
		07	153	153	0.00	153	0.00	163	6.54	156	1.96
		08	204	204	0.00	204	0.00	267	30.88	217	6.37
		09	338	338	0.00	384	13.61	351	3.85	348	2.96
		10	306	306	0.00	306	0.00	306	0.00	306	0.00
	Late	01	37	37	0.00	38	2.70	45	21.62	40	8.11
		02	42	42	0.00	42	0.00	49	16.67	43	2.38
		03	48	48	0.00	54	12.50	65	35.42	52	8.33
		04	73	73	0.00	108	47.95	113	54.79	82	12.33
		05	44	44	0.00	44	0.00	46	4.55	45	2.27
		06	42	42	0.00	42	0.00	43	2.38	42	0.00
		07	42	42	0.00	42	0.00	46	9.52	43	2.38
		08	17	17	0.00	28	64.71	17	0.00	18	5.88
		09	17	17	0.00	27	58.82	20	17.65	19	11.76
		10	43	43	0.00	43	0.00	47	9.30	45	4.65

Tabela 5.5: Resultado das intâncias Sprint

Instância			INRC	Gomes	Gap	Coin-Or CBC	Gap	NRPSMS	Gap	NRPSBS	Gap
Medium	Early	01	240	240	0.00	254	5.83	265	10.42	245	2.08
		02	240	240	0.00	252	5.00	286	19.17	246	2.50
		03	236	236	0.00	SNE	-	263	11.44	239	1.27
		04	237	237	0.00	238	0.42	276	16.46	242	2.11
		05	303	303	0.00	304	0.33	348	14.85	309	1.98
	Hidden	01	122	111	-9.02	SNE	-	292	139.34	173	41.80
		02	221	221	0.00	SNE	-	364	64.71	271	22.62
		03	36	34	-5.56	SNE	-	118	227.78	57	58.33
		04	80	80	0.00	SNE	-	147	83.75	99	23.75
		05	119	119	0.00	SNE	-	286	1140.34	156	31.09
	Late	01	157	157	0.00	250	59.24	267	70.06	194	23.57
		02	18	18	0.00	SNE	-	93	416.67	42	133.33
		03	29	29	0.00	51	75.86	126	334.48	50	72.41
		04	35	35	0.00	128	265.71	90	157.14	59	68.57
		05	107	107	0.00	573	435.51	296	176.64	161	50.47
Long	Early	01	197	197	0.00	SNE	-	215	9.14	201	2.03
		02	219	219	0.00	219	0.00	246	12.33	234	6.85
		03	240	240	0.00	240	0.00	253	5.42	241	0.42
		04	303	303	0.00	303	0.00	308	1.65	307	1.32
		05	284	284	0.00	292	2.82	297	4.58	287	1.06
	Hidden	01	346	346	0.00	SNE	-	583	68.50	483	39.60
		02	89	89	0.00	SNE	-	184	106.74	138	55.06
		03	38	38	0.00	680	1689.47	119	213.16	113	197.37
		04	22	22	0.00	SNE	-	121	450.00	69	213.64
		05	41	41	0.00	670	1534.15	125	204.88	99	141.46
	Late	01	235	235	0.00	SNE	-	426	81.28	315	34.04
		02	229	229	0.00	2725	1089.96	435	89.96	365	59.39
		03	220	220	0.00	2053	833.18	502	128.18	358	62.73
		04	221	222	0.45	SNE	-	446	101.81	335	51.58
		05	83	83	0.00	SNE	-	271	226.51	193	132.53

Tabela 5.6: Resultado das intâncias Medium e Long

Análise dos Resultados

Analisando os resultados nas tabelas 5.5 e 5.6, percebe-se que os algoritmos MS e BS obtiveram bons resultados. No entanto, não foi possível atingir soluções ótimas em todas as instâncias.

Comparando o desempenho das duas versões paralelas implementadas, é possível notar que a versão *Best-Start* obteve, em grande maioria, resultados significativamente melhores. O fato da versão BS sempre partir de soluções iniciais melhores que o MS pode justificar seu melhor desempenho. Um fator que provavelmente poderia favorecer a versão *Multi-Start* seria permitir que o algoritmo executasse em cada iteração o tempo total permitido. Deste modo, o algoritmo seria executado por 15 minutos em cada iteração das instâncias *Sprint*, 60 minutos em cada iteração de problemas do tipo *Medium* e 180 minutos por iteração em instâncias *Long*.

Além das tabelas contendo os resultados computacionais, as figuras 5.1, 5.2 e 5.3 contêm gráficos comparativos que permitem visualizar os resultados obtidos de forma mais superficial. Nos gráficos abaixo, a primeira série de instâncias (de 1 a 10 ou de 1 a 5) representam as instâncias do tipo *early*, a segunda descreve o tipo *hidden* e a terceira o tipo *late*.

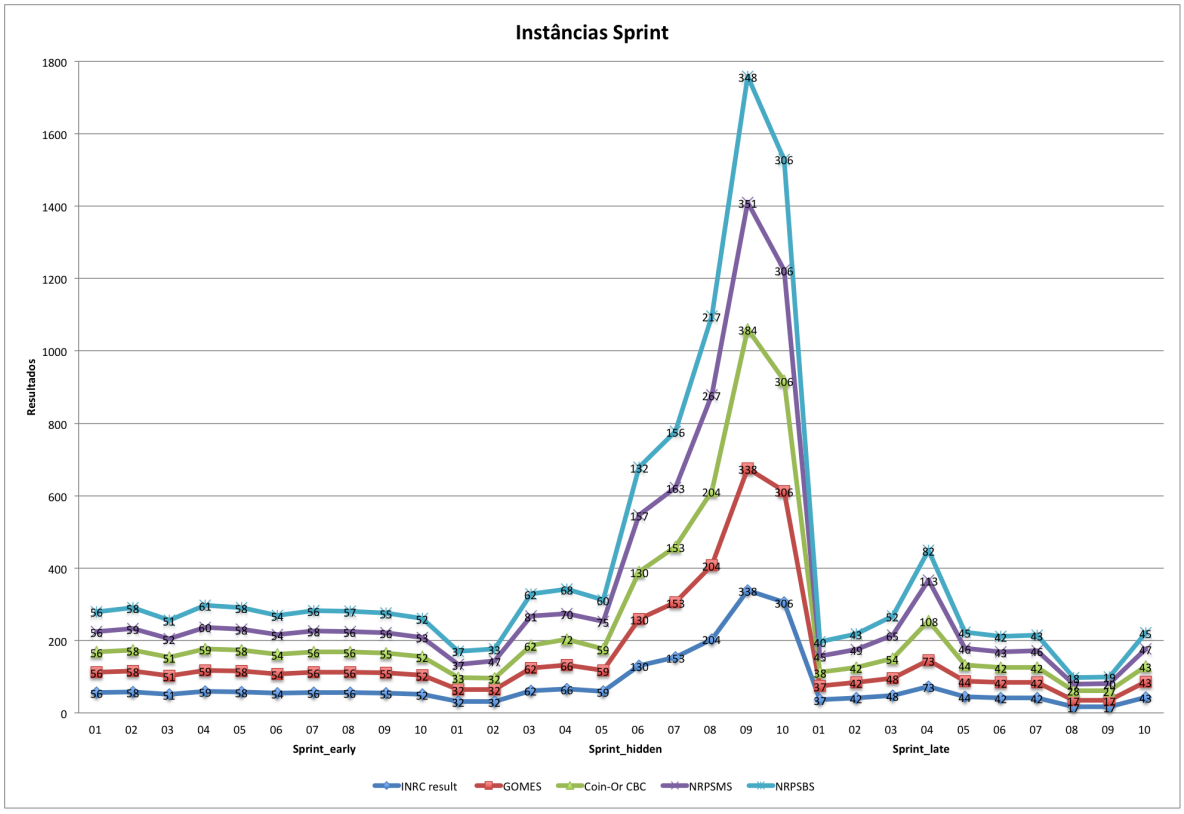

Figura 5.1: Gráfico comparativo com instâncias *Sprint*

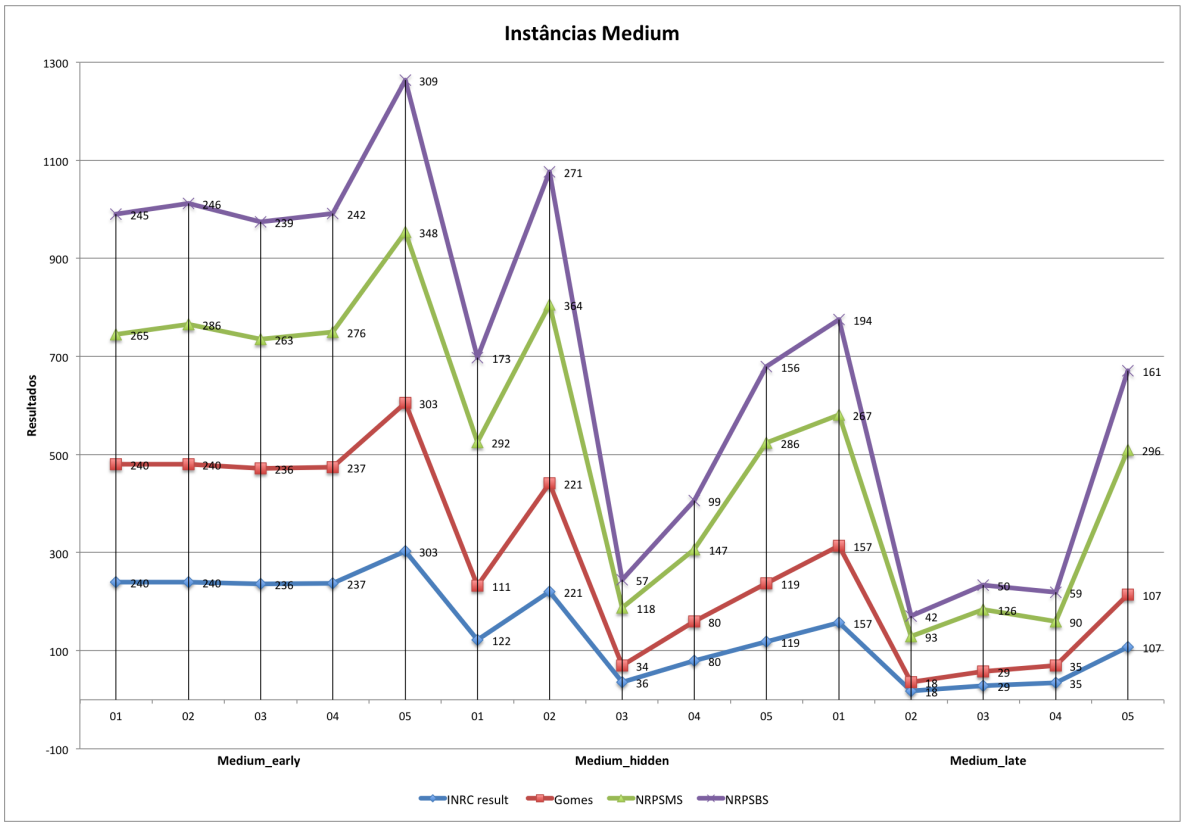

Figura 5.2: Gráfico comparativo com instâncias *Medium*

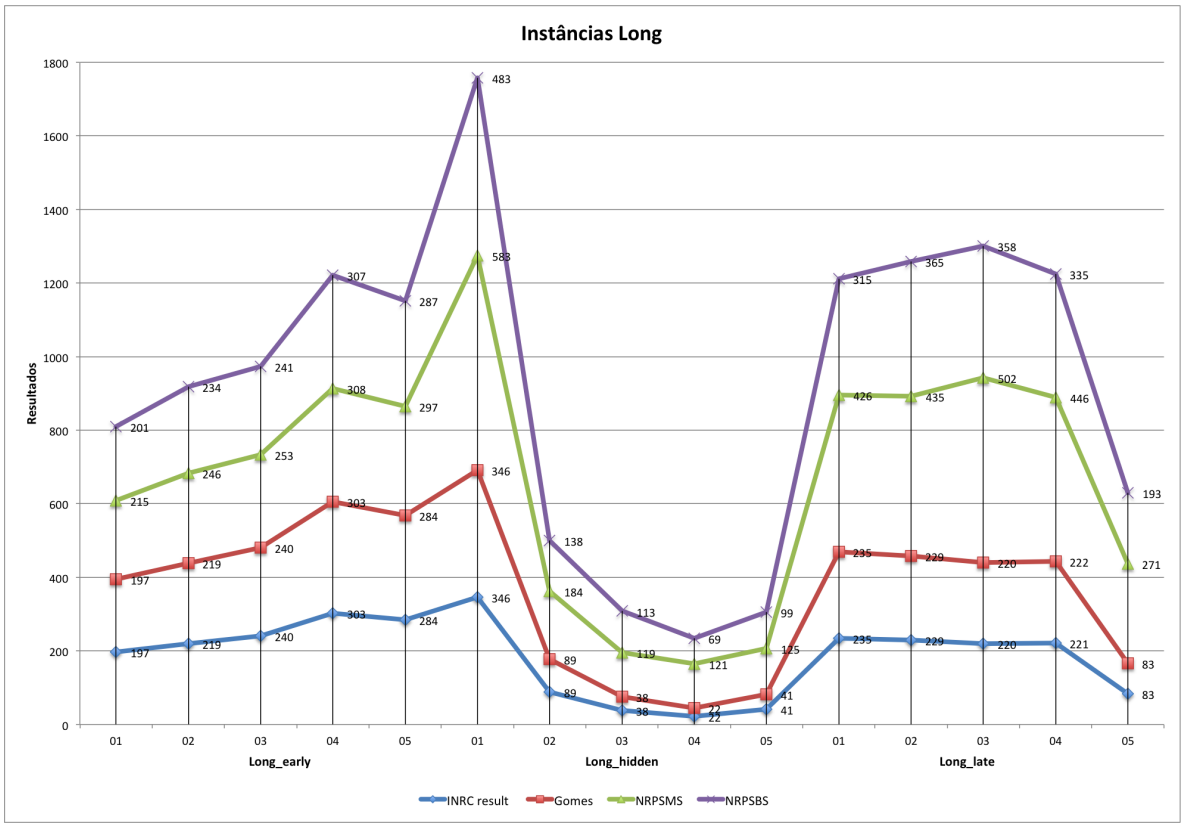

Figura 5.3: Gráfico comparativo com instâncias *Long*

Capítulo 6

Conclusões

Neste trabalho de conclusão de curso foi apresentada uma abordagem para o problema de escalonamento de enfermeiros. Em virtude de sua dificuldade de solução foram utilizadas heurísticas, metaheurísticas, técnicas de concorrência e de programação inteira.

Dentre as atividades desenvolvidas, destacam-se as estruturas de fixação – responsáveis por gerar problemas menores e acelerar o processo de otimização – e a adaptação da metaheurística RINS. Na etapa final de desenvolvimento o sistema passou a executar de forma concorrente, explorando de forma mais aguda os recursos de computadores *multi-core*.

Para avaliar os métodos propostos, foram executados vários experimentos utilizando uma variedade de instâncias, disponibilizadas pela competição internacional (INRC), com características distintas, analisando assim, o desempenho do método em diversos cenários. No entanto, este trabalho não foi suficiente para tornar o CBC competitivo nas instâncias da INRC quando comparado aos melhores resolvedores comerciais, como o CPLEX.

Como neste trabalho foram desenvolvidas, em teoria, quatro estruturas de vizinhança, a primeira alternativa para tentar melhorar os resultados obtidos seria implementar novas estruturas de vizinhança para que o espaço de soluções seja explorado de forma mais diversificada. Além das novas estruturas de vizinhança, outros métodos de busca local podem acrescentar muito ao sistema, aumentando o número de soluções candidatas.

Caso resultados melhores não sejam alcançados com tais alterações, outras metaheurísticas podem ser avaliadas. Uma alternativa seria implementar o Iterated Local Search (ILS) de forma paralela aos métodos desenvolvidos, composto por diversos métodos de perturbação. Além do ILS, a Busca Tabu (BT) também é amplamente utilizada para resolver problemas combinatórios, inclusive o NRP. Desta forma, agregaríamos ainda mais a flexibilidade das metaheurísticas com garantia e poderio da programação matemática.

Apêndice

Soluções Obtidas

As tabelas 6.1 e 6.2 apresentam exemplos de soluções possíveis para as instâncias *sprint_hidden08* e *medium_hidden04*. Nestas tabelas, cada linha representa um enfermeiro e cada coluna um dia do planejamento. Cada célula $m_{i,j}$ indica o turno a ser trabalhado pelo enfermeiro i no dia j. As Células que se encontram vazias indicam dias de folga.

SPRINT_HIDDEN08																												
Enf.	**d1**	**d2**	**d3**	**d4**	**d5**	**d6**	**d7**	**d8**	**d9**	**d10**	**d11**	**d12**	**d13**	**d14**	**d15**	**d16**	**d17**	**d18**	**d19**	**d20**	**d21**	**d22**	**d23**	**d24**	**d25**	**d26**	**d27**	**d28**
0	E	E			E	E	E	E	D			D	D	L				E	D	D	E	E			E	N	N	N
1	N	N	N					E	N	N	N			D	D	D	E	E			D	L	L			D	D	E
2	L	L	L	L			L	L	L	L	L			N	L	L	L	L			E	E	D	D	E			
3			D	E	N	N	N			D	E	N	N	L			L	L	L	L	L			L	L	L	L	L
4	L	L	L	L			E	D	E	E	L			E	L	L			N	N	L			E	D	E	E	L
5	E	E	E				L	L				E	E	E					E	E				E	N			
6				D	L	L				E	E				E	E	D	D				D	E	N				
7	D	D	E	N					E	L					E	E	N					L	L					E
8				E	D	D	D				D	L	L				E	N					E	L	L			
9								N	L						N	N					N	N	N					D

Tabela 6.1: Exemplo de solução para instâncias *sprint_ hidden*

MEDIUM_HIDDEN04																												
Enf.	**d1**	**d2**	**d3**	**d4**	**d5**	**d6**	**d7**	**d8**	**d9**	**d10**	**d11**	**d12**	**d13**	**d14**	**d15**	**d16**	**d17**	**d18**	**d19**	**d20**	**d21**	**d22**	**d23**	**d24**	**d25**	**d26**	**d27**	**d28**
0	E	E	D	E			E	E	E	D	L			E	D	E	E	L			E	D	E	E	L			E
1	E	D	D	L			E	E	D	D	L			E	E	L	N	L			D	E	E	E	E			E
2	E	D	E	E			D	D	E	E	E			E	N	N	L	L			D	D	E	E	D			N
3				D	E	E	E	D			N	N	N	L	L			E	E	E	E		D	D	E	E	E	D
4	E	D	D	L			D	L	L	N	L			N	L	N	N	L			L	N	L	L	L			L
5	N	N	L	L			D	D	E	E	L			D	E	L	L	L			L	N	L	L	L			N
6	N	N	N	L			E	E	E	E	E			E	D	E	E	E			N	N	L	N	L			D
7	D	E	E	L			N	L	N	N	L			E	D	D	E	E			E	E	E	E	E			D
8				N	L	L	N	L		L	N	L	L	L			E	D	D	D	L			N	N	L	L	L
9				D	D	D	E	E	L		D	E	E	N	L			N	L	L	L	L			D	D	D	E
10				E	D	D	L				N	L	L	N	L	N	L				L	L	L	N	L			
11				N	N	N	N	L	L	L				L	N	L	N				E	N	L	L	L			
12				N	N	N	N				D	D	D	L	L	N	L				L	L	N	N				
13				E	E	E	L				DH	DH	DH	DH	DH	DH		DH	DH	DH	DH	DH	E	DH				
14				E	E	E	L				N	N	N	N	N	L	L				N	L	N	L				
15	E	E	E	E					D	E	E	D	D					N	L	L	N	L						E
20	L	L	L					L	L	L	E					D	D	D	D	D					N	L	L	L
16	L	L	L					E	D	E							E	E	E	E					D	D	D	E
17	L	L	L					E	E	D					E	E	E	D							E	N	N	L
18	L	N	L					L	N	N				L	N	L					E	E	E	E				
19	N	L	N				L	N	N	N				D	E	E					E	E	D	D				
21	D	E	E					N	L	L	L						D	E	L	L					E	E	E	E
22	N	L	L				E	N	N	L				D	E	E	DH				D	E	N	L				
23	E	E	E					N	L	L						D	L	N	N	N					E	E	E	L
24	L	L	N	L					E	E	D	E	E					N	N	N	N	L	N					N
25											E	E	E	E	E	E	D	E	E	E								
26	L	N	N	N	L	L	L																L	L	N	N	N	N
27	D	E	E	D	L	L	L															D	D	E	N	L	L	L
28											E	L	L	L	L	L	N	L										
29	DH	DH	DH	DH	DH	DH	DH	DH	DH	DH												E	DH	D	DH	DH	DH	DH

Tabela 6.2: Exemplo de solução para instâncias *medium_hidden*

Referências Bibliográficas

Almeida, B. M. A.; Toffolo, T. Â. M. e Souza, M. J. F. (2008). Msgvns: Algoritmo heurístico para o problema de gerenciamento da escala operacional de controladores de tráfego aéreo. Monografia, Departamento de Computação, Universidade Federal de Ouro Preto.

Burke, E.; Causmaecker, P. D.; Berghe, G. V. e V, G. (1998). A hybrid tabu search algorithm for the nurse rostering problem. *Proceedings of the Second Asia-Pacific Conference on Simulated Evolution and Learning*.

Burke, E.; Causmaecker, P. D.; Petrovic, S. e Berghe, G. V. (2004). Variable neighbourhood search for nurse rostering problems. In *Book Metaheuristics*, pp. 153–172. Kluwer Academic Publishers Norwell.

Burke, E. K. e Curtois, T. (2012). New computational results for nurse rostering benchmark instances. School of Computer Science, University of Nottingham.

Causmaecker, D. e Berghe, G. V. (2003). Relaxation of coverage constraints in hospital personnel rostering. In in Computer Science, L. N., editor, *P*ractice and Theory of Automated Timetabling, Fourth International Conference, volume 2740, pp. 129–147.

Dagum, L. e Menon, R. (1998). Openmp: an industry standard api for shared-memory programming. *Computational Science Engineering, IEEE*, 5(1):46 – 55.

Danna, E.; Rothberg, E. e C.Pape (2005). Exploring relaxation induced neighbourhoods to improve mip solutions. *Mathematical Programming: Series A and B*, 102:71–90.

Forrest, J. e Lougee-Heimer, R. (2005). *CBC User Guide*. Department of Mathematical Sciences, IBM T. J. Watson Research Center, IBM Research, 1101 Kitchawan Road, Yorktown Heights, New York 10598. Tutorials in Operations Research.

Gomes, R. A. M. (2012). Técnicas de programação inteira para o problema de escalonamento de enfermeiras. Dissertação de Mestrado, Departamento de Computação, Universidade Federal de Ouro Preto.

Gomes, T. M.; Santos, H. G. e Souza, M. J. F. (2013). A pre-processing aware rins based mip heuristic. *8th International Workshop on Hybrid Metaheuristics: HM 2013, 2013, Napoli. Hybrid Metaheuristics*, 7919:1–11.

Hadwan, M. e Ayob, M. (2010). A constructive shift patterns approach with simulated annealing for nurse rostering problem. *2010 International Symposium in Information Technology*, 1:1–6.

Hansen, P.; Mladenović, N. e Pérez, J. A. M. (2008). Variable neighbourhood search: methods and applications. *4OR*, 6:319 – 360.

Haspeslagh, S.; Causmaecker, P. D.; Stolevik, M. e Schaerf, A. (2010). *First International Nurse Rostering Competition 2010*. CODeS, Department of Computer Science, KULeuven Campus Kortrijk, Belgium; SINTEF ICT, Department of Applied Mathematics, Norway; DIEGM, University of Udine, Italy, 1 edição.

Hu, B. e Raidl, G. R. (2006). Variable neighborhood descent with self-adaptive neighborhood-ordering. *Proceedings of the 7th EU/MEeting on Adaptive, Self-Adaptive, and Multi-Level Metaheuristics*.

Lenstra, J.; Kan, A. R. e Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362.

Makhorin, A. (2001). Gnu linear programming kit. *Moscow Aviation Institute, Moscow, Russia*, 38.

Mine, M. T.; Souza, M. J. F.; de Souza Alves Silva, M.; Ochi, L. S. e da Silva, T. C. B. (2010). O problema de roteamento de veículos com coleta e entrega simultânea: uma abordagem via iterated local search e genius. *Revista Transportes*, 18(3):60–71.

Mueller; W., C. e McCloskey, J. C. (1990). Nurses job satisfaction: A proposed measure. *Nursing Research*, 39(2):113–117.

Nepomuceno, N. V. (2006). Combinação de metaheurísticas e programação linear inteira: uma metodologia híbrida aplicada ao problema de carregamento de contêiner. Dissertação apresentada ao Curso de Mestrado em Informática Aplicada da Universidade de Fortaleza como requisito parcial para obtenção do Título de Mestre em Informática.

Qu, R. e He, F. (2009). A hybrid constraint programming approach for nurse rostering problems. *Applications and Innovations in Intelligent Systems XVI*, pp. 211– 224.

Ribas, S.; Coelho, I. M.; Souza, M. J. F. e Menotti, D. (2006). Parallel iterated local search aplicado ao planejamento operacional de lavra. *XLI SBPO 2009 - Pesquisa Operacional na Gestão do Conhecimento*, pp. 2037 – 2048.

Rosenberg, F.; Muller, M. B.; Leitnery, P.; Michlmayry, A.; Bouguettaya, A. e Dustdary, S. (2010). Metaheuristic optimization of large-scale qos-aware service compositions. *2010 IEEE International Conference on Services Computing – Miami, FL*, pp. 97 – 104.

Sakamoto, C.; Miyazaki, T.; Kuwayama, M.; Saisho, K. e Fukuda, A. (1999). Design and implementation of a parallel pthread library (ppl) with parallelism and portability. *Systems and Computers in Japan*, 29:28 – 35.

Saltzman, M.; Ladáanyi, L. e Ralphs, T. (2004). The coin-or open solver interface: Technology overview.

Santos, H. G.; Toffolo, T. Â. M.; Gomes, R. A. e Vareto, R. H. (2012). Integer programming techniques for the nurse rostering problem. In *PATAT 2012 – 9th International Conference on the Practice and Theory of Automated Timetablin*, pp. 257 – 282, Son, Norway. SINTEF.

Silva, A. S. N.; Sampaio, R. M. e Alvarenga, G. B. (2005). Uma aplicação de simulated annealing para o problema de alocação de salas. *Journal of Computer Science - Info Comp*, 4:59–66.

Souza, M. J. F. (2011). *Inteligência Computacional para Otimização*. Universidade Federal de Ouro Preto, 2011/1 edição. Notas de aula da disciplina Inteligência Computacional para Otimização.

Souza, M. J. F.; Toffolo, T. Â. M.; Pontes, R. C. V. e Silva, G. P. (2006). Heurística de recobrimento aplicada à escala de tripulações aéreas. *XXXVIII Simpósio Brasileiro de Pesquisa Operacional*, pp. 1637–1647.

Wikipedia (2011). Metaheuristic. On-line. This page was last modified on 11 August 2013 at 01:57.