Nurse Rostering Problem

Ricardo Pinho, Vitor Figueira

Metodologias de Planeamento e Escalonamento
Mestrado Integrado em Engenharia Informática e Computação
Faculdade de Engenharia Universidade do Porto

27 de Agosto de 2013

Resumo

Nurse Rostering são horários gerados para alocação de enfermeiras no seu horário laboral da sua respetiva especialidade, tendo em conta *hard* e *soft constraints*. Essas restrições induzem um nível de complexidade elevado na construção de algoritmos eficientes e que gerem o pretendido no menor tempo possível. Aplicando o Tabu Search e algoritmos evolucionários (memetic algorithm), podemos analisar a eficácia e desempenho na otimização de horários de forma a satisfazer o maior número de *soft constraints*. Após análise dos resultados obtidos concluímos que todas as soluções apresentam eficácia em diferentes cenários.

1 Introdução

O objetivo deste projeto é a criação de horários para staff de hospital, mais especificamente para enfermeiros, sendo flexível o suficiente para conseguir satisfazer várias necessidades de cada enfermeiro, cada especialidade e do hospital. Pretendemos aplicar algumas soluções algorítmicas ao problema, implementando-as num ambiente de software, de modo a poder comparar várias variáveis, como tempo, número de restrições satisfeitas, a quantidade de pessoas a alocar, entre outras. Com estes dados podemos compará-los e determinar as qualidades e defeitos de cada solução implementada. Após análise deste problema no livro Handbook of Schedulling [1, Capítulo. 44],

mais precisamente no caso de Nurse Rostering, podemos observar que existem diversas soluções presentes, cada uma delas com pros e contras. O documento é composto pela descrição do problema, contendo detalhes sobre a estrutura de dados usada, as *hard* e *soft constraints* usadas, é descrito todos os algoritmos implementados, tanto como para a solução inicial como para as as soluções otimizadas, e finalmente contém análise de dados, comparações entre soluções e a conclusão atingida após análises dos resultados obtidos. Existem várias referências que podem ser encontradas no final do artigo científico para informações mais detalhadas sobre esta matéria.

2 Nurse Rostering

2.1 Descrição do Problema

O problema de nurse rostering consiste em gerar um horário de trabalho, onde estão envolvidos todos os enfermeiros alocados num determinado hospital.

Para esse hospital, cada enfermeiro tem de ter uma alocação conforme a demanda exigida pelo hospital. Um hospital tem várias áreas de especialidade, por isso é exigido que existam vários enfermeiros de cada especialidade alocados num hospital. Mas é preciso notar que cada enfermeiro tem a si restrições de trabalho, como evitar excesso de trabalho ou ser alocado em excesso em determinado turno ou em determinado dia, de entre outras restrições que podem ser aplicadas nestes casos.

Através de algoritmos complexos, podemos obter horários que respeitem o maior número de restrições para todos os enfermeiros de cada especialidade associados a um hospital.

2.2 Restrições

Em cada horário existem dois tipos de restrições (*constraints*): *hard constraints*, que não devem ser violadas para gerar uma solução viável, e *soft constraints* que devem ser cumpridas, dentro do possível, para gerar uma boa solução. Quantas mais *soft constraints* forem cumpridas, melhor é a solução gerada.

2.2.1 Hard Constraints

Existem 3 *Hard Constraints* que devem ser cumpridas para gerar uma solução viável:

- Todas as vagas de cada turno em cada especialidade têm que ser preenchidas por funcionários(enfermeiros).
- Nenhum enfermeiro pode ser alocado duas vezes no mesmo turno.
- Cada especialidade só pode ver as suas vagas preenchidas por enfermeiros com a mesma especialidade.

2.2.2 Soft Constraints

Existem muitas *Soft Constraints*, pelo que serão apenas mencionadas as restrições aplicadas nas soluções e nos dados obtidos (Estas restrições referem-se a cada enfermeiro do hospital):

- Número máximo de dias consecutivos.
- Número mínimo de dias consecutivos.
- Número de dias livre.
- Dia livre depois de fazer turno da noite.
- Número máximo de turnos por dia.
- Número máximo de turnos por semana.

2.3 Estrutura de dados

Para podermos resolver este problema, tivemos em consideração a maneira mais simples e completa de representar os dados do problema. Para alcançarmos uma estrutura eficiente foi desenvolvido uma base de dados, onde se centra no hospital em questão. O hospital tem todas as enfermeiras empregadas e todas as especialidades que suporta. Cada enfermeira só pode estar associada com uma especialidade e contém um conjunto de restrições associadas, como podem ser observadas no capítulo 2.2. Também de lembrar que cada hospital tem como predefinição as *hard constraints* associadas. Quanto

à estrutura da solução final, consideramos alocação de enfermeiros para uma semana, onde cada dia contém três turnos: o turno da manha, da tarde e da noite. Cada horário irá corresponder a uma especialidade, havendo no total N horários, sendo N o número total de especialidades que o hospital suporta. Quando o algoritmo interage com esta estrutura , irá aplicá-la a cada especialidade. Com esta divisão por especialidade, conseguimos obter resultados simples e fáceis de interpretar. (a verificar se falta mais algum detalhe e verificar texto na sua globalidade)

3 Algoritmos Implementados

3.1 Descrição das soluções

As soluções escolhidas tiveram em conta as conclusões apresentadas no livro, sendo os mais eficientes o Pesquisa Tabu Híbrida, Algoritmo Memetico e Switch. Todas as soluções usam uma solução inicial aleatória. Cada algoritmo é corrido uma vez por cada Especialidade de um Hospital.

- **Pesquisa Tabu Híbrida**: Esta solução gera horários a partir de mudanças da solução inicial usando a Pesquisa Tabu com movimentos específicos do problema.
- **Algoritmo Memético**: Algoritmo Memético usa uma pesquisa local a uma população de horários criados aleatoriamente, tal como a solução inicial, seguido da aplicação do Algoritmo Genético.
- **Switch**: Junção da Pesquisa Tabu Híbrida com Algoritmo Genético.

3.1.1 Solução Inicial

Para criar a primeira solução viável, é verificado para cada especialidade que cada turno é preenchido com um enfermeiro dessa especialidade de maneira aleatória.

Após obter um horário inicial para cada especialidade são invocadas as meta heurísticas.

3.1.2 Pesquisa Local

Pesquisa uma população de horários aleatórios de maneira igual

3.1.3 Pesquisa Tabu

A Pesquisa Tabu usa um conjunto de classes para processar a solução e criar uma população com a melhor pontuação possível após um número de iterações.

Classes

- **Solution** Estrutura que será processada e irá gerar a população de Horários. É constituída por uma especialidade e um valor. Esse valor representa a qualidade do horário da especialidade.
- **ObjectiveFunction** Função que avalia a *Solution*. Pode avaliar o resultado de uma solução com um Move sem o realizar. O resultado da Solução é avaliado segundo o Algoritmo 1.
- **Moves** Efetua os movimentos escolhidos para gerar uma nova solução. Esses *moves* podem ser a troca de turnos de enfermeiros ou substituição de um enfermeiro para um dado turno como mostra o algoritmo 1.
- **MoveManager** Calcula todos os movimentos possíveis. Rejeita movimentos que violem hard constraints ou movimentos redundantes (um enfermeiro trocar por outro turno seu). O algoritmo 1 mostra como é feita essa filtragem.
- **TabuList** Guarda a lista de movimentos que são considerados Tabu, estando proibidos de serem efetuados. Estes são selecionados baseado nos últimos movimentos efetuados. O tamanho da lista é de 10. A identificação de um movimento é feita por um hashcode calculado da seguinte forma:

  $\text{Hash} = Day \times 10 + Shift$

- **TSearch** Classe que inicia o algoritmo e guarda a melhor solução encontrada no horário da Especialidade.

Algoritmo 1 Pesquisa Tabu

ObjectiveFunction

$score = 0$
while $N < Soluton \leftarrow getConstraints()$ **do**

```
    if Solution← isConstraintVerified(N) then
      score =score + Solution← getConstraintValue(N)
    end if
    N++
  end while
  return score
```

Moves

```
  if moveType ==shift then
    Solution ← getSchedule()[day1][shift1] ← remove(nurse)
    Solution ← getSchedule()[day1][shift1] ← add(nurse2)
    Solution ← getSchedule()[day2][shift2] ← remove(nurse2)
    Solution ← getSchedule()[day2][shift2] ← add(nurse)
  else
    Solution ← getSchedule()[day][shift] ← remove(nurse)
    Solution ← getSchedule()[day][shift] ← add(nurse2)
  end if
```

MoveManager

```
  {troca de turnos entre enfermeiros}
  for each shift do
    nurse = Specialty ← getNurseAt(shift)
    for l=0; l < NumofDays;l++ do
      for h=0; h < NumofShifts;h++ do
        if !Specialty ← shiftEmpty(l, h) then
          nurse2 = Specialty ← getSchedule() ← getAssignment(l, h)[0]
          moves.add(nurse, shift, nurse2, h)
        end if
      end for
    end for
  end for
  {substituição de enfermeiros}
  for each shift do
    nurse = Specialty ← getNurseAt(shift)
    for j=0; j < Specialty ← getNurses();j++ do
      nurse2 = Specialty ← getNurses()[j]
      moves.add(nurse, nurse2, shift)
```

```
    end for
  end for
  return moves
```

3.1.4 Algoritmo Genético

Com a população inicial gerada, o algoritmo genético seleciona os dois melhores resultados de dois subgrupos da população inicial gerada e evolui a população cruzando os melhores resultados, criando mutações e guardando os melhores resultados da população anterior. O algoritmo irá evoluir durante um número de iterações ou até chegar à solução ótima (solução que respeite todas as *constraints*).

Classes

- **Algorithm**: Classe onde é processado o algoritmo. Formado por três grandes componentes: o *createOffspring*, *crossOver* e *Mutate* (Algoritmo 2).
- **GenPop**: Classe invocada pelo programa para iterar o algoritmo e para devolver a solução.

Algoritmo 2 Algoritmo Genético

```
                    Algorithm
{Função que gera a nova população}
Population = createOffspring(Population)
newPopulation = Population
if elitism then
  saveSchedule(newPopulation, getBest(Population))
end if

{Crossover}
for newSchedule in newPopulation do
  schedule1 = tournamentSelection(Population)
  schedule2 = tournamentSelection(Population)
  for shift in newSchedule do
    if random ≤ uniformRate then
      newSchedule ← shift = schedule1 ← shift
```

```
      else
        newSchedule ← shift = schedule2 ← shift
      end if
    end for
    saveSchedule(newPopulation, newSchedule)
  end for

  {Mutation}
  for newSchedule in newPopulation do
    for shift in newSchedule do
      if random ≤ mutationRate then
        nurse = newSchedule ← getNurses()[random]
        switchNurseinShift(shift, nurse)
      end if
    end for
  end for
  while numofIterations or foundOptimalSOlution() do
  end while
```

4 Análise de Dados

Foram testados os três algoritmos com a mesma base de dados, sendo esta constituída por 7 especialidades, cada uma com um número diferente de enfermeiros, vagas e restrições. Todos os algoritmos foram executados em 4 vezes, com iterações de Pesquisa Tabu e Algoritmo Genético diferentes:

1. 10 Iterações
2. 25 Iterações
3. 50 Iterações
4. 100 Iterações

A Tabela 1 e Figura 1 representam os resultados obtidos de cada algoritmo para cada especialidade e o tempo de execução.

	Spec 1	Spec 2	Spec 3	Spec 4	Spec 5	Spec 6	Spec 7	$\bar{x}$	RunTime(s)
TS1	297	87	209	92	170	256	84	170.3	7.2
Mem1	262	90	209	87	159	251	84	163.1	4.13

Switch1	302	97	209	92	161	256	84	171.5	8.9
TS2	315	105	209	92	178	256	84	176.9	17.8
Mem2	289	100	209	84	157	251	84	167.7	13.3
Switch2	309	103	209	92	178	256	84	175.9	22.5
TS3	317	100	209	92	178	256	84	176.6	36.5
Mem3	287	95	209	92	157	251	84	167.9	29.4
Switch3	325	103	209	92	173	256	84	177.4	48.2
TS4	320	108	209	92	178	256	84	178.1	71.2
Mem4	287	100	209	92	160	251	84	169.0	63.75
Switch4	320	108	209	92	181	256	84	178.6	96.3

Tabela 1: Resultados obtidos através da execução dos testes aplicados ao mesmo problema.

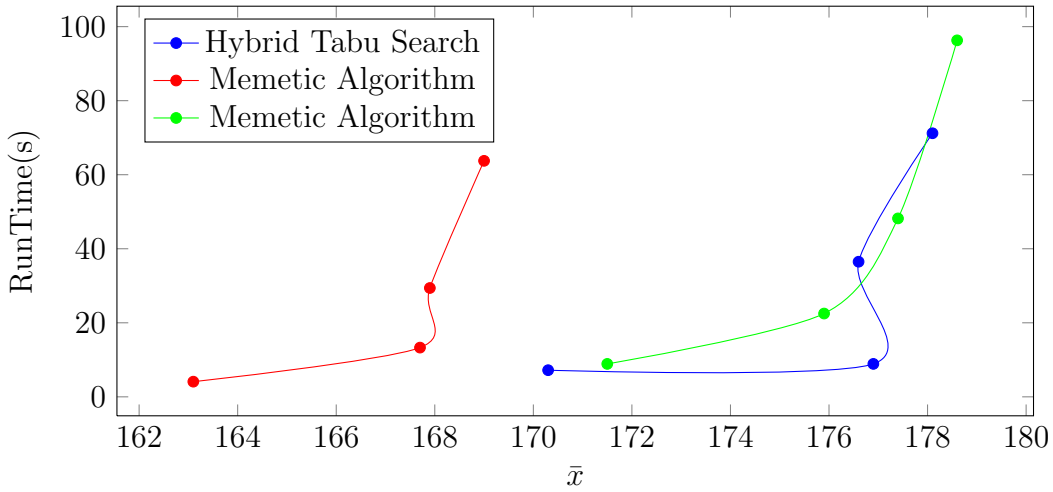

Figura 1: Comparação dos vários tempos atingidos por cada algoritmo com o valor médio alcançado

5 Conclusões e Análise de Resultados

Após realizado vários testes e análise aos dados obtidos, podemos concluir que, dependendo das variáveis a ter em conta, o algoritmo ótimo para construção de horários nunca é o mesmo.

Em suma, quando se trata de obter um resultado suficiente e rápido, o

Algoritmo Memético revela ser o mais eficiente a produzir resultados nestas condições. Para uma solução com o maior número de restrições satisfeitas, o Tabu Search demonstra ser o mais eficiente a reproduzir tais resultados, porém demora tempo a terminar a execução do algoritmo. Por fim, o melhor algoritmo, mas mais lento, é o Switch, produzindo resultados marginalmente superiores. Para todos os algoritmos, à medida que e aumentado o número de iterações, aumenta também o tempo de execução e melhora o resultado da solução.

Também podemos observar que, se todas as restrições forem satisfeitas para um determinado problema, tanto no Tabu Search como para o Memetic Algorithm, o tempo de execução torna-se muito semelhante entre os algoritmos. Isto é visível na Especialidade 3 e 7 onde, independentemente do número de iterações, o resultado é idêntico em todos os algoritmos. Devido ao tamanho da amostra, estes resultados devem ser associados a uma grande margem de erro, pois é provável que os resultados variem com condições como a grandeza do hospital e das restrições. Para estes resultados, foi construída uma framework para a criação da base de dados e a geração de horários com a possibilidade de escolha de parâmetros. Desta forma tornou-se fácil a manipulação e geração de novos dados. Concluindo, O planeamento de horários para funcionários de hospitais é bastante complexo, como é visível através das variáveis envolvidas. Através do estudo deste problema por Berghe, Causmaecker, Burke [2][3] e muitos outros, é possível obter resultados eficientes com a automatização da geração de horários, tornando esta uma opção viável face a criação manual.

Referências

[1] E. K. Burke, P. Causmaecker, G. V. Berghe: *Novel Metaheuristic Approaches to Nurse Rostering Problems in Belgian Hospitals*. CHAPMAN & HALL/CRC,2004.

[2] G. V. Berghe: *Novel Metaheuristic Approaches to Nurse Rostering Problems in Belgian Hospitals*. 2003.

[3] E. K. Burke, P. Causmaecker, P. Cowling G. V. Berghe: *A Memetic Approach to the Nurse Rostering Problem*. Applied Intelligence 15, 199–214, 2001.