METAHEURÍSTICAS GRASP, ILS E *ITERATED GREEDY* APLICADAS A PROBLEMAS DE AGENDAMENTO DE CIRURGIAS ELETIVAS EM HOSPITAIS DE GRANDE PORTE

Giselle Paranhos de Andrade[1], **Eduardo Camargo de Siqueira**[1], **Sérgio Ricardo de Souza**[1]

[1]Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Av. Amazonas, 7675, CEP 30510-000, Belo Horizonte (MG), Brasil

giselle@dppg.cefetmg.br, eduardosiqueira@dppg.cefetmg.br,

sergio@dppg.cefetmg.br

Resumo. *Neste artigo é tratado o Problema de Agendamento de Cirurgias Eletivas (PACE). Este problema consiste em alocar recursos hospitalares para a realização de cirurgias eletivas. O estudo visa promover um sistema gerencial que tenha, como objetivo, a utilização de recursos hospitalares de forma eficiente, visando à redução de custos nos centros cirúrgicos. O PACE será tratado como um Problema de Programação em Máquinas Paralelas Idênticas com Tempos de Preparação Dependentes da Sequência, uma classe de Problemas de Sequenciamento (Scheduling Problem). Para resolução do PACE foram implementadas as metaheurísticas GRASP, ILS e IG utilizando quatro instâncias reais de hospitais de grande porte, localizados na região metropolitana de Belo Horizonte, MG. Os resultados alcançados mostram a eficiência da metaheurística IG para a resolução de problemas desta natureza. Nota-se ainda que a fase de busca local, dos Algoritmos GRASP e ILS, aumentam o custo computacional dos algoritmos, sem necessariamente garantir a melhora da solução encontrada durante a fase de construção.*

PALAVRAS-CHAVE: *Problema de Agendamento de Cirurgias Eletivas, Metaheurísticas, Otimização*

Abstract. *This paper is concerned with the problem of Surgical Case Scheduling Problem (SCSP). This problem consists in allocating hospital resources to perform elective surgery. This study aims to promote a management system that has, as objective, the use of hospital resources efficiently, for reducing costs in Surgical Centers. The SCSP will be treated as a Identical Parallel Machines Scheduling Problem with Sequence Dependent Setup Time, a special class of Scheduling Problem. For the problem resolution, GRASP, IG and ILS metaheuristics were implemented and tested using four real instances of large hospitals, located in the metropolitan region of Belo Horizonte, MG. The results show the efficacy of IG metaheuristic to solve such problems. Note also that the local search phase increase the computational cost of algorithms, but not necessarily leading the improvement of the solution found during the construction phase. .*

KEYWORDS: *Surgical Case Scheduling Problem, Metaheuristics, Otimization*

1 Introdução

O gerenciamento da prestação de serviços de saúde nos hospitais está se tornando cada vez mais importante. Hospitais querem, por um lado, reduzir custos e melhorar os seus ativos financeiros e, por outro lado, querem elevar o nível de satisfação do paciente. Uma unidade de particular interesse é o Centro Cirúrgico (CC), ou melhor, a utilização das salas cirúrgicas, uma vez que o CC é a unidade responsável pelo maior custo do hospital e, segundo Macario et al. (1995), é o principal centro de receita da instituição.

Os fatores mencionados acima são tratados no Problema de Agendamento de Salas Cirúrgicas (*Surgical Case Scheduling* - SCS), problema que consiste em alocar recursos hospitalares para casos cirúrgicos e que também define o melhor instante de realização das cirurgias. Esta tarefa tem um papel decisivo na utilização de recursos hospitalares de forma eficiente, segundo Carter e Tovey (1992). O Problema de SCS é considerado, na literatura, um problema clássico de otimização combinatória, pertencente à classe NP-Difícil, segundo Carter e Tovey (1992). Logo, as técnicas heurísticas e as metaheurísticas, de maneira geral, tem sido largamente utilizadas na resolução de problemas desta natureza.

Doravante, denotar-se-á o SCS como PACE (Problema de Agendamento de Cirurgias Eletivas), que consiste em agendar cirurgias previamente conhecidas, desconsiderando casos de cirurgias de emergência. Para a resolução do PACE, é programado um sequenciamento de cirurgias, estabelecendo uma agenda cirúrgica, com o objetivo de minimizar o *makespan*, ou seja, minimizar o horário de término da última cirurgia.

O restante deste trabalho está organizado como segue. Na seção 2, é apresentada a revisão bibliográfica para o problema. Na seção 3, desenvolve-se a caracterização e definição do problema. Na seção 4, apresenta-se a metodologia adotada para estrutura de dados utilizada. Na seção 5, os algoritmos implementados são detalhados. Na seção 6, são expostos os resultados computacionais. E, por último, a seção 7 apresenta-se a conclusão do trabalho.

2 Revisão Bibliográfica

Segundo Inês Proença (2010), o processo de planejamento de cirurgias eletivas pode ser dividido em três fases: Planejamento de Casos Mistos *(Case Mix Planning)*; Planejamento Mestre de Cirurgias *(Master Surgery Planning)*; e Agendamento de Casos Eletivos *(Elective Case Scheduling)*.

A fase de Planejamento de Casos Mistos *(Case Mix Planning)* analisa a disponibilidade, em horas das salas cirúrgicas, distribuída pelos diferentes cirurgiões ou equipes cirúrgicas. Está situada em um nível estratégico de decisão e, geralmente, é realizada anualmente. A distribuição do tempo pode ter em conta a capacidade operativa de cada cirurgião ou de cada grupo cirúrgico e a quantidade esperada de pacientes ao longo do correspondente horizonte temporal. Hughes e Soliman (1978) e Blake e Carter (2002) apresentam diferentes abordagens para esta fase do planejamento.

A fase de Planejamento Mestre de Cirurgias *(Master Surgery Planning)* envolve o desenvolvimento de uma agenda cirúrgica. Trata-se de um documento cíclico, que define o número e o tipo de salas de operações disponíveis, as horas em que as salas estão abertas, definindo, ainda, cirurgiões ou grupos de cirurgias que têm prioridade sobre o tempo das salas cirúrgicas. Esta fase enquadra-se em um nível tático da gestão hospitalar. O horizonte

temporal nesta fase do planejamento é mais reduzido do que na primeira fase. Blake e Carter (2002) e Belien e Demeulemeester (2007) propõem uma série de modelos para a construção de agendamentos de cirurgias para esta fase.

Já na fase de Agendamento de Casos Eletivos *(Elective Case Scheduling)* é estabelecido o agendamento de cada cirurgia em uma base diária. Esta fase situa-se em um nível operacional. Trabalhos relativos a esta fase são encontrados em Magerlein e Martin (1978), Przasnyski (1986), Ozkarahan (1995) Kharrajal et al. (2006) e Cardoen et al. (2010).

Pham e Klinkert (2008) apresentam um modelo em programação linear inteira mista, baseado em uma extensão do *Job Shop Scheduling Problem*, denominada *Multi-Mode Blocking Job Shop*. O modelo define um período de início para cada uma das três fases necessárias na realização de uma cirurgia (pré-operatório, operatório e pós-operatório) e aloca, para cada uma das três fases, um conjunto de recursos necessários. Com técnicas de bloqueio, os autores apresentam uma solução para o problema de restrições de disponibilidade dos equipamentos. É possível, através do algoritmo apresentado, bloquear os recursos indisponíveis no momento da cirurgia. O objetivo é minimizar o período de início da última cirurgia a ser realizada.

3 Caracterização do problema de agendamento de cirurgias eletivas (PACE)

Nesta seção é abordada a descrição do Problema de Agendamento de Cirurgias Eletivas PACE, o qual será tratado neste trabalho, como um Problema de Programação em Máquinas Paralelas Idênticas com Tempos de Preparação de Máquina dependente da Sequência *(Identical Parallel Machine Scheduling Problem with Sequence Dependent Setup Times)*. Esta é uma classe particular de problemas de sequenciamento *(Scheduling Problem)*.

Neste problema, tem-se um conjunto $N = \{1, ...n\}$ de n tarefas e um conjunto $M = \{1, ..., m\}$ de m máquinas idênticas, com as seguintes características: (a) cada tarefa deve ser processada exatamente uma vez por apenas uma máquina; (b) cada tarefa j possui um tempo de processamento p_j ; (c) existem tempos de preparação s_{jk} entre as tarefas, em que as tarefas j e k serão processadas, nesta ordem. Estes tempos de preparação são dependentes da sequência. O objetivo é encontrar um sequenciamento das n tarefas nas m máquinas de forma a minimizar o tempo de conclusão do sequenciamento, o chamado *makespan* ou C_{max}. Pelas características citadas, este problema é definido como $P|Sjk|C_{max}$, segundo Pinedo (2008).

Com o objetivo de solucionar o PACE usando as características do problema de programação em máquinas paralelas, será utilizada a equivalência entre máquina e sala cirúrgica; e entre tarefa e cirurgia. O exemplo a seguir ilustra essa correspondência. A Tabela 1 contém os tempos de processamento de sete cirurgias realizadas em duas salas cirúrgicas. Na Tabela 2 estão contidos os tempos de preparação para as salas. A Figura 1 ilustra um possível sequenciamento para este exemplo.

Tabela 1. Tempos de Processamento das cirurgias

j	1	2	3	4	5	6	7
p_j	20	25	15	32	38	23	65

Na Figura 1, observa-se que a cirurgia 6 é alocada à sala S2 na terceira posição, com tempo de processamento p6 = 23, tendo a cirurgia 4 como predecessora e a cirurgia

Tabela 2. Tempos de Preparação (*setup*) das cirurgias

S	1	2	3	4	5	6	7
1	0	1	6	5	10	3	6
2	4	0	6	2	7	7	4
3	7	3	0	6	8	1	3
4	3	8	1	0	12	10	2
5	8	3	5	7	0	4	7
6	8	8	5	4	9	0	9
7	1	4	6	1	5	6	0

3 como sucessora. As partes hachuradas da figura representam os tempos de preparação (higienização) entre as cirurgias. Assim, neste exemplo, são computados os tempos s46 = 10 e s63 = 5. O tempo de conclusão das cirurgias na sala S1 é 120 e o da sala S2 é 130, o que resulta em um *makespan* de 130 unidades de tempo.

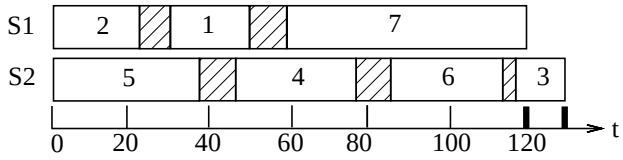


Figura 1. Exemplo de um possível sequenciamento.

No primeiro momento, é necessário listar todos os recursos necessários para a realização das cirurgias, como a disponibilidade de recursos humanos (cirurgiões, enfermeiros e anestesistas), além dos recursos materiais (salas cirúrgicas, de recuperação pós-anestésica - RPA - e unidade de tratamento intensivo - UTI) e dos equipamentos diversos, conforme Tabela 3, presente na seção 5.

A Figura 2 mostra, por exemplo, que a cirurgia1 é realizada na sala S_2, com a participação do cirurgião C_3, enfermeiro E_2 e anestesista A_2 utilizando o equipamento Eq_2, e que após a cirurgia o paciente se recupera na sala RPA; Já a cirurgia2 é realizada na sala S_1, com a participação cirurgião C_1, enfermeiro E_1 e anestesista A_1 utilizando o equipamento Eq_1, e que após a cirurgia o paciente também se recupera na sala RPA. Na Figura

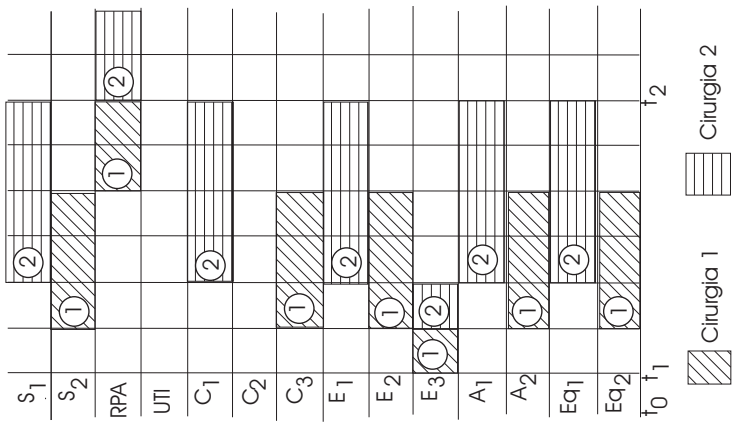
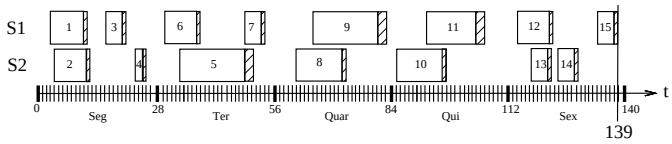


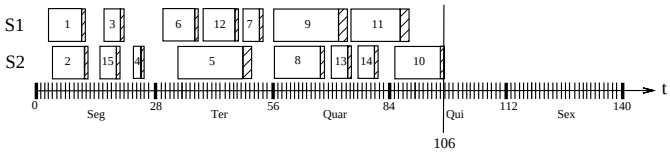
Figura 2. Exemplo de Gráfico de Gantt mostrando a realização de duas cirurgias com todos os recursos necessários.

3(a) é apresentada uma solução inicial de um agendamento semanal, contemplando 5 dias (segunda à sexta-feira), entre 07hs às 19hs. O período disponível para o agendamento

corresponde a 12 horas diárias e mais 2 horas, pois a última cirurgia pode iniciar-se às 19hs e ter duração máxima de 2 horas, totalizando, então 14hs de agendamento diários, ou seja, 70hs semanais. O período disponível para o agendamento de *slots* de 30min, ou seja, uma cirurgia com duração de 2hs possui 4 *slots*. Considerando que para cada dia são disponibilizados 28 *slots*, tem-se então um total de 140 *slots* para o agendamento semanal de cirurgias. Esta figura ilustra um agendamento contemplando 15 cirurgias realizadas em cinco dias da semana. Os *slots* hachurados representam a duração de higienização das salas, que podem variar de um a dois *slots*, de acordo com a duração e especialização da cirurgia. O *makespan* corresponde ao instante de término da última cirurgia, no caso, a cirurgia de número 15, que foi concluída no *slot* 139. A Figura 3(b) apresenta uma solução



(a) Exemplo de uma solução inicial do PACE.



(b) Exemplo de uma solução final do PACE

final para o agendamento semanal apresentado na Figura 3(a). Observe que todas as cirurgias agendadas para a sala S1 durante a sexta-feira foram realocadas para outros dias da semana; que é possível realocar a cirurgia de número 15, agendada para a sala S1 durante a sexta-feira, para a sala S2 na segunda-feira; que as cirurgias 13 e 14, agendadas para sala S2, podem ser agendadas para quarta-feira nesta mesma sala. Com estas realocações, o *makespan* diminui para 106 slots.

4 Metodologia

Esta seção apresenta os procedimentos propostos para a solução do Problema de Agendamento de Cirurgias Eletivas (PACE). Inicialmente, é mostrada a estrutura de dados utilizada para a representação de uma solução. Em seguida, são apresentadas as três metaheurísticas implementadas, a saber, *Greedy Randomized Adaptative Search Procedure* (GRASP), *Iterated Local Search* (ILS) e *Iterated Greedy* (IG). Os Algoritmos 1, 2, 3, 4, 5 e 6 mostram os seus pseudocódigos.

4.1 Representação da Solução

A solução é representada através de dois vetores. O primeiro vetor (seq) representa a sequência em que as cirurgias devem ser realizadas, enquanto que o segundo vetor (sala) representa em quais das salas a cirurgia será realizada, visto que cada cirurgia pode ser realizada em mais de uma sala. A Figura 3 representa a solução para o exemplo apresentado na seção 3.

4.2 Estruturas de Vizinhança

Para a implementação das metaheurísticas GRASP e ILS, a vizinhança da solução é explorada através dos seguintes movimentos:

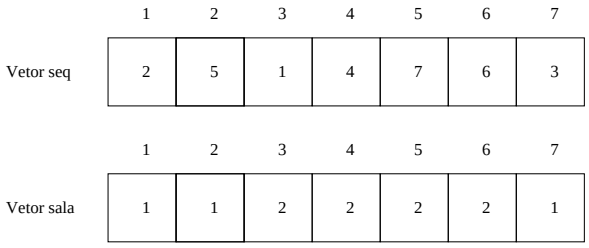


Figura 3. Representação da solução através de dois vetores.

.

i) Troca de cirurgias na mesma sala: consiste em trocar a posição de duas cirurgias na mesma sala;
ii) Realocação de cirurgias na mesma sala: consiste em realocar uma cirurgia para uma nova posição na mesma sala;
iii) Troca de cirurgias em salas distintas: consiste em considerar duas cirurgias e trocar as salas que as executam;
iv) Realocação de cirurgias em salas distintas: consiste em considerar duas cirurgias e realocá-las para uma nova posição em outra sala.
v) Troca em bloco de cirurgias na mesma sala: consiste em trocar a posição de seis cirurgias na mesma sala;
vi) Realocação em bloco de cirurgias na mesma sala: consiste em realocar três cirurgias para uma nova posição na mesma sala;
vii) Troca em bloco de cirurgias em salas distintas: consiste em considerar seis cirurgias e trocar as salas que as executam;
viii) Realocação em bloco de cirurgias em salas distintas: consiste em considerar seis cirurgias e realocá-las para uma nova posição em outra sala.

5 Metaheurísticas Implementadas

5.1 *Greedy Randomized Adaptative Search Procedure (GRASP)*

A metaheurística *Greedy Randomized Adaptative Search Procedure (GRASP)*, proposta por Feo e Resende (1995), é um processo iterativo no qual cada iteração consiste de duas fases: (i) fase construtiva, que gera soluções factíveis para o problema; e (ii) fase de busca local, que busca o ótimo local na vizinhança das soluções iniciais, geradas pela fase de construção.

Na fase construtiva, uma função denominada `construçãoGRASP` é empregada, gerando uma Lista de Candidatos (LC). Tais candidatos são avaliados segundo um critério em que o tempo necessário para a realização de uma cirurgia seja maior ou igual à diferença entre T_{max} e $\alpha(T_{max} - T_{min})$, sendo α um parâmetro real variando entre 0 e 1. Para α = 1, tem-se uma solução totalmente aleatória; para α = 0, tem-se uma solução gulosa. Os candidatos que atendem a esta condição compõem a Lista Restrita de Candidatos (LRC). Logo, o algoritmo irá escolher aleatoriamente um dos candidatos da LRC, sequenciando-o e retirando-o da LC. Este passo se repete até que todas as tarefas sejam sequenciadas.

Para a segunda fase ou fase de busca local foi utilizado o método de Descida Aleatória. Este método analisa parte da vizinhança da solução corrente. Os movimentos e vizinhos são escolhidos aleatoriamente. No final, efetua-se uma comparação entre a melhor solução encontrada e a solução corrente. Havendo melhora, a solução corrente é atualizada.

O pseudocódigo do GRASP é mostrado no Algoritmo 1. Na linha 3, uma solução é construída de forma parcialmente gulosa com o método chamado `construcaoGRASP`, que consiste em escolher aleatoriamente a próxima cirurgia a entrar no sequenciamento, dentre uma lista de melhores candidatos. Essa lista é montada de acordo com a regra de menores tempo de execução. O parâmetro α define o tamanho da lista. Na linha 4 é feito um refinamento na solução através de busca local. A melhor solução é atualizada se houver melhora (linha 5 a 7). Esses passos são repetidos até que um critério de parada seja atendido.

Algoritmo 1: Pseudocódigo GRASP

Entrada: α, criterioParada

```
1. início
2.    repita
3.       s ← construcaoGRASP(α);
4.       s' ← buscaLocal(s);
5.       se fo(s') melhor que fo(s*) então
6.          s* ← s';
7.       fim se
8.    até criterioParada seja satisfeito;
9. fim
```

A busca local utilizada como heurística de refinamento foi o método Descida Aleatória, em que somente parte da vizinhança é analisada. A cada iteração um movimento é escolhido aleatoriamente e aplicado sobre a solução corrente. Se houver melhora a solução corrente é atualizada. O Algoritmo 2 ilustra seu funcionamento.

Algoritmo 2: Pseudocódigo Busca Local Descida Aleatória

Entrada: s, maxIter

```
1. início
2.    r ← Quantidade de movimentos (no caso, r = 4);
3.    Para i ← 1 até maxIter faça
4.       s' ← seleciona aleatoriamente um vizinho (dentre as r vizinhanças);
5.       se s' melhor que s então
6.          s ← s';
7.       fim
8.    fim
9.    retorna s;
10. fim
```

5.2 *Iterated Local Search* (ILS)

A meta-heurística ILS *(Iterated Local Search)*, segundo Lourenço et al. (2003), é um método de busca iterativa que faz uso de pertubações da solução (alterações na solução corrente), tendo, como principal objetivo, a diversificação da busca, de modo a escapar e visitar diferentes ótimos locais. Quatro são os principais componentes que definem o método ILS: geração da solução inicial, busca local, pertubação e critério de aceitação. O ILS baseia-se na idéia de aplicar uma busca local em uma solução inicial qualquer até que se encontre um ótimo local e, então, perturbar a solução encontrada e reiniciar a busca local. A perturbação deve ser de tal forma que seja suficiente para escapar de um ótimo local e permitir a exploração de outras regiões do espaço de buscas. O método ILS é, portanto, um método de busca local que procura focar a busca não no espaço completo de

soluções, mas em um pequeno subespaço definido por soluções que são ótimos locais de determinado procedimento de otimização.

O pseudocódigo do ILS encontra-se descrito no Algoritmo 3. Na linha 2 é construída uma solução com o método `construçãoGRASP`. Na linha 3 a solução é refinada, através de busca local, utilizando o Método de Descida Aleatória. Na linha 6 é realizada uma perturbação na solução atual, o nível de pertubação vai sendo aumentado a cada iteração sem melhora. O Algoritmo 4 descreve o pseudocódigo da função perturbação utilizada. Neste método considera-se o nível de perturbação a quantidade de vezes que um dos oito movimentos será aplicado. Após a perturbação, essa nova solução é refinada, conforme Algoritmo 3 (linha 7). Se houver melhora na solução, ela é atualizada e o nível de perturbação reinicializado. Esses passos são repetidos até que um critério de parada seja atendido. Como critério de parada, foi adotado o número de 100 iterações sem melhora, ou seja, o método para quando a melhor solução se mantém inalterada por 100 iterações.

Algoritmo 3: Pseudocódigo ILS

```
Entrada: α, criterioParada
1.  início
2.    s ← construcaoGRASP(α);
3.    s* ← buscaLocal(s);
4.    nivel ← 1;
5.    repita
6.      s ← perturbacao(s*, nivel);
7.      s' ← buscaLocal(s);
8.      se fo(s') melhor que fo(s*) então
9.        s* ← s';
10.       nivel ← 1;
11.     senao
12.       nivel ← nivel + 1;
13.     fim se
14.   até criterioParada seja satisfeito;
15. fim
```

5.2.1 Função Perturbação da Solução

A função perturbação (alterações na solução corrente) citada no Algoritmo 3 e descrita no Algoritmo 4 tem como principal objetivo a diversificação da busca local, de modo a escapar e visitar diferentes ótimos locais. O nível de perturbação define a quantidade de vezes que um dos movimentos, escolhido aleatoriamente, será realizado.

Algoritmo 4: Pseudocódigo Perturbação

```
Entrada: s, nivel
1.  início
2.    Para i ← 1 até nivel faça
3.      Escolhe um movimento aleatoriamente;
4.      s ← aplicaMovimento();
5.    fim
6.    retorna s;
7.  fim
```

5.3 *Iterated Greedy Search*

Iterated Greedy Search (IG) é um método heurístico baseado em um princípio de construção e desconstrução de soluções, que tem se mostrado eficiente na solução de di-

versas classes de problemas de sequenciamento. Ruiz e Stutzle (2007) mostrou seu desempenho para um problema de programação *flowshop* de permutação em que o objetivo é minimizar o *makespan* (FSP-Cmax).

O Algoritmo 5 mostra o pseudocódigo do IG. Na linha 2 é construída a solução inicial com o método `construcaoGRASP`, já detalhada na seção 5.2. Na linha 3 é aplicado o método de busca local descrita no Algoritmo 6. Este método consiste em selecionar uma cirurgia da solução corrente e inseri-la na melhor posição possível da sequência. Se houver melhora, a solução corrente é atualizada. Esses passos são repetidos até que todas as cirurgias sejam consideradas. O algoritmo proposto aplica iterativamente duas fases. Na primeira fase, chamada de destruição, d (no caso d = 8) tarefas são retiradas da solução corrente e armazenadas no vetor π_R; já na fase de construção, as tarefas retiradas são reinseridas na melhor posição da sequência da solução corrente. Após este procedimento, a busca local é aplicada novamente. Se houver melhora na solução, então a solução corrente é atualizada, e se esta solução for a melhor solução encontrada, a melhor solução é atualizada. A solução corrente poderá ser atualizada também se o critério de aceitação (linha 21), dado pela expressão:

$$pr < exp\frac{-(C_{max}(\pi') - C_{max}(\pi))}{Temperatura} \quad (1)$$

for atendido. No critério de aceitação a, temperatura é calculada por:

$$Temperatura = T \cdot \frac{\sum_{i=1}^{n} p_i}{n} \quad (2)$$

em que T =0,5 é o parâmetro de entrada do algoritmo; p_i é o tempo de duração da cirurgia i; e n é a quantidade de cirurgias.

Algoritmo 5: Pseudocódigo IG

Entrada: $d, Temperatura, criterioParada$

```
1.  início
2.    π ← construcaoGRASP();
3.    π ← BuscaLocal(π);
4.    π_b ← π;
5.    repita
6.      π' ← π;
7.      para i ← 1 até d faça
8.        Retira uma cirurgia aleatoriamente de π' e insere em π_R;
9.      fim
10.     para i ← 1 até d faça
11.       Retira a cirurgia π_R(i) e insere na melhor posição possivel de π';
12.     fim
13.     π' ← BuscaLocal(π);
14.     se π' melhor que π então
15.       π ← π';
16.       se π' melhor que π_b então
17.         π_b ← π;
18.       fim
19.     senão
20.       pr ← Número aleatório entre 0 e 1;
21.       se pr < exp{−(C_max(π') − C_max(π))/Temperatura} então
22.         π ← π';
23.       fim
24.     fim
25.   até criterioParada ser satisfeito;
26. retorna π_b;
27. fim
```

Algoritmo 6: Pseudocódigo BuscaLocal Para IG

Entrada: π
1. **início**
2. $melhora \leftarrow Verdadeiro$;
3. **enquanto** $melhora$ **faça**
4. $melhora \leftarrow Falso$;
5. **para** $i \leftarrow 1$ **até** n **faça**
6. Retirar uma cirurgia k aleatória de π (sem repetição)
7. $\pi' \leftarrow$ insere a tarefa k na melhor posição possivel de π;
8. **se** π' melhor que π **então**
9. $\pi \leftarrow \pi'$;
10. $melhora \leftarrow Verdadeiro$;
11. **fim**
12. **fim**
13. **fim**
14. **retorna** π
15. **fim**

6 Resultados Computacionais

Os algoritmos GRASP e ILS foram implementados na linguagem C++, utilizando a IDE Borland C++ Builder, e testados em um computador Intel Core 2 Duo 2.10GHz e 3GB de memória RAM. Os parâmetros utilizados para gerar as instâncias foram obtidos em quatro hospitais de grande porte localizados na região metropolitana de Belo Horizonte, Minas Gerais, e estão detalhados na Tabela 3.

Tabela 3. Instâncias utilizadas e suas características

	H1(162x7)	H2(192x12)	H3(216x18)	H4(266x16)
Leitos	157	272	319	501
Cirurgias	162	192	216	266
Salas	7	12	18	16
Cirurgiões	74	112	126	157
Anestesistas	20	25	31	37
Enfermeiros	18	21	27	34
Equipamentos	21	27	31	30

Tabela 4. Resultados obtidos para cada Metaheurística, aplicada nas 4 instâncias reais de hospitais de grande porte.

Problemas Teste	Soluções encontradas pelo GRASP com ($K = 2$)						
	Fase de Construção	GRASP		ILS		IG	
	makespan	*makespan*		*makespan*		*makespan*	
	Melhor	Melhor	Médio	Melhor	Médio	Melhor	Médio
H1(162x7)	190	134	136,2	131	132,6	128	130,7
H2(192x12)	126	90	91,6	84	87,9	72	74,4
H3(216x18)	131	107	110,3	112	113,4	92	96,0
H4(266x16)	162	114	117,2	105	112,2	96	103,8

Os algoritmos foram executados 30 vezes cada, para cada uma das quatro instâncias. Na Tabela 4 são apresentadas as soluções iniciais obtidas na fase de construção do algoritmo GRASP, os melhores resultados alcançados e os valores médios de cada execução. Nota-se o melhor desempenho da heurística IG nas quatro instâncias testadas. Veja ainda que esta metaheurística consegue minimizar o *makespan* da instância H1(162x7) que inicialmente era de 190*slots* de tempo para apenas 128*slots*. Isto significa que o *makespan* foi minimizado em 32,63%. O algoritmo consegue alocar de maneira eficiente todos os recursos necessários (salas, cirurgiões, enfermeiros, anestesistas e equipamentos) para a

realização das cirurgias. Assim a garantia do agendamento é maior, pois todos os recursos estão devidamente alocados, respeitando-se as restrições de disponibilidade. Com isto, as salas disponíveis podem receber cirurgias de emergência ou até mesmo outras cirurgias eletivas.

Para as instâncias H2(192x12), H3(216x18) e H4(266x16) o *makespan* foi minimizado em 42,86%, 29,77% e 40,74%, respectivamente.

De forma a comparar os algoritmos com relação ao tempo necessário para encontrar um valor alvo, foi realizado experimentos segundo a abordagem indicada em (Aiex et al. (2002)). Por meio de gráficos TTTPlot *(Time-To-Target Plots)*, é possível verificar a probabilidade que cada algoritmo possui de alcançar os valores alvos em função do tempo de execução. Os algoritmos são executados 100 vezes e sempre que o alvo é alcançado, eles eram interrompidos e o tempo registrado. Esses tempos foram, então, ordenados de forma crescente e, para cada tempo t_i foi associada uma probabilidade p_i = (i - 0,5)/100.

A Figura 4 mostra o comportamento dos algoritmos GRASP, ILS e IG para a instância H1(162x7). Verifica-se o melhor desempenho da metaheurística IG, pois neste caso, alcançou o valor alvo em aproximadamente 22 segundos. Já o ILS alcançou o valor alvo em 42 segundos de execução e a metaherística GRASP apresentou para esta instância o pior comportamento, sendo necessários 120 segundos para alcançar o valor alvo.

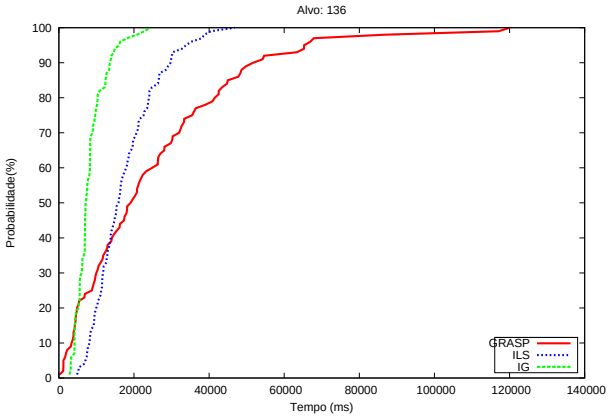


Figura 4. Time-to-target da instância H1 com alvo igual a 136.

7 Conclusão e Trabalhos Futuros

Este trabalho teve seu foco no Problema de Agendamento de Cirurgias Eletivas. O objetivo é minimizar o *makespan*, que compreende o instante de término da última tarefa. Para resolvê-lo, foram testadas as metaheurísticas GRASP, ILS e *Iterated Greedy*.

Através dos testes é comprovada a melhor eficiência da metaheurística IG em relação ao GRASP e o ILS, apresentando em média resultados 36,50% melhores quando comparados com a solução inicial.

Como continuação deste trabalho, serão efetuados testes dos algoritmos com outros tipos de movimentos.

Agradecimentos

Os autores agradecem ao CEFET-MG, à CAPES, à FAPEMIG e ao CNPq pelo apoio ao desenvolvimento deste trabalho.

Referências

Aiex, R. M.; Resende, M. G. C. e Ribeiro, C. C. (2002). Probability distribution of solution time in grasp: An experimental investigation. *Journal of Heuristics*, v. 8, p. 343–373.

Belien, J. e Demeulemeester, E. (2007). Building cyclic master surgery schedules with leveled resulting bed occupancy. *European Journal of Operational Research*, v. 2, n. 176, p. 1185–1204.

Blake, J.T. e Carter, Michael W. (2002). A goal programming approach to strategic resource allocation in acute care hospitals. *European Journal of Operational Research*, v. 140, p. 541–561.

Cardoen, B.; Demeulemeester, E. e Beliën, J. (2010). Operating room planning and scheduling: a literature review. *European Journal of Operational Research*, v. 3, p. 333–333.

Carter, M. W. e Tovey, C. A. (1992). When is the classroom assignment problem hard? *Operations Research*, v. 40, n. 1, p. 28–30.

Feo, T.A. e Resende, M.G.C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, p. 109–133.

Hughes, William L. e Soliman, Soliman Y. (1978). Short-term case mix management with linear programming. *Hospital & Health Services Administration*, v. 30, p. 52–60.

Kharrajal, S.; Albert, P. e Chaabanel, S. (2006). Bloco de programação para um calendário cirúrgico. *Anais de Conferência Internacional sobre Sistemas de Serviços e Sistemas de Gerenciamento*.

Lourenço, H.R.; Martin, O.C. e Stützle, T. (2003). Iterated local search. glover,g.f. e kochenberger, editor, handbook of metaheuristics. *Kluwer Academic Publishers, Boston*, p. 321–353.

Macario, A.; Vitez, T. S.; Dunn, B. e McDonald, T. (1995). Where are the costs in perioperative care?: Analysis of hospital costs and charges for inpatient surgical care. *Anesthesiology*, v. 83, n. 6, p. 1138–1144.

Magerlein, J. M. e Martin, J. B. (1978). Surgical demand scheduling: A review. *Health Services Research*, p. 418–433.

Ozkarahan, I. (1995). Allocation of surgical procedures to operating rooms. *Journal of Medical Systems*, v. 4, n. 19, p. 333–352.

Pham, D. N. e Klinkert, A. (2008). Surgical case scheduling as a generalized job shop-scheduling problem. v. 185, p. 1011–1025.

Pinedo, M. (2008). *Scheduling: theory, algorithms, and systems*. Springer Verlag.

Proença, Inês Marques. *Planejamento de Cirurgias Eletivas - Abordagens em Programação Inteira*. PhD thesis, Tese de Doutorado, Departamento de Estatística e Investigação Operacional, Lisboa, (2010).

Przasnyski, Zbigniew H. (1986). Operating room scheduling: a literature review. *AORN Journal*, v. 44, p. 67–79.

Ruiz, R. e Stutzle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, v. 177, p. 2033–2049.