INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Problemas de Escalonamento de Pessoal em Enfermarias Hospitalares

T. M. Dias *D. F. Ferber* *C. C. de Souza*
A. V. Moura

Technical Report - IC-02-02 - Relatório Técnico

February - 2002 - Fevereiro

Problemas de Escalonamento de Pessoal em Enfermarias Hospitalares

Tiago M. Dias[*] Daniel F. Ferber[†] Cid C. de Souza[‡] Arnaldo V. Moura[§]

Resumo

Nesse relatório técnico, heurísticas baseadas em algoritmos evolutivos (algoritmos genéticos) e técnicas de busca local (busca tabu) são aplicadas ao problema das escalas de serviço para as enfermarias do Hospital das Clínicas da Unicamp. Também é descrita uma ferramenta computacional para construir as escalas de serviço do hospital. Essa ferramenta compreende módulos resolvedores que implementam versões dessas heurísticas e também uma interface amigável para estimular o uso da ferramenta pelos administradores do hospital.

Keywords:
Otimização Combinatória, Algoritmo Genético, Busta Tabu, Escalonamento, Administração Hospitalar.

1 Introdução

A Diretoria de Enfermagem do Hospital das Clínicas (HC) opera aproximadamente 30 enfermarias especializadas que, juntas, compreendem um quadro de funcionários de aproximadamente 1500 profissionais, como enfermeiros, auxiliares e supervisores. Mensalmente, a diretoria desenvolve escalas de trabalho para o próximo mês, envolvendo todos os funcionários sob sua supervisão. As escalas estão sujeitas a uma série de restrições, tais como, entre outras: procurar atender as preferências pessoais dos funcionários por dias de folga, obeceder restrições trabalhistas e garantir um número mínimo de profissionais trabalhando em cada enfermaria, de modo a manter a qualidade do atendimento. Muitas dessas imposições são conflitantes entre si e, além disso, é conveniente que as escalas construídas satisfaçam a certos critérios de otimização, tais como minimizar o número de horas-extra usadas para suprir a falta de funcionários regulares.

Atualmente, a construção dessas escalas de serviço consome um número considerável de dias de planejamento na Diretoria da Enfermagem. Esse relatório descreve uma ferramenta computacional que pode ser usada para construir as escalas de serviço das enfermarias do HC de forma automática. O módulo resolvedor da ferramenta está baseado em duas

---

[*] Supported by FAPESP grant 00/11468-3
[†] Supported by FAPESP grant 00/11473-7
[‡] Supported by FINEP (ProNEx-107/97), and CNPq
[§] Supported by FAPESP grant 99/05999-4

heurísticas. Pode tanto utilizar-se de um algoritmo evolutivo, tipo genético, quanto pode empregar uma heurística de busca local, tipo tabu. A implementação dessas heurísticas é aqui descrita em detalhes. Vários resultados obtidos sobre instâncias reais do problema são apresentados e comparados.

Além de resolver o problema de otimização das escalas de trabalho, esse relatório descreve também uma uma interface de uso amigável para os usuários finais do *software*. Essa interface foi implementada para que os usuários do *software* operem com formulários eletrônicos simples, auto-explicativos e semelhantes aos formulários em papel hoje em uso. Essa concepção do *software* é muito importante para incentivar o seu uso pelos administradores do hospital, visto que a maioria dos usuários no HC possui pouca experiência no manuseio de computadores.

A primeira versão da ferramenta já foi instlada nas dependências do HC e a primeira turma, de aproximadamente 40 funcionários, já foi treinada no uso da ferramenta. Esses funcionários supervisionam todas as enfermarias do HC, inclusive o pronto-socorro e o ambulatório. Numa segunda etapa, a ferramenta será disponibilizada para uso por todas as enfermarias do HC.

As atividades descritas nesse relatório foram desenvolvidas como projetos de Iniciação Científica dos dois primeiros autores, sob a supervisão dos dois últimos.

2 O Problema das Escalas de Serviço no HC

Entre os mais comuns na gestão de atividades, o problema de escalonamento de horários de trabalho, nas suas inúmeras variações, está amplamente documentado na literatura [5, 3, 18, 12]. Por outro lado, não existe nenhuma solução geral disponível no mercado, em conseqüência das singularidades das restrições e dos critérios operacionais de cada problema. Cada caso, via de regra, requer o desenvolvimento de um modelo específico e de um projeto de algoritmo adaptado às condições deste modelo. Só então, em instâncias reais de grande porte, é possível contar com o uso de ferramentas computacionais automáticas para se buscar uma solução satisfatória.

O Hospital das Clínicas da Unicamp atende à população de Campinas e das cidades vizinhas. Além de oferecer atendimento gratuito e de qualidade a estas populações, permite aos estudantes de enfermagem e de medicina praticarem conhecimentos adquiridos em sala de aula. A Diretoria de Enfermagem do HC opera, aproximadamente, 30 enfermarias especializadas. Para tal, conta com cerca de 1500 profissionais entre supervisores, enfermeiros e auxiliares [1]. Mensalmente, este setor do HC deve construir uma escala de trabalho adequada para a operação do mês vindouro, de modo a atender à várias restrições, muitas delas conflitantes entre si, e ainda de modo a satisfazer às especificidades de cada enfermaria. Este é um problema de difícil solução. A magnitude do problema pode ser apreciada se imaginarmos uma matriz de zeros e uns, com cerca de 1500 linhas (uma para cada funcionário) e 30 colunas (uma para cada dia do mês), onde a cela (i, j) contém um 1 se o funcionário i está trabalhando no dia j. O número de possibilidades é enorme. A tarefa equivale a preencher esta matriz de forma a cumprir as várias restrições de cada enfermaria. A busca de uma solução adequada consome, no total, um número considerável de dias de trabalho das supervisões das enfermarias e da coordenação geral do setor de enfermagem do HC [1].

Nas enfermarias do HC, os plantões de serviço dividem-se em três turnos: manhã, tarde e noite. Os funcionários trabalham somente no turno para o qual foram contratados. Para cada turno está prevista uma necessidade mínima de funcionários em serviço, tanto enfermeiros quanto auxiliares de enfermagem, de forma a manter a qualidade do serviço. Este número pode variar entre dias úteis e fins de semana, conforme especificado pela coordenação da área de enfermagem do HC. Eventuais necessidades adicionais de profissionais devem ser preenchidas com o recurso da hora-extra, um expediente que, segundo a Direção do HC, deve manter-se no mínimo possível.

Devido a sua especificidade, a escala do plantão noturno é a mais simples e sua construção não oferece dificuldade, mesmo sob o aspecto de otimização. A escala é formada por um padrão fixo de 12 horas de trabalho alternadas com 60 horas de folga. Respeitada a disponibilidade de cada funcionário, a escala de serviço para o plantão noturno pode ser contruída sem maiores tropeços. Por ser mais simples, a construção da escala para atender aos plantões noturnos não foi explicitamente contemplada na primeira versão da ferramenta computacional e, portanto, não é descutida nesse relatório.

A partir de reuniões [1] com a Diretoria de Enfermagem do HC, e com base nos exemplos reais fornecidos pela mesma, foi criada uma especificação para o problema das escalas de serviço, cujos principais pontos são apresentados a seguir.

O problema de otimização precisa ser resolvido para cada mês e para cada enfermaria, em outras palavras, cada instância do problema corresponde a um mês. Para simplificar a abordagem, o problema foi dividido em vários sub-problemas menores, cada um formado por uma determinada enfermaria, determinado tipo de funcionário (enfermeiros ou auxiliares de enfermagem e técnicos) e determinado turno de trabalho (manhã ou tarde).

Cada enfermaria especifica, para cada turno, um número mínimo de funcionários de cada tipo que devem trabalhar em dias úteis e outro número mínimo, normalmente menor, para cada tipo de funcionário necessário nos fins de semana. Para maior flexibilidade, a interface da ferramenta computacional permite não só fornecer o número comum de funcionários para todos os dias úteis e para todos os fins de semana e feriados, como permite também estipular um número específico de funcionários para cada dia do mês. Caso não seja possível alocar um número suficiente de funcionários para certo dia, a solução deve recorrer a horas-extra para completar o quadro de funcionários em atividade naquele dia.

A escala ideal precisa satisfazer restrições relacionadas a cada funcionário. O profissional não deve ser escalado em dias nos quais está impedido de trabalhar por outros motivos, tais como licenças médicas ou férias. Nesta primeira versão da feramenta, os impedimentos de cada funcionário são marcados através da interface. Os módulos resolvedores não fazem distinção quanto a natureza do impedimento.

O número mínimo de folgas que deve ser atribuído a cada funcionário não é comum. Funcionários com impedimentos devem receber, dentro do intervalo menor de dias que estão trabalhando, um número de folgas menor que os demais funcionários que não estão impedidos no mesmo período. Por exemplo, se um funcionário encontra-se em licença médica durante 20 dias, deve receber, nesse mês, menor número de folgas que um outro funcionário que está disponível e trabalha por todo o mês. Uma fórmula fechada para calcular o número de folgas a serem atribuídas a cada funcionário envolveria muitos fatores subjetivos. Assim, preferiu-se deixar este dado como um parâmetro de entrada para cada funcionário. Esse parâmetro pode ser especificado através da interface de uso da ferramenta. Ainda assim, a ferramenta sugere valores a serem utilizados, calculando o número de folgas proporcionalmente.

Das folgas atribuídas a um funcionário, um certo número mínimo delas deve coincidir com fins de semana. Na ferramenta computacional, esse número mínimo pode ser especificado de forma particular para cada um dos funcionários a cada execução dos módulos resolvedores que constrõem as escalas de serviço. Uma outra característica desejável é distribuir os finais de semana de forma igualitária entre todos os funcionários.

Um número mínimo de dias de trabalho deve ocorrer entre duas folgas consecutivos, de modo a evitar alternâncias muito curtas entre dias de trabalho e dias de descanso. Esse número também é passível de especificação através da interface da ferramenta, e entra como parâmetro a cada execução dos módulos resolvedores. Com essa restrição, evita-se dias de trabalho e folga alternados.

As preferências dos funcionários por dias de folga são indicadas por eles mesmos e se constituem em parâmetros de entrada do problema. Cada funcionário tem direito a que um número mínimo das preferências que assinalou lhe seja atribuído como folgas durante o mês. Esse número é especificado de forma particular para cada funcionário, podendo ser alterado a cada execução dos módulos resolvedores. Se um determinado funcionário indicar mais preferências do que o mínimo que lhe deve ser atribuído, o algoritmo é livre para escolher dentre todas as preferências indicadas pelo funcionário e não será obrigado a lhe conceder as preferências que indicou em excesso, embora considere esse fato durante sua execução, privilegiando soluções que atendam preferências em excesso do mínimo.

Segundo a Diretoria de Enfermagem do HC, também é conveniente se considerar folgas de alta prioridade, que sempre deveriam ser atribuídas. Uma alternativa seria decretar-se impedimento nesses dias, garantindo que o funcionário não estaria de serviço. No entando, preferiu-se introduzir, além da preferência comum, uma nova preferência denominada "altamente desejável", e que será sempre atendida.

O padrão de folgas deve ser tal que nenhum funcionário trabalhe mais de seis dias consecutivos. Além do limite máximo de dias consecutivos de trabalho, é possivel também especificar um número mínimo de dias consecutivos de trabalho. Dessa forma, é possível evitar alternâncias entre dias de trabalho e dias folga, algo considerado inconveniente, como discutido acima.

Existe uma outra categoria de folga, denominada "em haver". São folgas concedidas ao funcionário para compensar horas-extra trabalhadas em outras atividades tais como atividades administrativas ou cursos de especialização. A diferenciação dessa folga na interface é meramente uma distinção visual para o usuário, pois o algoritmo a trata como impedimento, uma vez que esse tipo de folga pode ser considerada equivalente a impedimentos.

Alguns profissionais pertencem a uma categoria especial, os "plantonistas". Esses funcionários trabalham apenas em dias específicos, normalmente nos fins de semana, mas folgam em todos os outros dias. Como o padrão de escala de trabalho para esses funcionários está determinado a priori, eles não são considerados para efeitos de otimização. Porém, sua presença influencia o total de funcionários disponíveis em cada dia, e esse fator deve ser levado em conta pelos módulos resolvedores.

A escala ideal ainda possui outras características, como usar o mínimo possível de horas-extra. A ferramenta computacional permite estipular um limite máximo comum de horas-extra por dia, além de um limite superior para a soma de horas-extra para todo o mês.

O problema da construção das escalas de serviço consiste, então, em distribuir as folgas necessárias de tal maneira que, atendidas as preferências dos funcionários e satisfeitas as demais restrições, o número horas-extra atribuído seja mínimo. Em face à dificuldade do problema, já seria de grande utilidade a automação da construção das escalas de serviço, gerando escalas de qualidade semelhante àquelas construídas manualmente. A economia de tempo e esforço por parte da supervisão das enfermarias do HC já seria apreciável, justificando plenamente o uso da ferramental computacional.

Nesse relatório, usaremos como instâncias reais escalas de trabalho para uma versão reduzida do problema geral do HC, enfocando apenas as “Enfermarias de Retaguarda”, ou seja, a Emergência Clínica e a Cirurgia de Trauma, encarregadas dos tratamentos intensivos e semi-intensivos. Para essas enfermarias, disponibilidades e necessidades de pessoal típicas estão descritas na Tabela 1, para os períodos da manhã e da tarde. A ferramenta contruída, claro, pode perfeitamente atender também às demais enfermarias do HC, pois os problemas são muito similares.

	Dias úteis		Feriados	
	Manhã	Tarde	Manhã	Tarde
Enfermeiros	6/4	5/4	6/4	5/4
Auxiliares	13/9	12/9	12/8	11/8

Tabela 1: Disponibilidade/necessidade diária de profissionais nas enfermarias de retaguarda

3 Algoritmos Genéticos e Busca Tabu

As organizações dependentes do uso intensivo de mão de obra dispendem, periodicamente, consideráveis recursos para a montagem de escalas de trabalho. Tipicamente, para certo conjunto de funcionários, deseja-se produzir uma escala de trabalho de modo a satisfazer uma vasta gama de restrições de natureza variada [28, 7, 8, 6, 11]. A obtenção de uma solução ótima, segundo algum critério pré-definido, muitas vezes é uma exigência adicional na maioria das instituições e estabelecimentos comerciais. Por conseguinte, problemas desta natureza (problemas de *timetabling* e de *scheduling*) têm sido alvo de várias pesquisas ao longo do tempo, com objetivo de buscar algoritmos computacionais eficientes e viáveis para situações reais.

Em princípio, os algoritmos que resolvem problemas de otimização podem ser divididos em 3 grandes grupos: os algoritmos exatos, os algoritmos aproximados e as heurísticas. Um algoritmo exato é aquele que sempre produz uma solução de valor ótimo qualquer que seja a instância de entrada. Já um algoritmo aproximado, independentemente da entrada, gera uma solução cujo valor garantidamente está pelo menos a uma distância constante (absoluta ou percentual) do valor ótimo. Ocorre que uma vasta gama de problemas de otimização podem ser classificados como NP-difíceis [13]. Isso praticamente exclui algoritmos determinísticos eficientes para esses problemas, especialmente para instâncias de entrada suficientemente grandes para acomodar dados reais. Na ausência de algoritmos exatos ou aproximados, ou mesmo se esses algoritmos não forem adequados, uma alternativa é o uso de heurísticas eficazes que possam auxiliar no processo de busca de boas soluções. Heurísticas são algoritmos capazes de encontrar soluções viáveis mas para as quais não é possível oferecer qualquer garantia, a priori, do afastamento máximo da solução encontrada em relação ao valor ótimo.

A grosso modo, as heurísticas podem ser classificadas em dois grandes grupos: as construtivas e as de melhoria ou de busca local . As heurísticas construtivas são aquelas em que, a partir de uma solução vazia, o algoritmo vai iterativamente escolhendo e adicionando um novo elemento a solução corrente até que esta se torna viável. Já as heurísticas de melhoria são aquelas em que o algoritmo, partindo das soluções atuais de que dispõe, vai migrando para soluções melhores até que um determinado critério de parada seja atingido. Nesse relatório, exploraremos duas heurísticas para auxiliar na busca de soluções para o problema das escalas de serviço no HC: algoritmos genéticos e busta tabu.

Algoritmos evolutivos, ou algoritmos genéticos [9, 16], oferecem uma técnica eficiente para resolver problemas de otimização combinatória em geral. Essa classe de algoritmos é, na realidade, uma família uniforme de heurísticas que acomodam uma ampla gama de funções e critérios de otimização. Apesar de não garantir soluções ótimas, este tipo de algoritmo mostrou-se bastante atrativo na prática, devido a sua potencialidade de produzir respostas de boa qualidade, e num tempo de computação tolerável, quando aplicado a problemas de escalonamento [28, 7, 8, 11, 20, 23, 15].

Algoritmos de busca tabu [14] também oferecem uma boa alternativa para resolução de problemas de otimização combinatória e, em particular, para problemas de escalomento de pessoal. Vários artigos da literatura descrevem aplicações bem-sucedidas deste método na resolução de problemas complexos variados (cf., [24]), mostrando que ele também é capaz de produzir soluções de boa qualidade em um tempo computacional aceitável.

Em geral, em ambas as heurísticas, cada restrição do problema que não é contemplada na solução dá origem a uma penalização ponderada na função objetivo. As heurísticas, então, procuram minimizar o valor da função objetivo e, desta forma, trabalham para evitar falhas nas escalas construídas. Por outro lado, como técnicas de otimização ambas se mostraram bastante eficientes, produzindo, em poucos minutos de execução em um PC típico, escalas de serviço bastante adequadas para a operação das enfermarias do HC. Por outro lado, cabe reforçar que as heurísticas não oferecem garantias de que chegarão a soluções ótimas, embora, em geral, as soluções finais encontradas são bastante satisfatórias.

As duas subseções seguintes oferecem uma rápida introdução às técnicas de algoritmos genéticos e busca tabu.

3.1 Algoritmos Genéticos

Esta subseção apresenta um breve resumo dos tópicos mais relevantes sobre algoritmos genéticos. Esse resumo foi baseado principalmente nos artigos [12, 18, 5, 21, 3, 25] e nos textos [22, 17, 10].

No início da década de 70, John Holland publicou um trabalho [19] que marcou o princípio de estudos mais sistemáticos sobre algoritmos genéticos (AGs). Uma das virtudes dos AGs é obter, de uma maneira uniforme, soluções adequadas para problemas complexos. Desde então, muitos pesquisadores desenvolveram vários trabalhos que exploram as potencialidades de AGs em muitos cenários.

3.1.1 A idéia central

AGs são heurísticas que imitam o funcionamento da seleção natural sobre uma população de indivíduos. A seleção natural privilegia indivíduos que apresentem características vantajosas e que estejam melhor adaptados ao meio no qual vivem. A evolução de uma população ocorre, essencialmente, por diferenciação dos genes dos indivíduos que a compõem. Essa diferenciação se dá através de dois mecanismos principais. Em primeiro lugar, o processo de reprodução combina características (possivelmente favoráveis) de indivíduos em um processo chamado recombinação dos genótipos. Em segundo lugar, processos de mutação introduzem transformações aleatórias nos genes.

A execução dos AGs imita o modo como uma população de indivíduos evolui na natureza. Há quatro conceitos importantes a considerar:

1. a evolução ocorre nos cromossomos, e não no ser vivo que os mesmos codificam;
2. a seleção natural privilegia a existência de seres melhor adaptados ao meio no qual vivem;
3. a reprodução combina características (possivelmente favoráveis) dos indivíduos para criar novos seres, num processo denominado de *crossover*; e
4. processos de mutação randômicos alteram as características de um indivíduo.

Esse modelo privilegia o aparecimento de indivíduos cada vez melhor adaptados ao ambiente e que vão substituir outros indivíduos na população.

De maneira análoga, um AG codifica um candidato à solução na forma de um cromossomo, representado por uma estrutura de dados. Cada instância de um cromossomo pode ser interpretada como um indivíduo. A idéia básica embutida em todo AG consiste em repetir os seguintes passos, dada uma população inicial de cromossomos:

1. Certos indivíduos da população corrente são selecionados para cruzamento. Em geral, procura-se privilegiar os indivíduos que detêm um alto valor de qualidade, medido através de uma função que, dado um cromossomo, retorna um número representativo de sua qualidade como solução de acordo com as necessidades do problema (função de *fitness*). Na seleção para cruzamento é, via de regra, injetado também um certo grau de aleatoriedade.
2. Os cromossomos selecionados passam pela operação de *crossover*, quando a informação de seus genes é recombinada, dando origem a novos indivíduos. Há vários operadores de *crossover* que podem ser usados, alguns envolvendo também certos parâmetros de aleatoriedade.

3. Os novos indivíduos gerados pela operação de *crossover* sofrem mutações, quando alguns dos genes presentes no cromossomo podem ser modificados de forma aleatória.

4. Esses novos indivíduos, finalmente, substituem, em parte, os indivíduos geradores originais, dando origem a uma nova população de cromossomos. E o processo se repete.

O ciclo é executado por um certo número fixo de vezes, ou até que o valor da função de *fitness* dos indivíduos na populaçao convirja para um valor limite. A solução final é, então, extraída da população final. Não está excluída a hipótese de que alguns cromossomos na população possam representar soluções que violem certas restrições, ditas restrições não impositivas. Este mecanismo garante uma maior dispersão dos indivíduos e pode contribuir para que o processo não se prenda a mínimos ou máximos locais.

De uma maneira mais precisa, o algoritmo pode ser descrito como segue. Seja S o conjunto de todas as soluções admissíveis, das quais algumas podem violar restrições não impositivas. Seja n o tamanho das populações. Este valor é um parâmetro do sistema e, via de regra, é obtido por experimentação. Seja $f : S \mapsto R$ uma função real capaz de avaliar a qualidade do indivíduo repreentado por um cromossomo. No instante $t = 0$, o AG cria uma população inicial $G_0 = \{g_1^0,\ g_2^0,\ \ldots,\ g_n^0\} \subseteq S$, de tamanho n. Usualmente, cada indivíduo g_i^0 é gerado de forma aleatória. Dada a população corrente $G_t = \{g_1^t,\ g_2^t,\ \ldots,\ g_n^t\} \subseteq S$, num instante $t \geq 0$, o AG associa a cada $g_i^t \in G_t$ o valor $f(g_i^t)$. Para obter os indivíduos da nova população, G_{t+1}, o AG escolhe um grupo de indivíduos de G_t, privilegiando, usualmente, aqueles indivíduos g_i^t onde $f(g_i^t)$ seja maior. Seja $H = \{g_{i_1}^t,\ g_{i_2}^t,\ \ldots,\ g_{i_k}^t\}$, o conjunto dos elementos selecionados. Em seguida, o AG aplica, em seqüência, os operadores de recombinação e mutação (*crossover*) aos elementos de H, gerando um conjunto $H' = \{g'_{i_1},\ g'_{i_2},\ \ldots,\ g'_{i_p}\}$ de novos indivíduos. Os parâmetros k e p são escolhidos por experimentação e, usualmente, tem-se $k = p$. A nova população intermediária G' é, então, formada por $G' = (G \setminus H) \cup H' \subseteq S$. Aos elementos da população G' é aplicado o operador de mutação, obtendo-se uma nova coleção de indivíduos, G''. Finalmente, a população G_{t+1} é gerada selecionando-se os r elementos de G_t que serão excluídos, sejam eles $E = \{g_{j_1}^t,\ g_{j_2}^t,\ \ldots,\ g_{j_r}^t\} \subseteq G_t$, e fazendo-se $G_{t+1} = (G_t \setminus E) \cup G''$. O parâmetro r também pode ser escolhido experimentalmente mas, via de regra, usa-se $r = k$.

3.1.2 Codificação e tamanho da população

Em um algoritmo genético tradicional, a codificação de um candidato à solução ocorre através um uma seqüência ordenada de símbolos extraídos de um alfabeto. Existem duas principais vertentes quanto à questão da melhor codificação. Parte dos livros e artigos defende o uso de um alfabeto binário (composto de 0's e 1's), mas há muitos exemplos de opiniões contrárias. Ainda não há, portanto, um consenso quanto a melhor estratégia para a codificação. Preferencialmente, uma solução não deve permitir mais de uma representação nos cromossomos, ou seja, a função para construir a solução a partir do cromossomo é, se possível, bijetora.

Semelhante à questão da melhor codificação, também não existe um tamanho ótimo para a população. Ao trabalhar com poucos indivíduos, arrisca-se não cobrir adequadamente todo os espaço de busca. Por outro lado, um número excessivo de indivíduos exige expressivos recursos computacionais adicionais, sem necessariamente levar a soluções significativamente melhores.

3.1.3 Reprodução e mutação

Ao realizar a reprodução, o algoritmo genético substitui uma fração G da população de tamanho n. Caso $G = 1$, então cada nova geração é formada de indivíduos novos. No entanto, corre-se o risco de um cromossomo muito bom perder-se logo em seguida na próxima geração. De Jong [17] sugere manter os melhores m indivíduos, num método conhecido como *elitismo*. Garante-se, assim, que o algoritmo genético nunca regredirá durante as iterações. No entanto, esta estratégia tende a limitar o alcance da busca no espaço de soluções. Pode-se também repor somente um pequeno número de indivíduos por geração ($G << 1$), método denominado *steady-state*.

O método da escolha de indivíduos para reprodução é um assunto bastante abordado na literatura. O mais simples, denominado *stochastic sampling with replacement* ou *seleção por roleta*, realiza escolhas de cromossomos baseadas em uma distribuição de probabilidades calculada a partir da qualidade dos indivíduos. A probabilidade p_i do indivíduo i, numa população de n indivíduos, é dada por $p_i = q_i / \sum_{j=1}^{n} q_j$, onde q_i é a qualidade do mesmo, medida através da função de *fitness*. Nesse método, um indivíduo poderia ser sempre o escolhido, viciando a amostra.

Para contornar esta desvantagem, De Jong propôs um método com o nome de *stochastic sampling without replacement*, que consiste em atribuir a cada cromossomo um contador e_i indicando quantas vezes deve ser selecionado. No início, e_i é dado por $e_i = n \times q_i/q$, onde n é o tamanho da população e q a média das qualidades. A seleção continua pelo método da roleta, usando os e_i como indicadores de probabilidades. Quando ocorrer uma seleção de um cromossomo, seu contador é decrementado em uma unidade. Somente cromossomos com contador positivo podem ser escolhidos. Isto garante que um cromossomo seja escolhido no máximo $n \times q_i/q$ vezes.

Uma terceira alternativa, conhecida como *deterministic samplig*, calcula o número de seleções de um cromossomo pela parte inteira de $e_i = n \times q_i/q$. Para completar a população, é realizada uma seleção por roleta usando a parte fracionária de e_i.

Outra alternativa é o método chamado *stochastic tournament*, que seleciona um número t de cromossomos através do método da roleta. O melhor dentre esses t cromossomos será o escolhido.

Um ponto merecedor de atenção é qual função utilizar para ajustar o valor numérico relativo a qualidade do indivíduo. Por exemplo, o resultado de seleções em uma população cuja qualidade varia de 30 a 100 será muito diferente de uma com qualidades entre 1030 e 1100. No primeiro caso, qualquer um dos métodos mencionados anteriormente exerce uma forte pressão sobre os melhores indivíduos, podendo levar a uma convergência prematura (perda de diversidade). No segundo caso, os valores estarão muito próximos entre si, e praticamente não contribuirão para a evolução dos indivíduos.

Uma alterntiva consiste em calcular $q_i = c_{max} - f_i(c_{max} - c_{min})/n$, onde c_{max} e c_{min} são os valores máximo e mínimo desejados para a qualidade do melhor e do pior indivíduo, respectivamente, e f_i é o valor gerado pela função de *fitness* o cromossomo i. Para evitar a perda da capacidade de evolução no decorrer das iterações, quando todos os cromossomos tendem a receber valores de q_i semelhantes e próximos da probabilidade de seleção, Davis [10] sugere calcular q_i da seguinte maneira: $q_i = f_i - f_m + c$, onde f_m é o valor resultante da função avaliadora para o pior cromossomo i. Por fim, pode-se projetar os resultados de forma linear no intervalo entre c_{min} e c_{max}, na forma

$$q_i = c_{min} + \frac{f_i - f_m}{f_M - f_m} \cdot \frac{c_{max}}{c_{min}} \tag{1}$$

A ampla variedade de métodos de recombinação e mutação impede com que todos sejam descritos em detalhe. Muitos ainda dependem da codificação usada nos cromossomos e de possíveis restrições quanto a ocorrência dos símbolos no cromossomo.

Além da seleção dos cromossomos para recombinação, há ainda a questão de como definir os pontos de corte para troca de segmentos entre dois cromossomos que são recombinados. A definição desses pontos deve procurar preservar as vizinhanças onde se encontrem indivíduos favoráveis. Esse aspecto pode se guiado pela estrutura interna da representação dos cromossomos e, em geral, é uma tarefa não trivial. Via de regra, explora-se algumas alternativas para determinar que tipo de corte se mostre mais favorável, na média das execuções.

Como vemos, pelo menos três pontos diferenciam um algoritmo genético das demais heurísticas:

1. manipula uma codificação da solução, ao invés da própria solução;
2. analisa simultaneamente um grande conjunto de pontos do espaço de busca, e não um único ponto;
3. emprega de regras com aleatoridades para efetuar decisões.

3.1.4 Vantagens relativas

AGs apresentam características vantajosas que estimulam seu uso em problemas de otimização combinatória. Entre elas estão:

1. a possibilidade de percorrer todo o espaço de soluções, ao contrário de outras buscas heurísticas que podem ser capturadas em mínimos ou máximos locais;
2. a inexistência de restrições quanto à natureza da função para avaliar a aptidão de candidatos;
3. a possibilidade de se especificar pesos diferentes para objetivos distintos, conforme a hierarquia de restrições do problema;
4. o fato de que problemas classificados como NP-difícieis não comprometem o funcionamento do algoritmo;

5. o algoritmo processa esquemas (padrões de ocorrência dos símbolos nos cromossomos) em paralelo;

6. em dado momento, esquemas cujas qualidades estejam acima da média da população, tendem a crescer em número, enquanto ou outros, menos vantajosos, tendem a tornar-se uma minoria;

7. um novo esquema, composto por pequenos segmentos de outros esquemas anteriores, poderá apresentar qualidade superior aos demais.

Por outro lado, AGs não garantem soluções ótimas, mas, à semelhança de outras heurísticas, certamente são uma ferramenta importante na busca de soluções de qualidade. Com certeza, no caso da enfermarias do HC, as soluções encontradas se mostraram de qualidade superior às soluções encontradas sem auxílio computacional. Além disso, o processo de busca com auxílio computacional é muito mais rápido.

3.1.5 Textos em destaque

Devido a sua relevância para o problema do HC em particular, merecem destaque os textos [12, 10, 26, 4].

O primeiro texto [12] trata do emprego de algoritmos genéticos em problemas de escalonamento. A primeira parte discute em detalhes alguns problemas semelhantes, tal como agendamento de exames na Universidade de Edinburgh, bem como a criação de tabelas de horários para disciplinas na mesma universidade. O texto discute como uma série de restrições foram codificadas em um algoritmo genético. Apesar de abordar problemas semelhantes ao problema das enfermarias do HC, este artigo não trata de escalonamento de funcionários. No entanto, a descrição de diferentes restrições e detalhes de suas representações em cromossomos sugeriram várias idéias que foram considerados na implementação da ferramenta computacional criada para solucionar o problema do HC.

O segundo texto [10] apresenta uma extensa coleção de informações interessantes para se projetar um algoritmo genético mais eficiente, tanto em relação à qualidade da solução, como também quanto ao tempo de execução.

O terceiro texto citado acima [26] foi escrito na forma de um turorial e apresenta os conceitos básicos sobre algoritmos genéticos. Depois de uma introdução ao assunto, o tutorial discute as representações de soluções em cromossomos e a função objetivo. No entanto, o texto limita-se ao algoritmo genético canônico, que emprega uma seqüência de bits como cromossomo. Com o auxílio do Teorema dos Esquemas (*Schemata theorem*), base da justificativa do funcionamento dos algoritmos genéticos, o tutorial analisa a eficácia de diferentes operadores de recombinação e mutação para um processo de convergência adequado. Sendo um texto bastante objetivo e sucinto, o tutorial transmite com eficiência alguns conceitos importantes que são úteis quando se implementa algoritmos genéticos.

No último texto mencionado acima [4], o autor descreve de maneira bastante resumida como programou um algoritmo genético para solucionar com sucesso o problema de construção de horários em escolas secundárias italianas, utilizando um cromossomo de forma não canônica. O autor sugere utilizar um cromossomo com símbolos, cada qual codificando uma situação de um professor em um determinado horário. Um filtro e restrições nas operações de recombinação garantem soluções sempre viáveis. Diferente dos algoritmos genéticos tradicionais, o autor usa um operador de recombinação que, ao invés de realizar o intercâmbio aleatório de trechos do cromossomo, troca somente os símbolos dos funcionários selecionados segundo um critério relacionado com uma função objetivo local para cada professor. A ferramenta computacional descrita nesse relatório emprega alguns conceitos apresentados neste artigo como, por exemplo, a garantia de que a restrição no número de folgas é sempre respeitada através do uso de métodos de mutação e recombinação adequados.

3.2 Busca Tabu

De um modo geral, um problema de otimização é aquele em que se deseja minimizar ou maximizar uma função de uma ou mais variáveis as quais devem satisfazer a um conjunto de restrições que definem o domínio das soluções viáveis. Matematicamente, isto poderia ser descrito na forma:

$$\begin{array}{ll} \min / \max & f(x) \\ \text{Sujeito a} & x \in \mathcal{D}, \end{array}$$

onde $\mathcal{D}$ é o domínio das soluções viáveis e $f(x)$ é a função objetivo. Assim, por exemplo, se tivermos um problema de programação linear com n variáveis, $\mathcal{D}$ é descrito por um sistema linear da forma $Ax \leq b$, onde $x \in \mathbb{R}^n_+$, sendo $f(x)$ uma função linear em x. Já se o problema for de programação linear inteira, as variáveis só poderão assumir valores inteiros, i.e., teremos $x \in \mathbb{Z}^n_+$.

A exemplo do que ocorre com boa parte dos algoritmos de otimização, os algoritmos de busca local podem ser resumidos pelos seguintes passos.

Passo 1 Inicializar com uma solução viável qualquer.

Passo 2 Enquanto o *critério de parada* não for satisfeito faça

Passo 2.1 Escolher uma solução *vizinha* da solução corrente

Passo 2.2 Mover para a solução escolhida;

Passo 3 Retornar a melhor solução visitada no Passo 2.

Portanto, para caracterizar um algoritmo de busca local, é preciso definir qual o *critério de parada*, o que caracteriza a *vizinhança* de uma solução (Passo 2.1) e o método de escolha da solução para a qual será feito o movimento (Passo 2.2). Note-se, por exemplo, que o algoritimo simplex clássico para programação linear tem uma estrutura como aquela representada acima. A solução inicial é o vértice retornado pela fase 1 do algoritmo, as soluções vizinhas são dadas pelos vértices adjacentes ao vértice corrente, e o vértice escolhido para realizar o movimento é aquele que corresponde à melhor variação (unitária) da função objetivo.

Em uma heurística de busca local clássica, o critério de parada do Passo 2 é atingido quando a solução corrente é tal que nenhum movimento pode ser realizado sem que se chegue a uma solução vizinha de custo melhor que o da solução corrente. Neste caso, diz-se que a solução corrente é um mínimo local. O problema com esta abordagem pode ser entendido analisando o gráfico da Figura 1, que descreve o comportamento da função objetivo de um problema de otimização definido para uma única variável. Considerando que a vizinhança está restrita a um intervalo pequeno da reta real, nota-se que se o algoritmo de busca local chegar ao ponto de mínimo local indicado, o critério de parada será atingido e, portanto, não mais será possível atingir o ótimo global da função objetivo.

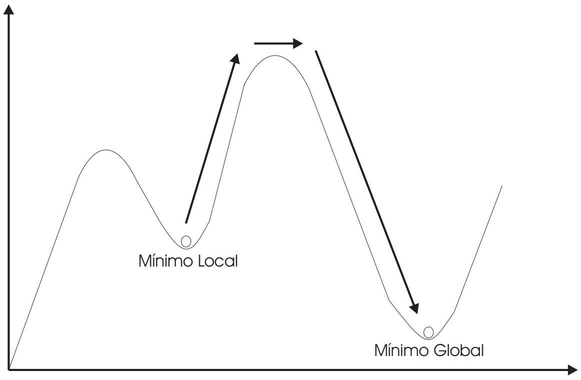


Figura 1: Gráfico de uma função objetivo com uma variável.

A busca tabu aparece nesse contexto como uma técnica para guiar a busca local de modo a permitir que ela escape de ótimos locais. Por isso, a busca tabu, juntamente com outros métodos que aplicam esta mesma técnica, é conhecida como uma *metaheurística*.

A idéia é permitir que haja movimento para soluções vizinhas mesmo que essas eventualmente deteriorem a função objetivo. Tipicamente, o movimento é feito de modo que se chegue à solução vizinha de melhor custo mesmo que este seja pior do que o da solução corrente. Contudo, no caso de piora no custo, haveria um risco de que o movimento inverso levasse o algoritmo de volta à solução corrente da iteração anterior. A maneira encontrada para impedir este comportamento cíclico na busca tabu foi criar uma lista de movimentos que permanecem proibidos, ou *tabu*, durante um certo número de iterações. Este número de iterações que um movimento permanece na *lista tabu* é denominado de *prazo tabu*. O critério de parada usualmente adotado restringe-se a um simples limite sobre o número total de iterações ou então sobre o número total de iterações sem que o custo da melhor solução visitada tenha sido alterado. Portanto, os passos básicos de um algoritmo de busca tabu [27] são:

1. Iniciar uma lista tabu vazia.
2. Obter um solução inicial S.
3. Inicializar S^* fazendo $S^* = S$.
4. Enquanto o critério de parada não for satisfeito faça:
    (a) Encontre o conjunto $Q(S)$ das soluções vizinhas de S que podem ser atingidas por movimentos que não são tabu.

   (b) Escolha S' como sendo a solução de melhor custo em $Q(S)$.
   (c) Adicione o movimento $m(S, S')$, que leva de S a S', à lista tabu.
   (d) Atualize S com S', ou seja, substitua S por S'.
   (e) Se a solução S' é melhor que S^* então faça $S^* = S$.

5. Retornar a melhor solução encontrada.

Percebe-se que o algoritmo de busca tabu possui alguns parâmetros que precisam ser fixados. Dentre eles tem-se o número de iterações que definem o critério de parada, o tamanho do prazo tabu e o tamanho da lista tabu. O ajuste destes parâmetros é uma tarefa essencial para o bom desempenho do algoritmo e, usualmente, esse ajuste é feito de modo experimental.

Em alguns casos combinam-se outras técnicas com a busca tabu, visando torná-la ainda mais eficiente. Uma delas é a inclusão no algoritmo de um *critério de aspiração* segundo o qual os movimentos da lista tabu tornam-se aceitáveis desde que levem a soluções de melhor custo.

Finalmente, cabe observar que a busca tabu, apesar da tentativa de escapar dos mínimos locais, não oferece qualquer garantia de que chagará a uma solução ótima, embora, em geral, as soluções finais encontradas são superiores àquelas retornadas pelas heurísticas clássicas de busca local. Como descrito no início dessa seção, a busca tabu é uma heurística de busca local.

4 A Implementação dos Algoritmos

Para cada uma das heurísticas consideradas, algoritmos genéticos e busca tabu, foram desenvolvidas implementações. Nessa seção, são descritas em detalhes as implementações criadas para cada uma das duas heurísticas.

4.1 A Implementação dos Algoritmos Genéticos

Conforme descrito na seção de fundamentos teóricos, existe uma ampla gama de variações de implementações para algoritmos genéticos, sendo raro encontrar-se duas implementações semelhantes. Existem divergências na questão de: codificação dos cromossomos, como inicializá-los, como e quando realizar a mutação e a recombinação, qual a melhor estratégia de seleção de cromossomos, como substitur os cromosomos na população atual, e como promover ajustes nos resultados da avaliação da qualidade da solução. Considerando que cada um desses itens possui parâmetros próprios, existe uma infinidade de algoritmos genéticos candidatos para resolver um dado problema. No caso aqui descrito, foram implementadas duas variantes que diferem entre si, basicamente, na representação interna dos cromossomos e, portanto, também nos métodos de recombinação e mutação.

4.1.1 Descrição geral

O primeiro passo foi desenvolver uma algoritmo genético na linguagem C++ e com características de ser genérico, modular, parametrizado e flexível:

1. *Genérico*: O algoritmo não resolve apenas um problema em particular. Em princípio, pode ser utilizado para outros problemas de otimização totalmente distintos do problema que tratamos aqui. Tampouco define qual codificação utilizar, nem quais os métodos que devem operar sobre os cromossomos.
2. *Modular*: O código é composto por várias funcionalidades, as quais podem ser substituídas por outras sem intervenções no código fonte do algoritmo genético.
3. *Parametrizado*: Nenhuma grandeza é intrínseca ao código compilado. Qualquer ajuste de parâmetros pode ser incorporado alterando-se um arquivo texto externo.
4. *Flexível*: Não só os parâmetros são lidos do arquivo texto, como também a descrição dos módulos a serem utilizados. Isto significa que a modularidade é em tempo de execução, dispensando uma nova compilação sempre que se desejar alterar o comportamento do algoritmo.

Para se obter a modularidade e a flexibilidade desejadas, recorreu-se às facilidades da linguagem C++ para programação orientada a objetos. Segue uma descrição detalhada da implementação.

O algoritmo genético corresponde a uma classe, com campos como tamanho da população, número de gerações, tamanho da elite (número de melhores cromossomos copiados sem alteração) e número de cromossomos substituídos por geração. Antes de iniciar a execução, o algoritmo avalia estas informações e automaticamente modifica, se necessário, alguns destes parâmetros para adaptá-los à manipulação dos cromossomos.

Inicialmente, o algoritmo genético cria os cromossomos da primeira geração. Então, enquanto não convergir e enquanto não ultrapassar o limite máximo de iterações, o algoritmo continua processando os cromossomos em iterações sucessivas.

No incício de cada iteração, e para cada cromossomo, o algoritmo calcula sua qualidade, mapeia-a para um número representativo e ajusta esse valor para um intervalo conveniente.

Em seguida, supondo uma população de tamanho estável p, o algoritmo copia os m melhores cromossomos para a próxima geração, seleciona um certo número n de cromossomos para recombinação e mutação e, por fim, copia mais f cromossomos até completar a nova população de tamanho p, onde tal que $m+n+f = p$. Escolhendo os valores dos parâmetros m (tamanho da elite) e n (número de cromossomos substituídos por geração), pode-se simular as principais estratégias de substituição de população encontradas na literatura: troca de toda a população ($n = p, m = f = 0$), populações com interseção ($m = 0, n = p - i$, onde i é o tamanho da interseção) e estratégia elitista ($m > 0$).

Os métodos do algoritmo genético não manipulam diretamente as estruturas de dados que representam os cromossomos. Ao invés, e para assegurar generalidade, os métodos do algoritmo comunicam-se com objetos de classes as quais de fato conhecem e operam sobre as estruturas de dados dos cromossomos. Estes objetos, cujas classes são derivadas das classes abstratas descritas a seguir, também são passados como parâmetro para o algoritmo genético. É essa a característica que oferece a flexibilidade e a modularidade em tempo de execução. Para cada algoritmo genético específico, pode-se criar instâncias de uma série de classes particulares para o problema as quais, desde coerentes entre si, determinam o comportamento do algoritmo em questão. As principais classes estão descritas a seguir.

Conjunto de cromossomos : O algoritmo genético aceita qualquer tipo de representação de cromossomos, sejam eles uma cadeia de bits, um vetor de números inteiros ou até representações complexas como árvores ou grafos. Uma classe de objetos encarrega-se de guardar duas gerações com certo número constante de cromossomos, copiá-los de uma geração para a outra e fornecer acesso para a representação de cada cromossomo através de um índice. Não existe um objeto para cada cromossomo, pois isso resultaria em uma sobrecarga de processamento. O algoritmo genético tem a capacidade de manipular somente uma única classe de cromossomos por vez. Escolhida a estrutura de dados para representar os cromossomos, isso implica que o restante dos objetos devem ser compatíveis com a representação escolhida.

Inicializador de cromossomos : A forma de gerar os cromossomos da população inicial afeta a eficiência do algoritmo, embora de maneira menos significativa. Esta operação também depende da representação utilizada para os cromossomos. Assim, o programador deve implementar um método de inicialização compatível com a estrutura de dados escolhida para representar os cromossomos.

Método de recombinação : Dados o índice de dois cromossomos, esta classe calcula dois novos cromossomos a partir de uma estratégia de recombinação coerente com a representação dos cromossomos e com a natureza do problema. Caso exista somente um método de recombinação passado como parâmetro, o algoritmo genétco dispõe de três modos para aplicar esse método:

1. Em todos os cromossomos.
2. Em alguns os cromossomos, conforme uma certa probabilidade.
3. Em todos os cromossomos, porém somente em gerações que sejam múltiplas de um certo número.

Quando o algoritmo não realiza uma recombinação, ele simplesmente copia os dois cromossomos indicados. Havendo mais de um método de recombinação passado como parâmetro, o algoritmo pode:

1. Usar somente o primeiro.
2. Escolher um deles segundo certo um peso.

A ação a tomar nessas situações é definida no código fonte, mas uma extensão interessante seria torná-la um parâmetro do algoritmo.

Método de mutação : Esta classe encarrega-se de alterar localmente um cromossomo. De maneira análoga à recombinação, havendo somente um único método passado como parâmetro, o algoritmo pode aplicá-lo:

1. Em todos os cromossomos.
2. Em alguns os cromossomos, conforme uma certa probabilidade.
3. Em todos os cromossomos, porém somente em gerações que sejam múltiplas de um certo número.

Havendo mais de um método de mutação, o algoritmo pode:

1. Usar somente o primeiro.
2. Usar todos sucessivamente sobre cada cromossomo.
3. Escolher um deles segundo seu peso.
4. Usar cada método em todos os cromossomos, mas somente em gerações múltiplas de um certo número.

Em todos os casos que envolvem uma decisão baseada em uma probabilidade ou peso de um método, este valor é dado por uma função que também é um parâmetro. Com isto, os pesos ou probabilidades dos métodos podem variar com o decorrer das iterações. Esta é uma característica bastante interessante. Por exemplo, pode-se definir uma probabilidade de mutação que aumente a medida com que as soluções convergem.

Em geral, a representação de um cromossomo e da solução por ele codificada são objetos distintos. Raramente o problema apresenta uma simplicidade tal que uma avaliação diretamente sobre o cromossomo seja interessante. O algoritmo genético implementado constrói uma solução a partir do cromossomo para só então avaliá-la. Para isto, há três classes:

Solução : Tal como o conjunto de cromossomos, esta classe implementa uma respresentação da solução. Para cada algoritmo implementado, somente uma representação é possível.

Gerador de solução : Esta classe, a partir de um cromossomo, decodifica-o para a solução que representa.

Avaliador : Dada uma solução, esta classe avalia-a e retorna um número que de alguma maneira traduz a qualidade da solução. Quanto maior o número, melhor é a solução.

Foram implementadas ainda outras classes, mais genéricas e independentes do problema, e que também podem ser substituídas para alterar o comportamento do algoritmo. Entre elas:

Ajuste de qualidades : A distribuição dos valores numéricos correspondentes à qualidade das soluções influencia diretamente na velocidade de convergência do algoritmo. Situações infelizes podem levar a uma convergência prematura, gerando soluções insatisfatórias. Para evitar esse problema, o algoritmo genético aceita como parâmetro uma classe que pode ser usada para aplicar uma fórmula sobre a qualidade das soluções, de forma a ajustá-las segundo uma estratégia favorável.

Seleção de cromossomos : Não existe consenso na literatura quanto a melhor forma de escolher os cromossomos para construir a nova geração. Por este motivo, esta operação também é modularizada.

Com todo o aparato descrito anteriormente pronto e funcional, introduzir uma nova codificação ou mudar os métodos de operação do algoritmo é uma tarefa relativamente simples. Exigiria apenas a implentação de novas classes derivadas das classes abstratas.

4.1.2 A função objetivo

Um algoritmo genético trata usualmente de problemas de maximização. Porém, o problema das escalas visa minimizar o custo representado pela função objetivo, que deve cobrir todas as restrições presentes no problema. Assim, o método transforma o custo de uma escala e trata um problema de maximização equivalente. Para tal, calcula-se o inverso do custo da escala como o valor de *fitness* de um cromossomo. Esse procedimento, entretanto, dificulta a distinção de escalas com custos elevados, pois o inverso desses custos difereria muito pouco entre si. Para evitar essa anomalia, o valor da função objetivo é mapeado linearmente sobre um intervalo de números.

Na função objetivo, cada violação é formada por um certo número de ocorrências de alguma irregularidade em alguma parte da escala de um funcionário. Seja $o > 0$ o número de ocorrências da irregularidade e p o custo da penalização. Então calcula-se $p = v_1 + (o-1)v_2$. As constantes v_1 e v_2 são valores inteiros que correspondem ao valor da penalização para cada restrição. Como são passadas como parâmetro para o algoritmo, pode-se penalizar a primeira ocorrência de uma irregularidade com um custo diferente das demais ocorrências dessa mesma violação. Dependendo da situação, esta prática mostrou-se útil. O custo final da escala é dado pela soma dos custos das penalizações.

O procedimento utilizado pela função objetivo para contabilizar e penalizar as violações está descrito a seguir.

Impedimentos : Como os cromossomos representam as escalas de maneira a sempre respeitar os impedimentos nas duas versões do algoritmo genético, a função objetivo não verifica esta restrição.

Máximo e mínimo de dias consecutivos de trabalho : É preciso verificar se nenhum dos funcionários possui uma seqüência de dias de trabalho consecutivos e cujo comprimento seja menor que $MinDiasConsecutivos$ ou maior que $MaxDiasConsecutivos$. Com esta restrição podemos eliminar dias de folga e de trabalho intercalados escolhendo $MinDiasConsecutivos \geq 2$. Períodos de trabalho muito prolongados são evitados impondo $MaxDiasConsecutivos \leq 6$.

A verificação trabalha com dois contadores: i, que indica o número do dia que inicia uma seqüência de dias consecutivos de trabalho e f, que corresponde ao número do dia da folga que finaliza esta seqüência. Em primeiro lugar, localiza-se a última folga no histórico no mês anterior para esse funcionário. Como o número de dias no histórico possui pelo menos tantos dias quanto o número máximo de dias consecutivo de trabalho, temos certeza de que o histórico possuirá no mínimo uma folga, senão a escala do mês anterior estaria inválida. A posição dessa folga é representada por um número negativo u, que indica quantos dias essa folga antecede o primeiro dia do mês em questão. O primeiro valor de i será $i = u + 1$. O primeiro valor de f será o dia da primeira folga do funcionário na escala.

Em seguida, percorre-se os dias de trabalho consecutivos da escala e atualiza-se os contadores i e f. Se em algum momento, $f - i > MaxDiasConsecutivos$, então aplicamos a penalização quanto ao *excesso de dias de trabalho consecutivos*, usando

$f - i - MaxDiasConsecutivos$ como número de irregularidades. Por outro lado, se $f - i < MinDiasConsecutivos$, penalizamos em relação aos valores definidos para o *mínimo de dias de trabalho consecutivos*, utilizando $MinDiasConsecutivos - (f - i)$ como número de ocorrências da violação.

Mínimo e máximo de folgas : Este teste pode ser realizado de maneira mais simples. Os números de folgas desejados para cada funcionário j são dados pelo parâmetro $Folgas[j]$. Para cada funcionário, conta-se o número $f[j]$ de folgas que o mesmo recebeu durante o mês. Se para um funcionário j, $f[j] > Folgas[j]$, então aplica-se a penalização relacionada ao *excesso de folgas*. De maneira análoga, se obtivermos $f[j] < Folgas[j]$, penaliza-se com o custo de *falta de folgas*. Em ambos os casos, utilizamos $\|f[j] - Folgas[j]\|$ como número de irregularidades.

Mínimo e máximo de finais de semana : Cada funcionário j deve receber, no mínimo, $MinFimSemana[j]$ finais de semana como folga. Se possível, o número de folgas em finais de semana realmente concedidos deve ser semelhante para todos os funcionários. Ou seja, deseja-se equilibrar a distribuição de finais de semana em folga entre todos os funcionários. Em primeiro lugar, determina-se quais pares de dias constituem um final de semana, tomando o devido cuidado para se considerar também o primeiro dia do mês com o último dia do histórico do funcionário. Em seguida, para cada funcionário j, conta-se o número $d[j]$ de folgas que o mesmo recebeu nos domingos, o número $s[j]$ de folgas em sábados e o número $f[j]$ de finais de semana completo onde recebeu folgas. Calcula-se também o privilégio $p[j]$ do funcionário j quanto aos finais de semana, definido como $p[j] = c_d d[j] + c_s s[j] + c_f f[j]$. As constantes c_d, c_s e c_f permitem definir a importância de cada tipo de folga.

Se para um funcionário j encontrarmos $f[j] < MinFimSemana[j]$, então aplicamos a penalização relacionada a *falta de finais de semana*, usando o valor calculado por $MinFimSemana[j] - f[j]$ como o número de irregularidades. Para o mesmo funcionário, também penalizamos a função objetivo com um valor proporcional a $p_j - min(p[j])$. Ou seja, quanto mais privilegiado for o funcionário em relação ao menos privilegiado, maior a penalização.

Número de preferências atendidas : A última restrição está relacionada ao número correto de dias preferenciais de folga que são atendidos. Cada funcionário j deve escolher um certo número de dias nos quais deseja folgar. Contudo, somente um certo número mínimo, $MinPreferencias[j]$, destas preferências devem ser atendidas. No entanto, se for possível, o algoritmo concederá mais folgas em dias de preferências.

Esse cálculo também pode ser realizado de maneira simples. Para cada funcionário j, contabiliza-se o número $f[j]$ de folgas que coincidem com dias de preferência do funcionário e o número $p[j]$ de dias indicados como preferenciais. A comparação do número de preferências atendidas e o número mínimo desejado, entretanto, solicita cuidados extra. No caso do funcionário indicar um número menor de preferências ao qual tinha direito, o algoritmo inclui uma penalização somente se o número de preferências atendidas for menor que o número de preferências informadas. Matematicamente, se para um funcionário j ocorrer $f[j] < min(MinPreferencias[j], p[j])$, então aplica-se a penalização relacionada a *falta de preferências*, utilizando-se o valor de $min(MinPreferencias[j], p[j]) - f[j]$ como o número de ocorrências no cálculo da penalização. No entanto, se $p[j] > MinPreferencias[j]$, $f[j] > MinPreferencias[j]$ e $f[j] < p[j]$, então o funcionário indicou mais dias aos quais tinha direito e nem todas essas preferências foram atendidas. Neste caso, aplica-se uma penalização relacionada a *falta de preferências adicionais*, porém com custo significativamente menor, pois não se trata de uma violação de restrição, mas somente da não observação de uma propriedade desejada para a escala.

Horas extras : A função objetivo determina para cada dia i o número $t[i]$ de funcionários trabalhando. Existe um limitante superior para o número máximo de funcionários em regime de horas-extra por dia e por mês, ligados à disponibilidade de profissionais no hospital. Esses valores são descritos pelos parâmetros $MaxFuncionariosDia$ e $MaxFuncionariosMes$. Seja ainda $MinFuncionarios[i]$ o número de funcionários necessários num dia i.

Há quatro considerações sobre esta questão. Em primeiro lugar, se verificarmos que $t[i] > MinFuncionarios[i]$, então aplica-se uma leve penalização sobre o *excesso de funcionários*, utilizando o próprio excesso como número de ocorrências $(t[i] - MinFuncionarios[i])$. Normalmente, esta verificação tem um efeito brando, visto que o quadro do HC dificilmente apresenta funcionários em quantidade suficiente para cobrir o mínimo de funcionários necessários, todos os dias. No entanto, decidiu-se manter esta penalização para deixar o algoritmo mais robusto.

Em segundo lugar, se $e[i] < MinFuncionarios[i]$, então penaliza-se o déficit como *falta de funcionários*. Considerou-se a possibilidade de não penalizar as horas-extra que fossem inevitáveis, mas esta funcionalidade não apresentou resultados satisfatórios, pois as outras restrições passaram a ganhar muito mais importância na função objetivo. Nesse caso, o algoritmo tendia a agradar a todos os funcionários, mesmo que gerasse sérias violações em relação ao número de horas-extra.

Em terceiro lugar, se a condição $e[i] < MinFuncionarios[i] + MaxFuncionariosDia$ for verificada, então aplica-se uma penalização mais pesada para cada funcionário extra que extrapola o limite da inequação. Esta penalização merece alguns comentários. Inicialmente, penalizava-se somente uma única vez cada dia com mais de $(e[i] - MinFuncionarios[i] - MaxFuncionariosDia)$ funcionários, mas, em escalas com recursos humanos muito restritos, isto levava a uma tendência a atribuir folgas a todos os funcionários em alguns dias. Assim, optou-se também pela penalização progressiva para este item.

Em seguida, são apresentadas as duas versões das implementações que foram codificadas e testadas.

4.1.3 A primeira versão da implementação

A primeira vesão implementada foi baseada nas seguintes características:

1. Representação os cromossomos como seqüências de bits;
2. uso de um único método de recombinação simples, i.e., trocas podem ocorrer em um único ponto, aplicado em todas as gerações e em todos os cromossomos;
3. mutação de um único bit, aplicada em todas as gerações e em todos os cromossomos;
4. decodificação de uma cadeia de bits para uma escala de trabalho;
5. avaliaçao da escala segundo a função objetivo descrita acima;
6. linearização dos resultados para uma faixa de valores;
7. seleção dos cromossomos com probabilidade proporcional à sua qualidade (método da roleta).

Em seguida, discutimos alguns detalhes dessas características. Em primeiro lugar, vale ressaltar que as escalas são codificadas em cromossomos de *bits*, da seguinte maneira: para cada funcionário existe uma seqüência de bits, cada um correspondendo a um dia sem impedimento. O cromossomo constitui-se então da justaposição das seqüências de bits de todos os funcionários. Quando um bit está ativo, o algoritmo entende que o funcionário em questão estará trabalhando no dia associado àquele bit. Com esta codificação obtém-se um modelo simples, razoavelmente fácil de implementar, e com uma característica vantajosa: somente dias pertinentes ao problema são codificados.

Para criar os cromossomos iniciais, a cada bit é atribuído o valor um ou zero, segundo uma probabilidade. Esta probabilidade foi escolhida de maneira que a esperança para o número total de folgas da escala seja aproximadamente igual à soma do número de folgas que devem ser atribuídas a todos os funcionários. Notou-se que outras probabilidades também geravam soluções aceitáveis, mas exigiam um número consideravelmente maior de gerações, uma vez que o algoritmo precisaria esperar que a mutação corrigisse o número de folgas atribuídas aos funcionários.

O processo de recombinação ocorre de maneira canônica. Dados dois cromossomos, escolhe-se uma posição aleatória e troca-se os bits após esta posição de ambos os cromossomos. Este método não respeita as seqüências de bits de cada funcionário e, de fato, pode gerar novos cromossomos com número de folgas desequilibrado entre os funcionários. Por exemplo, em um novo cromossomo um funcionário com direito a 10 folgas, pode herdar 8 folgas do primeiro cromossomo e outras 8 do segundo, acumulando um excesso de folgas. O outro cromossomo novo, gerado no mesmo processo, herdaria somente 2 folgas de cada cromossomo pai, resultando num déficit de 6 folgas.

A mutação também segue o modelo tradicional. Dado um cromossomo, escolhe-se aleatoriamente um bit, e inverte-se seu valor. Tal como na recombinação, este método pode gerar funcionários com excesso ou déficit de folgas.

Fica sob responsabilidade da função objetivo penalizar as possíveis violações introduzidas pelos métodos descritos acima. Quanto às outras restrições, não existe nenhum mecanismo automático para tratá-las. No entanto, a função objetivo penaliza as escalas que apresentem alguma violação relacionada às outras restrições. Como discutido adiante, os ótimos resultados obtidos por esta versão, sinalizam que essas escolhas foram adequadas.

4.1.4 A segunda versão da implementação

Numa segunda versão, foi implementada uma estratégia diferente para codificação dos cromossomos, descrita a seguir. A mesma função objetivo ainda é utilizada.

Uma das características da função objetivo é possuir muitos parâmetros que devem ser otimizados em paralelo. Para gerar uma medida da qualidade da solução, a função objetivo atribui custos a cada tipo de restrição e o algoritmo tenta minimizar a soma do total dos custos da escala. O grande número de restrições e parâmetros envolvidos na função objetivo traz o complicador de se encontrar uma solução boa em um espaço de busca com várias dimensões. Sob estas condições, é plausível supor que seja comum o algoritmo melhorar a escala em relação a alguma restrição, enquanto essa mesma alteração afeta de maneira negativa uma outra restrição. Se ambas as restrições envolvidas apresentarem custos muito próximos, será difícil para o algoritmo decidir qual alternativa é realmente a melhor. Há ainda a questão da complexidade da função objetivo. Como o algoritmo realiza uma avaliação da solução representada por todos os cromossomos e em todas as gerações, o desempenho da função objetivo é estratégico. Aliada ao grande número de restrições a serem consideradas, todas essas avaliações podem provocar uma queda considerável no desempenho do algoritmo. Durante os estudos para a segunda versão buscou-se, portanto, encontrar uma alternativa com menos testes na avaliação da função objetivo, para justamente aliviar esse tipo de problema.

Tal como na versão anterior, continua-se a representar os cromossomos na forma de uma cadeia de bits. Para cada funcionário codifica-se somente seus dias não impedidos. Por outro lado, deseja-se garantir sempre que o número correto de folgas será atribuído a cada funcionário, evitando-se o problema no desbalanceamento do número de folgas verificado na versão anterior. Os cromossomos, portanto, seriam criados com o número correto de folgas e os métodos de mutação e recombinação manteriam esta propriedade ao longo das gerações. Isto significa que em cada seqüência de bits correspondente a um funcionário, o número de bits ativos deverá se manter constante. Com isso, elimina-se também os testes relativos à esta propriedade, quando da avaliação da função objetivo.

Para criar um cromossomo com essa propriedade, divide-se o mesmo nas seqüências de bits correspondentes a cada funcionário. Para cada uma dessas seqüências, ativamos aleatoriamente tantos bits quanto o número de folgas desejado para o respectivo funcionário.

Ao todo, três diferentes métodos de recombinação foram implementados. O primeiro escolhe aleatoriamente uma seqüência de funcionários, que é trocada entre os dois cromossomos. Assim, a propriedade do número de folgas por funcionário permanece válida, pois as trocas ocorrem somente nos limites das seqüências de bits de cada funcionário. O segundo método não se limita às fronteiras entre funcionários. Para garantir a propriedade do número de folgas, o método antes verifica quais posições constituem pontos viáveis de troca. Isto pode ser determinado percorrendo-se ambos os cromossomos do início para o fim, contanto para cada um deles quantos bits estão ativos do começo até a posição corrente. Se, em dada posição, o número de bits ativos for igual em cada um dos dois cromossomos, então essa posição forma um ponto de troca viável. O terceiro método funciona de maneira semelhante ao segundo. No entanto, nesse caso, escolhe-se um ponto de troca para cada funcionário. Ou seja, neste caso a recombinação troca partes das escalas de cada funcionário, mantendo a propriedade do número correto de folgas.

A mutação simplesmente escolhe aleatoriamente uma seqüência de bits de um funcionário. Dentro dessa seqüência, são sorteadas duas posições cujos bits tenham valores opostos e o conteúdo dessas posições é trocado. É fácil constatar que esse método mantém a propriedade do número correto de folgas, pois afeta somente um único funcionário e, para esse, não remove folgas, como também não adiciona folgas novas.

4.2 A Implementação da Busca Tabu

No caso da busca tabu também foram implementadas duas versões da heurística. As duas versões diferem, basicamente, na proposta de vizinhança que é percorrida a cada iteração do laço principal do algoritmo. A seguir, descrevem-se as partes comuns do algoritmo e, em seguida, discute-se as duas propostas de vizinhança.

4.2.1 A parte comum

O algoritmo de busca Tabu foi implementado na linguagem C++. Os objetos definidos foram modularizados de acordo com uma característica específica definindo um conjunto de funcionalidades. Cada item abaixo corresponde a um título dado para o objeto e o nome do objeto aparece entre parênteses:

Leitura de dados e escrita da solução (Interface): Esse objeto faz a leitura dos dados do arquivo padronizado, explicado na seção 8 do primeiro relatório. Os dados são extraídos e guardados em variáveis do objeto *Dados*, definido no item 5. O objeto de Interface também escreve a solução no mesmo arquivo padronizado, criando um novo grupo com o nome *PROPOSTA*, que será identificado como a solução proposta pelo algoritmo para o problema.

Leitura das Configurações (Configuracao): esse objeto é responsável por ler os parâmetros de configu- ração do algoritmo de busca Tabu (tamanho da lista Tabu, número de iterações a executar, e parâmetros utilizados na geração da solução inicial) e armazená-los. O arquivo de leitura é o mesmo utilizado pelo objeto *Penalidades.* Esse arquivo também é um arquivo padronizado que possui dois grupos: um grupo chamado *PENALIDADES* e outro chamado *CONFIGURACAO.* Dentro do grupo *CONFIGURACAO* estão os ítens que definem os parâmetros necessários para o algoritmo. Este grupo tem o seguinte formato:

```
[CONFIGURACAO]
TamanhodaListaTabu = 20
Iteracoes = 2000
Multiplicador = 2
Divisor = 3
```

Os nomes dos dois primeiros parâmetros indicam prontamente o seu significado. Os dois últimos parâmetros são utilizados na geração da solução inicial definida na seção 4.2.3. Como será visto nessa seção, o multiplicador e o divisor são utilizados para definir quantos dias de folga (F) e de trabalho (T) serão atribuídos à solução inicial.

Leitura das Penalidades (Penalidades): o algoritmo de busca Tabu é baseado na definição de penalidades para cada uma das restrições impostas ao problema. Essas penalidades são configuráveis em um arquivo texto padronizado. Esse arquivo é o mesmo utilizado pelo objeto *Configuracao*. O grupo definido nesse arquivo é representado abaixo:

```
[PENALIDADES]
FolgasMes = 500                 \\ Numero minimo de folgas que devem ser
                                \\ fornecidas para cada funcionario
FuncionariosDia = 60            \\ Necessidade de funcionarios em cada dia
MinPreferenciasAtendidas = 400  \\ Preferencias atendidas para cada funcionario
PreferenciasAtendidas = 50      \\ Penalizacao por cada preferencia nao atendida
FolgaFinaisSemana = 180         \\ Folga nos finais de semana
DiasConsecutivos = 500          \\ Maximo de dias consecutivos de trabalho
LimiteHorasExtraDia = 20        \\ Limite maximo de horas extras em cada dia
MinDiasIntersticio = 100        \\ Minimo de dias consecutivos de trabalho
PenalidadeSimples = 10          \\ Penalidades relacionadas ao balanceamento
PenalidadeDupla = 10            \\ de folgas simples e duplas em finais de semana
```

Este objeto lê os dados que informam as penalidades, armazenando-os para utilização pelo objeto que calcula o custo de cada solução. Nesse grupo, diferentes penalidades implicam em diferentes tipos de restrição. Ou seja, penalidades com valores menores representam restrições mais leves, que, em geral, não devem ser estritamente obedecidas para gerar soluções factíveis. Penalidades com valores mais altos são atribuídas a restrições que, quando não obedecidas, tornam a solução infactível.

Cálculo dos custos de cada solução (Custo): Cada uma das soluções gerada pelo algoritmo deve ser analisada com base nas restrições impostas pelo problema. Essa análise é feita pelo objeto *Custo* que calcula, através das penalidades definidas no objeto *Penalidades*, o custo da solução. A cada solução é atribuído um custo e o algoritmo evolui no sentido de encontrar a solução de menor custo.

Criação e manipulação de estruturas de dados (Matrizes): Na seção 4.2.2 são definidas as estruturas de dados utilizadas para manipular os dados e as soluções encontradas para o problema. Essas estruturas e sua manipulação são definidas neste objeto.

Aplicação do algoritmo (Tabu) **e vizinhanças** (Vizinhanca): O objeto responsável pela aplicação do algoritmo de busca Tabu, ou seja, aquele que manipula os dados, guardando a melhor solução obtida, está definido conjuntamente com o objeto responsável pela vizinhança. Esta última é essencial para a evolução da busca Tabu, definindo os movimentos que podem ser tomados a partir de uma determinada solução, produzindo novas soluções a serem analisadas pelo objeto *Custo*.

Outras funcionalidades (Outros): Outras funcionalidades, como, por exemplo, funções para cópia de dados, exibição, entre outros, são definidas neste objeto. Isso facilita a manipulação da estrutura de dados utilizada para lidar com a solução encontrada (matrizes).

Outras funcionalidades (Outros): Outras funcionalidades, tais como funções para cópia e exibição de dados, entre outras, são definidas neste objeto. Isso facilita a manipulação da estrutura de dados que lida com a solução encontrada (Matrizes).

4.2.2 A estrutura de dados

Além dos objetos exigidos pela implementação do algoritmo e seus dados, existe a necessidade de criação de uma estrutura principal para armazenar a solução encontrada para o problema e, a partir dela, calcular custos, mover para outras soluções utilizando-se do conceito de vizinhança, e enfim, gerar a solução final para o problema. Nessa implementação, optou-se pela utilização de matrizes para representar esses dados. O uso de matrizes é justificado pela própria composição dos dados de entrada e pela forma como devem ser apresentados os dados de saída. A saída deve ser uma planilha cujas linhas representam funcionários e as colunas os dias do mês. Assim, a estrutura de dados que representa a solução para o algoritmo é codificada na forma de uma matriz na qual as linhas e colunas têm a mesma representação da planilha de saída.

4.2.3 A obtenção da solução inicial

O algoritmo de busca Tabu necessita, em geral, de uma solução inicial para começar a sua execução. Neste projeto, a solução inicial é dada por uma matriz, nos padrões especificados na seção 4.2.2, gerada de forma pseudo-aleatória. Ou seja, a matriz é obtida atribuindo-se a cada posição um F, para folgas, um T para dias de trabalho, um I, para posições impedidas, ou um D, para folgas desejadas que devem ser atendidas. Como a exigência do hospital é de que no mínimo em aproximadamente 1/3 dos dias sejam atribuídas folgas para cada funcionário, a matriz é preenchida com 1/3 de Fs e 2/3 de Ts descontando-se os dias de impedimentos. Como visto no início da seção 4.2 o valor 2/3 pode ser parametrizado através da definição, no arquivo de parâmetros, de um *Multiplicador* e de um *Divisor*.

É importante observar que durante a execução do algoritmo, as posições marcadas com I (impedimentos) ou D (folgas desejadas) não podem ser alteradas, pois gerariam soluções infactíveis para o problema.

4.2.4 O cálculo dos custos de cada solução

Cada matriz de dados que representa uma solução deve ter o seu custo calculado para, a partir deste custo, manter-se a melhor solução pela qual o algoritmo passou. Na implementação da busca tabu para esse problema, o custo é calculado com base nas penalizações dadas pelo objeto *Penalidades*. Existem, basicamente, dois tipos de penalidades que podem ser atribuídas a uma matriz de dados, aumentando seu custo: as penalidades constantes e as penalidades cumulativas.

As penalidades constantes são somadas ao custo somente uma vez para cada restrição violada. Por exemplo, a restrição que indica o número mínimo de funcionários trabalhando em determinado dia. Se a quantidade de funcionários não é suficiente naquele dia, ocorre a penalização. Outro exemplo é a restrição sobre o número mínimo de folgas por funcionário. Nesse caso, a penalização ocorre somente uma vez por linha da matriz que apresenta uma violação.

Por outro lado, as penalidades ditas cumulativas aumentam o valor da penalização na proporção em que aumenta o desvio em relação ao limite estabelecido pela restrição. Por exemplo, a restrição de número máximo de dias consecutivos de trabalho induz uma penalização cumulativa. Supondo que o número máximo de dias de trabalho é 6, e que um funcionário está trabalhando 8 dias consecutivos e outro trabalha 7 dias consecutivos, a penalização imposta no primeiro caso será maior.

Após analisadas todas as restrições, e quando todas as penalizações forem somadas, o algoritmo responsável pela manipulação da solução é acionado. Esse algoritmo armazena a melhor solução e define o próximo movimento.

4.2.5 A obtenção da solução de melhor valor

A idéia central do algoritmo de busca tabu é obter e armazenar uma solução inicial, representada por uma matriz e seu custo.

A partir da solução inicial, o algoritmo implementa a estratégia de busca tabu, percorrendo a vizinhança em busca de melhores soluções que não sejam tabu. O número máximo de iterações que o algoritmo está programado para realizar é especificado no arquivo de configuração, armazenado no objeto *Configuração*. O algoritmo também pode encerrar sua execução antes de atingir esse limite de iterações, caso se verifique que a melhor solução não apresentou ganho após um certo número de iterações.

4.2.6 A primeira proposta de vizinhança

A vizinhança é um dos pontos chave no desenvolvimento de uma busca tabu. Ela define qual o conjunto de soluções para as quais o algoritmo pode se mover a partir da solução atual. Nessa primeira implementação, a proposta de vizinhança consiste em alterar uma das posições da matriz que representa a solução dada, substituindo um dia de trabalho por um dia de folga, ou vice-versa. Dessa forma, a substituição gera uma nova solução que pode ter um custo maior, igual ou menor que o custo da solução atual. É importante observar que posições da matriz que contenham impedimentos, plantões, folgas desejadas ou folgas em haver não são alteradas.

O algoritmo de busca tabu também restringe movimentos de troca quando o movimento proposto está presente na lista tabu. Nessa implementação, a lista tabu é composta por posições da matriz que já foram trocadas desde um número fixo n de trocas anteriores. O parâmetro n está definido no objeto *Configuracao*, parametrizado como *tamanho da lista tabu*, no arquivo de configurações. Se, por exemplo, o tamanho da lista tabu é 2 e uma troca proposta envolve um movimento que está presente nessa lista, o movimento não é considerado, e o algoritmo continua a buscar o próximo vizinho.

4.2.7 A segunda proposta de vizinhança

A segunda proposta de vizinhança consiste em, basicamente, deslocar dias de folga simples e dias de folgas duplas pela linha correspondente ao funcionário. O algoritmo de busca pode ser dividido em dois passos:

Primeiro passo : A partir de uma linha da matriz que representa um funcionário, escolhe-se uma folga simples qualquer. Por exemplo, no caso abaixo, folgas simples são representadas por um S e folgas duplas são representadas por dois Us consecutivos:

S_{i-1}	$\cdots$	$\cdots$	S_i	$\cdots$	$\cdots$	S_{i+1}	$\cdots$	$\cdots$	U	U	$\cdots$	S_t

A folga simples representada por S_i pode ser deslocada alguns dias para a esquerda, sem ultrapassar a posição de S_{i-1}, ou alguns dias para a direita, até encontrar a posição ocupada por S_{i+1}. O resultado no exemplo poderia ser (deslocando S_i totalmente para a esquerda):

S_{i-1}	S_i	$\cdots$	$\cdots$	$\cdots$	$\cdots$	S_{i+1}	$\cdots$	$\cdots$	U	U	$\cdots$	S_t

Segundo passo : Para implementação desse segundo passo, toma-se uma folga dupla em um dos finais de semana do mês (representada por dois Us consecutivos). Essa folga pode migrar quando o deslocamento de folgas no primeiro passo criou uma nova folga dupla em outro fim de semana do mês (representada por dois Ss consecutivos). É possível então trocar as folgas representadas pelos dois Ss com as folgas representadas pelos dois Us. O resultado é a movimentação da folga dupla em final de semana.

Por exemplo, para a mesma configuração representada no primeiro passo, se o encontro das folgas S_i e S_{i-1} corresponder a um sábado e a um domingo consecutivos, essas folgas podem ser transformadas numa folga dupla (dois Us). Nesse caso, a folga dupla representada por dois Us consecutivos à direita de S_{i+1} na configuração anterior ao passo 1, pode migrar para as posições ocupadas por S_i e S_{i-1} após o passo 1. As posições originais dessa folga dupla seriam transformadas em duas folgas simples, S_{t-1} e S_{t-2}. As novas folgas simples criadas podem, agora, ser desmembradas como descrito no passo 1. O resultado deste procedimento pode ser visto abaixo:

U	U	$\cdots$	$\cdots$	$\cdots$	$\cdots$	S_{i+1}	$\cdots$	$\cdots$	S_{t-2}	S_{t-1}	$\cdots$	S_t

A principal característica dessa vizinhança é evitar que algumas restrições do problema sejam violadas, limitando o espaço de busca por vizinhos. As restrições que se mantêm invioláveis incluem o número mínimo de folgas, o número mínimo de folgas em finais de semana e os interstícios entre dias de folga consecutivos. Essa última restrição é atendida impossibilitando todo movimento de folgas (definido no primeiro passo) que gere um interstício entre folgas consecutivas menor que o valor mínimo permitido.

5 Resultados com os Algoritmos Genéticos

A diretoria do HC forneceu 12 diferentes instâncias de escalas reais para a realização de testes com o algoritmo genético. Os testes já incorporaram também dados reais para o histórico de trabalho do mês anterior, de cada funcionário. Nenhuma das escalas criadas tira proveito da *folga altamente desejável*, uma vez que essa facilidade não foi exercitada nas instância reais. Uma avaliação dos algoritmos quando essa opção é ativada vai requerer novos dados reais.

Os seguintes objetivos foram considerados durante os testes:

1. Verificar a capacidade da função objetivo em avaliar todos os critérios desejados pelos usuários no hospital.
2. Encontrar uma combinação de custos para as penalizações que leve a boas escalas.
3. Determinar um ajuste para o funcionamento do algoritmo genético que permita encontrar as escalas de maneira eficiente.

Para manter a quantidade de informações dentro de um limite aceitável, a análise apresenta detalhes somente para a escala de *Maio de 2001, turno da manhã, funcionários auxiliares*. Os resultados com as demais escalas será apresentado no final desta seção, na forma de uma tabela resumida com o valor da função objetivo para cada caso.

A escala de *Maio de 2001, turno da manhã, funcionários auxiliares* é apresentada na Figura 2. Essa escala representa um caso típico encontrado no hospital, pois envolve funcionários plantonistas. A escala apresenta também dois funcionários com impedimentos, um fator complicador. Os impedimentos são bastante comuns, considerando que, devido ao grande número de funcionários, existe uma probabilidade alta de pelo menos um deles estar gozando de suas férias. O quadro de funcionários predominantemente feminino também implica em uma certa quantidade de impedimentos do tipo licença gestante. Outra característica interessante dessa escala é o número razoável de preferências de folga indicados pelos funcionários, algumas delas concentradas em poucos dias, gerando conflitos que devem ser resolvidos pelo algoritmo. Por fim, a diretoria do hospital forneceu também a escala realmente utilizada naquele mês, e que foi desenvolvida manualmente, permitindo o confronto entre os resultados obtidos pelo algoritmo e os dados reais.

Na Figura 2 está indicada apenas a simbologia mais importante. As células em branco indicam que o funcionário não possui impedimentos nem preferências para aquele dia. Os campos marcados com “F” correspondem a dias nos quais o funcionário deseja folgas (preferências). Os funcionários cuja escala está marcada com a letra “P” são plantonistas, folgando nos dias sem marcas e trabalhando nos dias marcados com “P”.

Uma análise preliminar da escala mostra que há um déficit de funcionários, exigindo pelo menos 26 funcionários trabalhando em horas-extra.

5.1 Parâmetros relevantes do algoritmo

A estrutura básica do algoritmo envolve uma variedade de parâmetros. Esses parâmetros definem quais módulos serão utilizados, e como os mesmos devem ser configurados. Aqui

	Dia																															número de	
Funcionário	**1**	2	3	4	**5**	**6**	7	8	9	10	11	**12**	**13**	14	15	16	17	18	**19**	**20**	21	22	23	24	25	**26**	**27**	28	29	30	31	folgas	prefs.
Grace	Impedimento																															7	2
Adelsa						F						F	F	F																		11	3
Cláudia																																11	3
Sílvia	P				P	P						P	P						P	P						P	P					–	–
Margarida					F							F														F						11	3
Eunice																																11	3
Jussara								F					F							F												11	3
Dimiciano									F										F						F							11	3
Elias	Impedimento																															0	0
Fátima				F					F	F	F																					11	3
Juliana																		F	F	F												11	3
Renata								F	F			F																				11	3
Stevers	P				P	P						P	P						P	P						P	P					–	–
Vasconcelos																																11	3
Eliane																																11	3
necessidade	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9		

Figura 2: Escala de Maio de 2001, turno da manhã, funcionários auxiliares

serão descritos somente os itens de maior interesse para os testes. Ou seja, grande parte da configuração foi mantida constante, alterando-se somente os parâmetros discutidos a seguir.

Número de gerações : Indica o máximo de ciclos que o algoritmo realizará. Uma geração corresponde a uma seqüência de avaliação dos cromossomos, seguida de recombinação e mutação. Uma geração não necessariamente renova todos os indivíduos. Um aumento do número de gerações possibilitaria um maior número de tentativas para o algoritmo melhorar a solução, mas acarretaria também um proporcional aumento no tempo de execução. Considerando que os recursos computacionais disponíveis no hospital são limitados, e que se deseja uma ferramenta semi-interativa, optou-se por manter o número de gerações razoavelmente baixo, sem afetar demasiadamente a qualidade final das soluções.

Tamanho da população : Indica quantos indivíduos (no caso, escalas) serão tratados em paralelo pelo algoritmo. Quanto maior este valor, mais escalas serão analisadas simultaneamente, e maiores serão as chances de se encontrar boas soluções. No entanto, este parâmetro também influencia diretamente no tempo de execução, e se possível, deve mantido em valores baixos.

Tamanho da elite : A elite é formada pelos cromossomos que obtiveram a melhor avaliação (no caso, menor custo) em toda a população. Este parâmetro indica quantos cromossomos formam a elite. Todos os cromossomos pertencentes à elite, além de concorrerem à recombinação, são copiados diretamente para a nova geração, sem sofrer qualquer tipo de alteração. Com este parâmetro o algoritmo mantém as melhores escalas encontradas durante as gerações.

Taxa de renovação entre gerações : Este parâmetro é a porcentagem da população da próxima geração que será criada a partir de recombinação e mutação. Este parâmetro não inclui a elite da população. Se a taxa for menor que um, então a população da próxima geração será completada copiando-se cromossomos escolhidos aleatóriamente, tendo como base seu valor de *fitness*. Vale notar que, sendo G o número de gerações, T a taxa de renovação e P o tamanho da população, então o número N de novos cromossomos gerados pelo algoritmo é dado por $N = G \times P \times T$.

Ajuste do valor da função objetivo : Dependendo da distribuição dos valores de *fitness* gerados pela avaliação dos cromossomos, o método que escolhe os cromossomos para recombinação pode sofrer de alguns problemas, como descrito anteriormente. Para aliviar esses problemas, os resultados da função objetivo são mapeados linearmente entre dois valores limitantes. O limite superior será atribuído ao melhor cromossomo e, de maneira análoga, o pior cromossomo será mapeado para o limite inferior. Se os limites deste intervalo forem próximos e elevados, então haverá pouca pressão de seleção sobre os cromossomos.

Outro item relevante para este ajuste é o fato de que o algoritmo procura minimizar o valor da função objetivo. Em geral, algoritmos genéticos são heurísticas que, usualmente, maximizam uma função objetivo. Para transformar o problema para maximização, nessa implementação, usa-se o inverso da função de *fitness* no mepeamento linear.

Porcentagem de mutação : Esse parâmetro indica com que probabilidade cada novo cromossomo sofrerá mutação. Valores elevados introduzem maior variedade na solução e a busca tende a cobrir mais pontos do espaço de busca. Já valores baixos preservam as (boas) características da solução atual. Uma porcentagem extremamente alta seria semelhante a uma busca aleatória.

Esse parâmetro também influi na robustez da ferramenta computacional. Quanto maior a taxa de mutação, maiores as diferenças entre as respostas obtidas entre diferentes execuções do algoritmo. Para uma mesma escala, em algumas execuções o algoritmo poderia encontrar soluções boas, e em outras violar um número maior de restrições.

Custo das violações : Esses valores são de fundamental importância, pois definem como a função objetivo penalizará as violações. Normalmente, os valores ocorrem aos pares, tal como descrito na seção sobre a função objetivo. O primeiro valor será utilizado para penalizar a primeira ocorrência de uma violação, e o segundo valor penaliza as próximas violações do mesmo tipo. Como, após a avaliação da função objetivo, ocorre um mapeamento para um intervalo fixo, a ordem de grandeza dos custos de violações pouco influi no comportamento do algoritmo, predominando a relação de proporções entre os custos.

5.2 Estratégia de testes e escolha de parâmetros

Considerando a quantidade de parâmetros disponíveis, alguns dos quais admitem uma infinidade de valores, o próprio trabalho de determinar a melhor configuração do algoritmo formaria um novo problema de otimização por si só. Decidiu-se manter os parâmetros fixos, variando somente um item em cada teste.

O custo das violações é um dos parâmetros mais importantes para uma boa performance do algoritmo. Os valores dos custos das penalizações foi obtido experimentalmente durante sucessivas avaliações dos resultados junto aos usuários do hospital. Na presença de novas sugestões, alguns parâmetros eram ajustados para adaptar o comportamento do algoritmo de acordo com as prioridades exigidas pelo hospital. Nesses experimentos, as decisões foram baseadas principalmente em avaliações intuitivas por parte dos usuários.

5.2.1 Escolha dos custos das violações

Inicialmente, com o intuito de minimizar as horas-extra, foram escolhidos custos altos para as violações ligadas às restrições correspondentes. As escalas resultantes geraram uma certa insatisfação entre os usuários, e a coordenação das enfermarias preferiu dar maior importância às violações causadas por preferências de foga não atendidas, além de procurar distribuir melhor as folgas em finais de semana entre todos os funcionários. Experimentos mostraram que com custos relativamente altos para violações associadas a folgas nos finais de semana e para violações sobre preferências de folgas, o algoritmo passou a gerar escalas com as propriedades desejadas, sem aumentar significativamente o número de hora-extras. Essa filosofia foi, então, incorporada ao conjunto de parâmetros.

Para evitar que um dia receba um número excessivamente alto de horas-extra, a violação dos limitantes de horas-extra por dia e por mês foi assodiada à custos relativamente altos. Verificou-se que o custo deveria ser no mínimo maior que o dobro dos custos das outras violações. Evita-se, assim, que o algoritmo extrapole o limite de horas-extra para atender por exemplo, uma preferência de folga, ou uma folga em final de semana.

Os seguintes custos foram propostos para as penalizações:

1. *Excesso de funcionários em um dia*: 10 para o primeiro excesso e 5 para os demais excessos.

2. *Falta de funcionário em um dia* (implica horas extras): 50 para a primeira falta e 25 para as demais faltas.

3. *Período de trabalho no qual um funcionário trabalha mais dias consecutivos que o máximo permitido* (período muito longo): 50 para o primeiro dia em excesso do período, 30 para os demais dias em excesso.

4. *Período com poucos dias consecutivos de trabalho* (período muito curto): 50 para o primeiro dia, 30 para os demais dias.

5. *Falta de folgas para um funcionário*: 70 para a primeira folga faltante, 40 para as demais necessárias para atingir o número correto.

6. *Excesso de folgas*: 20 para a primeira folga em excesso, 10 para as demais.
7. *Falta de um fim de semana*: 70 para a primeira e 30 para as demais.
8. *Privilégio sobre fins de semana*, explicado na seção sobre a função objetivo: 1 para cada privilégio a mais sobre o funcionário menos privilegiado.
9. *Pesos para cálculo do privilégio*: fim de semana completo: 3; sábados e domingos: 1.
10. *Mínimo de preferências não atendido*: 70 para a primeira preferência em falta, 60 para as demais.
11. *Preferências extras não atendidas (além do mínimo exigido)*: 10 tanto para a primeira e quanto para as demais.
12. *Número de horas-extra acima do limite máximo total*: 200.
13. *Número de horas-extra acima do limite máximo em um dia*: 150 para a primeira hora-extra em excesso, 100 para as demais.

Uma vez fixado um conjunto de penalizações adequadas, passou-se à etapa seguinte, na qual procurou-se determinar como variações nesses parâmetros influenciam na velocidade de processamento do algoritmo e na qualidade da solução produzida.

5.3 Escolha dos outros parâmetros

Os parâmetros iniciais foram escolhidos seguindo-se um certo bom senso e apoiado-se na experiência adquirida durante os testes e depuração dos módulos. As escolhas foram as seguintes:

1. *Número de gerações*: 6000. Este valor foi escolhido considerando-se o tempo de execução dos algoritmos. Verificou-se que somente poucas situações necessitavam de um número ainda maior de iterações e, mesmo nessas situações, os ganhos obtidos eram muito pequenos. Preferiu-se privilegiar um melhor ajuste dos outros parâmetros sem alargar o número de gerações consideradas, evitando tornar mais lenta a execução do algoritmo.
2. *Tamanho da elite*: 1. Foi utilizado o menor valor possível que mantivesse as melhores escalas entre as gerações.
3. *Tamanho da população*: 300. Durante os testes preliminares, verificou-se que somente populações maiores que 200 eram capazes de alcançar soluções razoáveis.
4. *Taxa de reposição*: 99%. Seguindo a estrutura dos algoritmos clássicos encontrados na literatura, decidiu-se repor toda a população a cada geração. No entanto, não seria possível escolher 100%, pois há um cromossomo na elite, que não será substituído.
5. *Intervalo de mapeamento da função objetivo*: de 7 a 2. Essa escolha foi arbitrária e será discutida mais detalhadamente em seguida.
6. *Taxa de mutação*: 20%. Este valor também foi escolhido arbitrariamente.

5.4 Forma de avaliação dos resultados

Cada avaliação apresenta dois gráficos. O primeiro gráfico compara os custos entre as configurações. Os custos em questão correspondem ao valor retornado pela função objetivo para as escalas geradas pelo algoritmo. Como estamos tratando de um problema de minimização, quanto menor o custo, melhor a escala. A barra cinza indica a média dos custos das soluções. As barras branca e preta correspondem ao custos da melhor e da pior escala, respectivamente, encontradas com a configuração em foco. Este gráfico permite observar qual foi o melhor ajuste para um certo parâmetro, e permite também verificar qual a robustez desse ajuste, i.e., quanto o custo varia de uma execução para a outra.

O segundo gráfico representa o custo médio dos cromossomos da população versus o número da geração. O custo, tal como no gráfico de barras, é dado pela função objetivo. Com este gráfico pode-se avaliar quão cedo o algoritmo convergiu para uma solução encontrada, e pode-se inferir se mais gerações beneficiariam a solução final ou simplesmente desperdiçariam tempo de execução.

Vale lembrar que o valor apresentado para o custo está relacionado a dois fatores: à quantidade de horas-extra e à possíveis violações. Como todas as escalas exigem horas-extra para completar o quadro de funcionários, não seria possível gerar uma escala de custo zero, mesmo se essa fosse a escala ótima. Assim, todas as escalas possuem um custo intrínseco associado às horas-extra inevitáveis. Pode-se determinar um limitante inferior para o número de horas-extra para a solução ótima da seguinte maneira. Cada escala exige um certo número de horas de trabalho durante o mês todo e fornece, através do quadro de funcionários, uma certa disponibilidade de horas de trabalho. O déficit precisa ser compensado com horas-extra.

O conceito de *funcionário-dia* foi tomado como a unidade de trabalho de uma escala. Como não há funcionários trabalhando em frações de turnos, e cada turno dispõe de seu corpo de funcionários, um funcionário-dia corresponde a um funcionário em um turno. Para determinar-se o número total de funcionários-dia necessários, basta somar a necessidade de funcionários por dia ao longo de todo o mês, no turno correspondente. A disponibiliade total de funcionários-dia pode ser calculada somando-se os dias não impedidos de todos os funcionários não plantonistas ao longo do mês, e subtraindo-se desse valor o número total de folgas que devem ser atribuídas na solução final. Sempre que houver um funcionário plantonista em serviço num determinado dia, a escala exigirá um profissional do regime de trabalho normal a menos nesse mesmo dia. Portanto, deve-se somar à disponibilidade total também o total de plantões previstos antecipadamente para a escala. Calculando-se a diferença entre a necessidade e a disponibilidade de funcionários-dia, pode-se obter o número mínimo teórico de horas-extra para a escala ideal multiplicando-se essa diferença pelo número de horas de trabalho no turno correspondente.

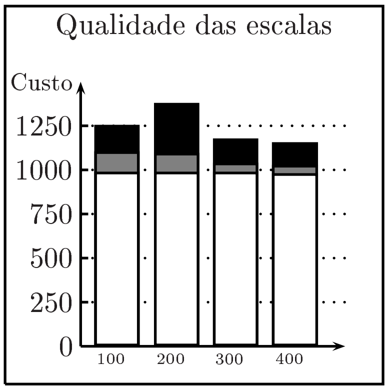
Cada teste consiste em 15 execuções do algoritmo genético com os mesmos parâmetros.

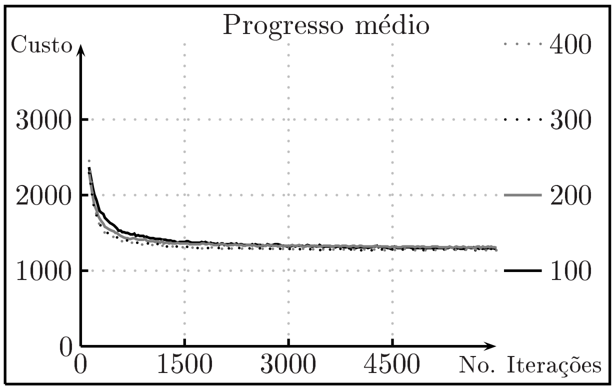
5.5 Testes com a primeira versão

A primeira versão foi submetida a uma extensas bateria de testes, descritos em seguida, onde estão agrupados por tópicos.

5.5.1 Tamanho da população

Neste tópico avaliou-se a influência do número de gerações na qualidade da solução final. Mantendo-se fixos todos os outros parâmetros, realizou-se testes com populações de 100, 200, 300 e 400 cromossomos.





Análise: Nota-se que, em todos os casos, a melhor escala não apresenta nenhuma violação das restrições e atinge o mínimo de 26 funcionários-dia em hora-extra.

Para a população de 100 indivíduos, ocorreram em média 1,5 violações, e 4 violações no pior caso. Com o aumento da população, a média de violações caiu para 0,55, mas o pior caso manteve-se inalterado. Nota-se também que no caso médio a escala melhora com o aumento da população, fato bastante razoável pois o algoritmo analisa mais indivíduos, e portanto acessa mais possibilidades.

As violações concentraram-se no atendimento de preferências e algumas poucas estavam associadas aos finais de semana de folga.

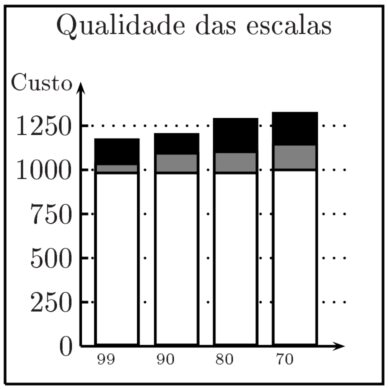
O tempos de execução aproximados, em segundos, foram de: 50, 100, 155, 215. O tamanho da população praticamente não influência na velocidade de convergência do algoritmo.

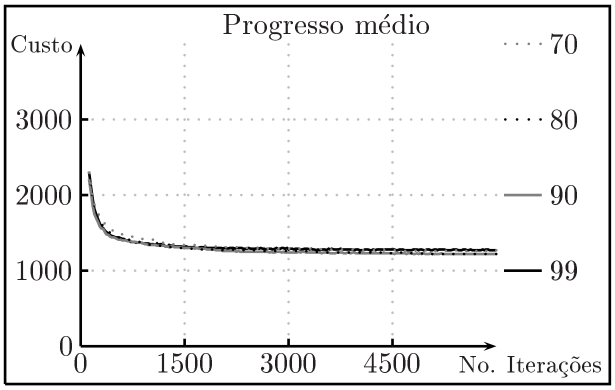
Conclusão: Trata-se na verdade de uma questão de custo-benefício. Como o tempo de execução aumenta em mais de 38% quando aumentamos a população de 300 para 400 indivíduos, sem diminuir significativamente o número médio de violações, nem mesmo no pior caso, considera-se 300 cromossomos como uma alternativa razoável para o tamanho da população.

Esta configuração permite atingir o número ótimo de horas-extra, com uma quantidade reduzida de violações.

5.5.2 Taxa de reposição

Neste teste verificamos como a porcentagem de cromossomos substituídos por geração influencia o comportamento do algoritmo. Realizaram-se testes com taxas de reposição de 99% (sendo 1% para a elite), 90%, 80% e 70%.





Análise: Tanto uma observação do gráfico, quanto uma análise detalhada das escalas geradas sugerem que a diminuição na taxa de reposição tende a aumentar o número médio de violações, como também tende a exacerbar o pior caso.

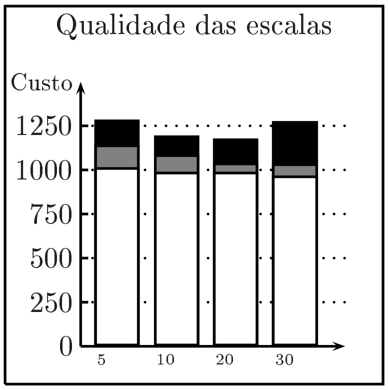
A quantidade de funcionários-dia em horas-extra manteve-se sempre no mínimo possível para o problema, e a velocidade de convergência permaneceu inalterada.

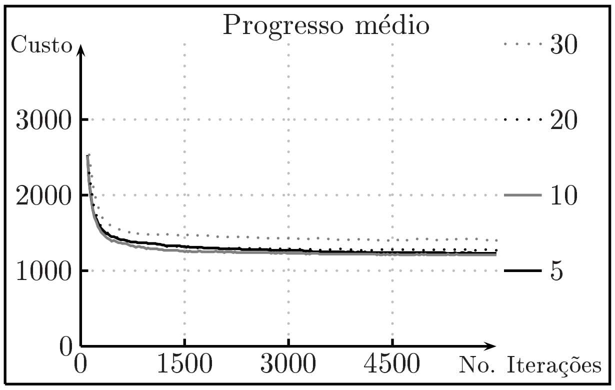
O tempo de execução com taxas menores foi levemente menor, provavelmente devido ao fato do que o algoritmo realiza um número menor de operações de recombinação e mutação. É plausível supor que a queda no tempo de execução com a diminuição da taxa de reposição se deve ao fato que são gerados menos cromossomos.

Conclusão: Não parece haver motivos para escolher-se uma taxa de reposição diferente da substituição total da população ao longo das gerações.

5.5.3 Probabilidade de mutação

Neste teste, avaliou-se a influência de diferentes taxas de mutação. Foram testadas taxas de 5%, 10%, 20% e 30%.





Análise: Nota-se que o aumento da taxa de mutação tende a melhorar as soluções do algoritmo. No entanto, a partir de 30%, surge também um efeito indesejável, que é a perda de estabilidade. A partir de taxas de mutação altas, a qualidade da solução começa a variar bastante entre as diferentes execuções com os mesmos parâmetros.

Em todos os casos, o algoritmo sempre foi capaz de atingir o número mínimo de horas-extra.

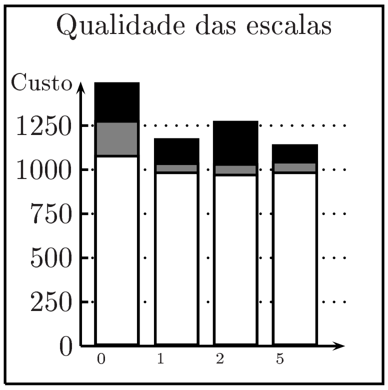
Em relação às violações, o número médio tende a decrescer com o aumento da taxa de mutação. No entanto, o pior caso manteve-se inalterado. As violações concentraram-se no atendimento de preferências e em um único caso na falta de folga em um final de semana.

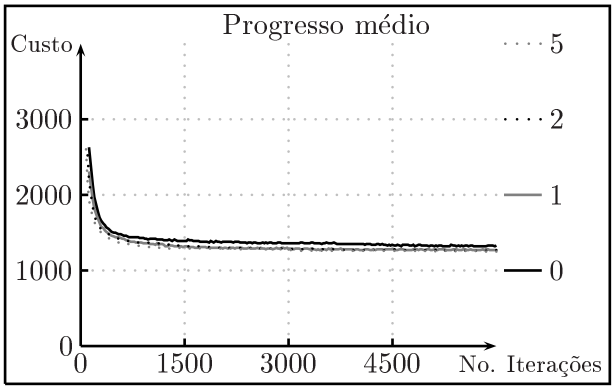
O tempo de execução manteve-se entre 150 e 160 segundos para todos os testes.

Conclusão: As taxas de 10% e de 20% apresentaram os melhores custos finais. No entanto, a taxa de 20% gerou um número menor de violações. Uma taxa de 30% também gera soluções adequadas, mas induz uma variabilidade muito grande no custo, tornando-a menos confiável.

5.5.4 Tamanho da elite

Neste teste analisou-se a influência do tamanho da elite no comportamento algoritmo, principalmente na sua velocidade de convergência. Uma elite de tamanho zero implica em um algoritmo genético cuja memória baseia-se somente nos cromossomos. O algoritmo corre o risco de perder sua melhor solução durante a busca, quando recombina sua melhor solução. Tamanhos maiores da elite evitam o problema, mas a garantia de sempre haver uma solução muito boa pode viciar o algoritmo em uma certa direção, diminuindo a cobertura do espaço de busca.


Qualidade das escalas


Progresso médio

Análise: Uma importante verificação é o fato de que não se percebeu grandes discrepâncias entre os resultados dos testes com e sem a elite. Se as soluções com elite fossem muito melhores que as soluções sem a elite, isso indicaria que o algoritmo se aproximaria uma busca aleatória de soluções.

Todos os tamanhos de elite maiores que zero apresentaram resultados semelhantes, sempre atingindo o mínimo de horas-extra e um número médio de violações menor que 1.

Para os testes sem elite, no entanto, observou-se uma grande variabilidade no custo das soluções, além de um número relativamente alto tanto no número médio de violações, como também para valor do pior caso.

Como esperado, o tempo de execução manteve-se invariável em entre aproximadamente 150 e 160 segundos, para todos os testes.

Conclusão: A cópia dos melhores cromossomos para a próxima geração mostrou-se realmente eficaz, pois gera soluções mais próximas da ótima e diminui a variabilidade nos custos das soluções.

Este efeito já pode ser alcançado com uma população de elite formada por um único cromossomo. Uma elite maior só implica em menos cromossomos que sofrem recombinação, limitando a exploração do espaço de busca.

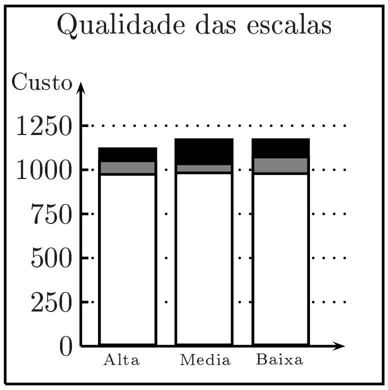
Este teste permitiu comprovar a funcionalidade do algoritmo.

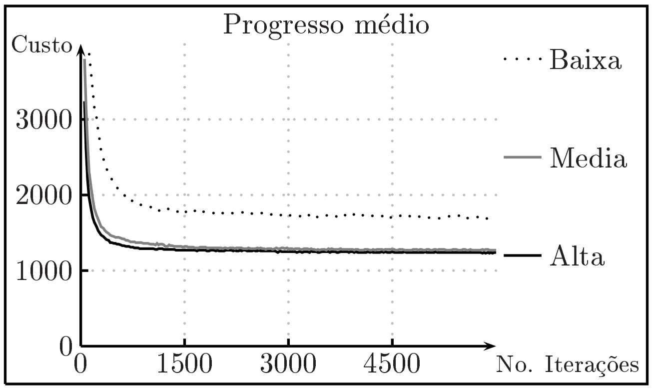
5.5.5 Pressão na seleção de cromossomos

Este teste produziu resultados interessantes, permitindo avaliar quão cedo o algoritmo encontra sua melhor solução e, ainda mais significativo, quando descarta mais cromossomos no processo de recombinação.

Caso os cromossomos convirjam prematuramente, perde-se qualidade na solução, pois ao se privilegiar demasiadamente os melhores cromossomos na recombinação, deixa-se de utilizar muitos cromossomos de bom custo, e que poderiam conter esquemas interessantes para formar uma solução melhor. Por outro lado, se desacelerarmos a convergência o algoritmo precisará de um número maior de gerações para encontrar uma solução adequada pois, nesse caso, estaria considerando cromossomos de menor qualidade. A pressão de seleção pode ser ajustada variando-se os limites da função de mapeamento linear utilizada.

Realizou-se testes com pressão forte, média e baixa, equivalendo, respectivamente, a mapeamentos nos intervalos [11-1], [7-2] e [11-7].





Análise: Tanto os testes com pressão alta como os testes com pressão média resultaram em um comportamento semelhante pois, nos dois casos, as relações entre o peso do melhor e do pior cromossomo ficaram muito próximas. No teste com pressão baixa, no entanto, o melhor cromossomo não recebe um custo significativamente melhor que o do pior cromossomo e, portanto, o algoritmo apresentou um comportamento diferente. O gráfico do progresso revela que o teste com pressão baixa encontrou dificuldades para convergir para um custo mais baixo do que aqueles obtidos com os outros ajustes.

Em todos os casos, as melhores soluções encontradas com cada configuração apresentaram um custo muito próximo.

Conclusão: Uma pressão baixa na seleção dos cromossomos não resultou nos benefícios desejados. Um ajuste conciliatório interessante seria usar o intervalo [7-2].

5.5.6 Conclusão sobre os testes com a primeira versão

Após analisar o efeito dos diferentes parâmetros, decidiu-se escolher mais de uma possibilidade para um ajuste final. Em algumas escalas mais simples, todos os ajustes, mesmo os menos eficientes, geraram soluções praticamente ótimas. Outras precisam de mais cuidado na determinação dos parâmetros. Observa-se, por exemplo, que o custo da solução decresce rapidamente nas primeiras 2.000 gerações. Depois, continua em uma queda leve, que persiste no máximo até a geração de número 8000 ou 10000. Além disso, os fatores que realmente influem no tempo de execução do algoritmo são o número de gerações e o tamanho da população. Decidiu-se, portanto, criar quatro conjuntos de parâmetros, e permitir ao usuário escolher, através da interface, qual destes conjuntos vai utilizar no cálculo das escalas.

O primeiro ajuste foi denominado **Expresso**. A intenção deste ajuste é permitir ao usuário resolver rapidamente escalas muito pequenas ou extremamente simples, sem consumir muito tempo. Pode também ser utilizado para se gerar soluções parciais ainda durante a entrada dos dados.

O ajuste aproveita o fato de que um número pequeno de gerações converge para soluções adequadas, se usarmos uma população apenas suficientemente grande para não gerar soluções de má qualidade. Taxas de mutação altas também não permitem ao algoritmo convergir rapidamente.

Os parâmetros escolhidos para esse modo foram:

1. Número de gerações: 2000
2. Tamanho da elite: 1
3. Tamanho da população: 200
4. Taxa de reposição: 99%
5. Taxa de mutação: 40%
6. Intervalo de mapeamento: 7 – 2

O segundo ajuste, denominado **Rápido**, tem como objetivo produzir escalas adequadas, mas com tempo de execução reduzido. Este ajuste utiliza os melhores valores encontrados em cada teste descrito anteriormente, mas limita o número de gerações e o tamanho da população, para economizar tempo de execução. Os parâmetros escolhidos para esse modo foram:

1. Número de gerações: 4000
2. Tamanho da elite: 1
3. Tamanho da população: 250

4. Taxa de reposição: 99%
5. Taxa de mutação: 20%
6. Intervalo de mapeamento: 7 – 2

O terceiro ajuste, chamado de **Médio**, emprega o melhor conjunto de parâmetros que puderam ser determinados a partir dos testes realizados.

Os parâmetros escolhidos para esse modo foram:

1. Número de gerações: 6000
2. Tamanho da elite: 1
3. Tamanho da população: 300
4. Taxa de reposição: 99%
5. Taxa de mutação: 20%
6. Intervalo de mapeamento: 7 – 2

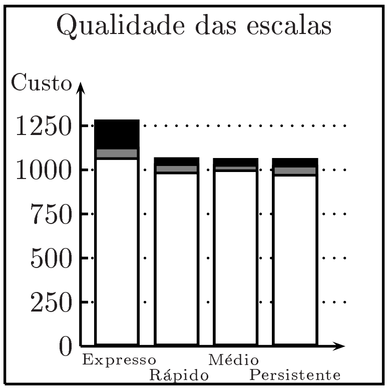
O último ajuste, chamado de **Persistente**, difere do ajuste médio somente no número maior de iterações que abriga. Com esse ajuste, o usuário terá opção de tentar encontrar uma solução ainda melhor para uma escala já determinada por um dos ajuste anteriores, caso disponha de tempo para isso. Ou então, representa uma alternativa para uma escala que venha mostrar-se extremamente difícil de se construir.

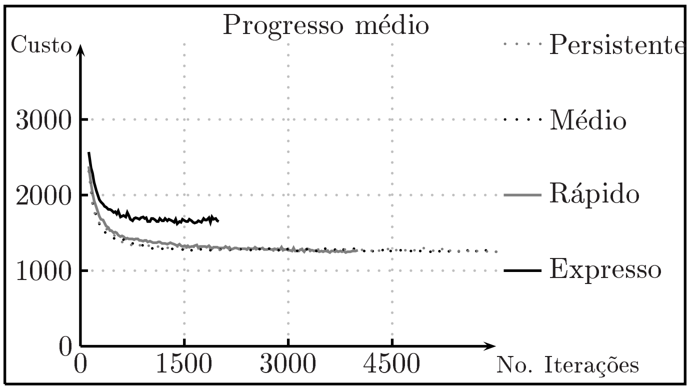
Os parâmetros escolhidos para esse modo foram:

1. Número de gerações: 10000
2. Tamanho da elite: 1
3. Tamanho da população: 300
4. Taxa de reposição: 99%
5. Taxa de mutação: 20%
6. Intervalo de mapeamento: 7 – 2

O gráfico a seguir mostra o comportamento de cada modo. Nota-se que o comportamento do ajuste denominado Expresso não converge para soluções da mesma qualidade que os outros ajustes. A forte ondulação em sua curva de progresso médio é conseqüência da alta taxa de mutação.

Para a escala de *Maio de 2001, turno da tarde, funcionários auxiliares*, os ajustes Rápido, Médio e Persistente apresentaram, praticamente, o mesmo comportamento. Isto significa que o ajuste Rápido seria a opção mais interessante para resolver essa instância.





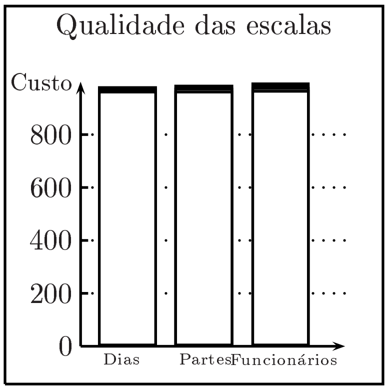
5.6 Testes com a segunda versão

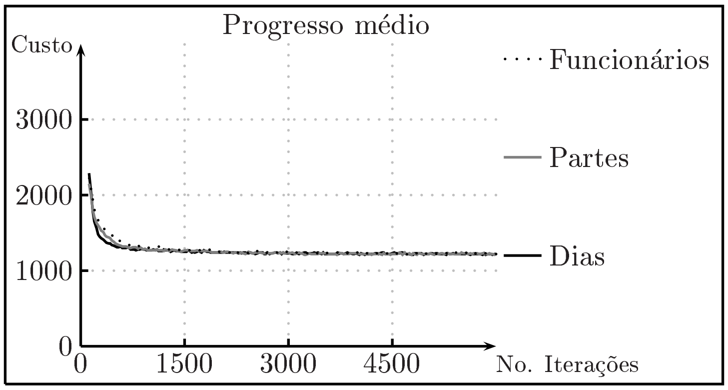
Os testes com segunda versão do algoritmo utilizaram o mesmo conjunto de parâmetros determinados, na subseção anterior, para o modo denominado Médio, com a população diminuída de 300 para 250 indivíduos. Os testes variaram os métodos de recombinação, mutação e os inicializadores dos cromossomos, usando os métodos implemantados para tal na segunda versão do algoritmo. Como as novas estruturas reduzem o espaço de busca, ao manter sempre constantes e corretos o número de folgas para cada funcionário, espera-se uma melhoria no comportamento do algoritmo implementado nessa versão.

5.6.1 Método de recombinação

Nesse teste, avaliou-se a influência de três métodos de recombinação implementados para a terceira versão:

1. Método 1, denominado **Funcionários**: as trocas ocorrem somente em posições que delimitam as seqüências de bits entre funcionários. Ou seja, as escalas dos funcionários são mantidas inalteradas, mas o método troca as escalas de funcionário entre os cromossomos.
2. Método 2, denominado **Partes**: realiza uma troca da parte inicial do cromossomo. O comprimento dessa parte é tal que não se viole o número de folgas.
3. Método 3, denominado **Dias**: realiza uma troca considerando os dias de cada funcionário, ainda sem alterar o número de folgas atribuído a cada funcionário.





Análise: Os resultados obtidos foram superiores aos colhidos com a primeira versão. Confrontando os resultados, notamos que os novos métodos de recombinação permitiram sempre encontrar soluções ótimas para a escala. Todos os métodos foram capazes de gerar o mínimo de horas-extra, a média de violações manteve-se muito perto de zero ($<< 0,1$) e o pior caso gerou somente uma única violação.

Essa nova versão do algoritmo melhorou todas as escalas testadas, quando comparadas com a versão anterior. A tabela abaixo resume os resultados obtidos, mostrando o número de violações da melhor escala encontrada, o número médio de violações e também o número de violações da pior escala.

Considerando que, durante 75 avaliações da mesma escala, o pior caso ocorreu somente 2 vezes, pode-se afirmar que a segunda versão do algoritmo possui qualidades interessantes para gerar escalas de serviço adequadas para o HC.

Comparativo entre o número de violações de cada versão

Escala	**Versão 1**			**Versão 2**		
	Mín.	Méd.	Máx.	Mín.	Méd.	Máx.
Agosto 2001 Manhã Auxiliares	0	0,3	2	0	0,13	2
Agosto 2001 Manhã Enfermeiros	1	3,55	11	1	1,93	4
Maio 20001 Manhã Auxiliares	0	0,6	3	0	0	0
Maio 20001 Manhã Enfermeiros	0	0	0	0	0	0
Maio 20001 Tarde Auxiliares	0	0	0	0	0	0
Maio 20001 Tarde Enfermeiros	0	0	0	0	0	0
Maio 1999 Tarde Auxiliares	0	0,5	1	0	0	0
Maio 1999 Tarde Enfermeiros	0	0,15	1	0	0,06	1
Setembro 20001 Manhã Auxiliares	0	1,15	6	0	0,33	2
Setembro 20001 Manhã Enfermeiros	0	0,3	2	0	0	0
Setembro 20001 Tarde Auxiliares	0	0,4	4	0	0	0
Setembro 20001 Tarde Enfermeiros	0	0	0	0	0	0

Os três métodos apresentaram um comportamento muito semelhante, tanto para o custo da solução final, como no decorrer do algoritmo. No entanto, o terceiro método (Dias), incorreu numa menor variação nos resultados e convergiu de forma mais acentuada nas primeiras iterações.

O tempo de execução da segunda vesão manteve-se na mesma faixa de 100 a 150 segundos, tal como na primeira versão do algoritmo, apesar de apresentar uma codificação mais complexa. A função objetivo, por outro lado, é verificada de forma mais expedita, uma vez que não foi necessário verificar o número de folgas para cada funcionário, que foi sempre mantido correto. Adaptando-se a função objetivo, foi possível reduzir o tempo de execução.

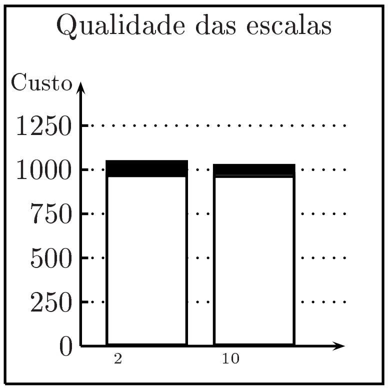
Conclusão: A segunda versão do algoritmo genético mostrou-se bem mais eficiente que a primeira versão, obtendo soluções ótimas para a maioria das escalas verificadas. Sua principal vantagem é reduzir o custo do pior caso, além de melhorar a média dos resultados e reduzir a variabilidade de custo entre as escalas calculadas.

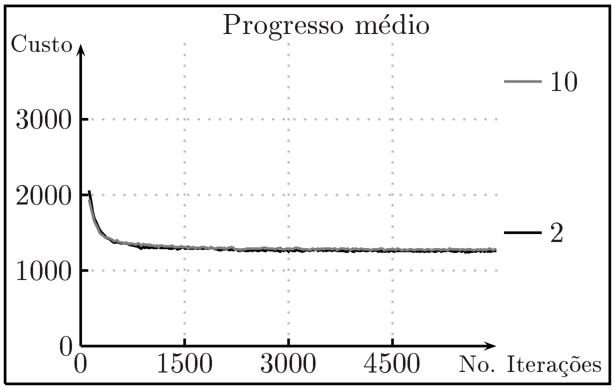
Os métodos de recombinação revelaram um desempenho semelhante, mas o terceiro método, que realiza uma troca para cada funcionário do cromossomo, gerou resultados com menor variação de custos.

5.6.2 Teste sem repetições de cromossomos

A literatura menciona que um algoritmo genético poderia sofrer uma queda de desempenho caso os métodos de mutação e recombinação venham a introduzir indivíduos iguais na população. A segunda versão do algoritmo incorpora uma heurística para evitar que a mutação ou a recombinação venham a gerar cromossomos duplicados. Testou-se também se esta preocupação é realmente pertinente no problema do HC.

O primeiro ajuste realiza somente duas tentativas de recombinação ou mutação, caso perceba que esteja prestes a introduzir um cromossomo duplicado. O segundo ajuste chega a realizar até 10 tentativas.





Conclusão: Os gráficos e a comparação das escalas não mostrou nenhum ganho aparente ao insistir em evitar repetições.

5.7 Comparativo com soluções manuais

Nessa subseção compara-se a solução obtida manualmente pela administração do hospital com uma escala gerada pelo algoritmo genético.

Solução manual:

	Dia																														
Funcionário	**1**	2	3	4	**5**	**6**	7	8	9	10	11	**12**	**13**	14	15	16	17	18	**19**	**20**	21	22	23	24	25	**26**	**27**	28	29	30	31
Grace	Impedimento									F					F	F				F		F					F			F	
Adelsa		F	F			F	F					F	F	F						F				F				F	F		
Cláudia	F					F	F				F		F					F			F				F		F	F		F	
Sílvia	P				P	P						P	P						P	P						P	P				
Margarida				F	F						F	F		F				F	F		F				F	F		F			
Eunice	F					F	F				F					F	F					F	F			F	F				F
Jussara			F					F		F			F				F		F	F				F			F		F		F
Dimiciano		F			F				F			F				F			F		F		F		F	F				F	
Elias	Impedimento																														
Maria			F	F					F	F	F				F		F						F	F					F		F
Juliana	F			F						F				F	F			F	F	F						F	F	F			
Renata	F				F	F		F	F			F			F					F	F				F					F	
Elaine	P				P	P						P	P						P	P						P	P				
Vasconcelos		F			F	F					F		F	F			F			F		F				F	F				
Eliane			F				F			F		F	F					F	F					F	F					F	F
necessidade	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
disponível	9	8	7	8	9	7	8	9	8	7	7	9	9	8	8	9	8	8	9	8	8	9	9	8	7	9	8	8	9	7	8
horas extras		1	2	1		1	2		1	2	2			1	1		1	1		1	1			1	2		1	1		2	1

Solução do algoritmo genético:

	Dia																														
Funcionário	**1**	2	3	4	**5**	**6**	7	8	9	10	11	**12**	**13**	14	15	16	17	18	**19**	**20**	21	22	23	24	25	**26**	**27**	28	29	30	31
Grace	Impedimento											F	F					F	F							F	F			F	
Adelsa	F			F	F	F					F	F	F	F			F						F				F				
Cláudia	F				F	F				F				F	F	F			F	F					F						F
Sílvia	P				P	P						P	P						P	P						P	P				
Margarida	F				F				F			F	F				F				F				F	F	F				F
Eunice		F	F				F					F	F			F			F	F	F						F	F			
Jussara					F			F			F	F	F				F			F	F	F	F						F		
Dimiciano					F	F			F				F	F					F	F				F	F					F	F
Elias	Impedimento																														
Maria				F					F	F	F						F				F			F	F	F	F			F	
Juliana	F				F	F						F					F	F	F	F	F	F							F		
Renata		F	F					F	F			F	F						F	F			F	F						F	
Elaine	P				P	P						P	P						P	P						P	P				
Vasconcelos				F			F				F				F	F						F				F	F	F	F	F	
Eliane				F						F	F				F			F	F	F					F	F	F	F			
necessidade	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
disponível	9	9	9	7	7	9	9	9	7	9	7	7	7	9	9	9	7	9	7	7	7	9	9	9	7	9	7	9	9	7	9
horas extras				2	2				2		2	2	2				2		2	2	2				2		2			2	

Análise: A escala manual apresenta um número alto de dias de trabalho intercalados com folgas, por exemplo o dia 21 para a funcionária Grace, ou então nos dias 12, 26 e 29 para a funcionária Cláudia. No total, somam-se 20 violações desta natureza presentes na solução manual. Na escala manual, muitos dos funcionários não receberam folga em nenhum final de semana, como por exemplo, os funcionários Grace, Cláudia, Margarida, Dominiciano e Maria. Já o algoritmo genético foi capaz de distribuir folgas em *dois* finais de semana para a maioria dos funcionários. O funcionário menos privilegiado recebeu somente uma folga em final de semana mas, em compensação, este mesmo fucnionário recebeu mais um sábado e outro domingo de folga. A escala manual também distribui mal as folgas nos finais de semana. Por exemplo, enquanto a Juliana recebeu dois finais de semana de folga, a Maria não recebeu nenhum. Nesse caso, a diferença máxima é de dois finais de semana. O algoritmo genético gerou um diferença de somente um final de semana entre o funcionário mais privilegiado e o menos privilegiado. O número de folgas atribuídas foi o correto, para todos os funcionários nas duas escalas. A solução manual dificilmente concede mais de duas folgas consecutivas. A gerada pelo algoritmo pode agrupar folgas em bloco maiores mas, segundo a diretoria da enfermagem, esta característica não é prejudicial.

A escala manual atende todas as preferências. Conforme observado nas reuniões com a diretoria das enfermarias, ao elaborar a solução manual inicialmente são distribuídas folgas para as preferências, para só então decidir como distribuir as demais folgas. Em casos de conflitos de preferências difíceis de resolver manualmente, a diretoria as renegociava com os funcionários envolvidos. O algoritmo genético não utiliza esta estratégia "gulosa" mas, mesmo assim, atende à toda as preferências.

As duas escalas utilizam o mínimo de horas-extra, ou seja, o algoritmo genético foi capaz de sugerir uma escala muito melhor, mas sem introduzir novas horas-extra.

De maneira resumida, a escala manual gera 25 violações e utiliza 26 funcionários-dia em hora-extra. Segundo a função objetivo, seu custo é de 2316. A escala gerada pelo algoritmo, não apresenta nenhuma violação, utiliza o mesmo número mínimo de 26 horas extras e resulta em um custo menor, de 992.

Conclusão: Com o auxílio da heurística desenvolvida, a diretoria dessa enfermaria poderia aumentar drasticamente a qualidade de suas escalas de trabalho. Basta dizer que a escala manual é fruto de um trabalho intensivo, que ocupa o coordenador responsável por um tempo considerável, enquanto que o algoritmo gerou uma escala bem melhor em apenas dois minutos de execução em um computador de médio porte (AMD Athlon de 600MHz).

5.8 Valores numéricos dos resultados

No final desta seção documento seção apresentaremos os resultados numéricos para cada teste utilizado na avaliação dos algoritmos.

Cada coluna corresponde a uma instância real fornecida pelo hospital. As linhas correspondem ao parâmetros utilizados.

Em cada célula da tabela, o primeiro número indica o custo da melhor escala encontrada com o ajuste correspondente a linha. O segundo número expressa a média dos custos com o ajuste. O último número corresponde ao custo da pior escala gerada pelo ajuste.

Todos os valores a foram calculados avaliando as escalas produzidas com a mesma função objetivo.

6 Resultados com a Busca Tabu

Esta seção é dedicada a apresentação de alguns resultados obtidos pelo algoritmo de busca tabu para entradas reais fornecidas pelo hospital. A seção está organizada de forma a possibilitar a experimentação com o algoritmo de busca tabu em situações controladas, de forma a permitir avaliar seu comportamento quando se variam determinados parâmetros importantes.

6.1 Organização dos testes

Para uma análise útil do comportamento do algoritmo é necessário implementar uma pequena modificação na geração da solução inicial. Como as soluções iniciais são aleatórias, para efeito de comparação, deve-se alterar o algoritmo que gera a primeira solução de forma que a partida para a mesma escala seja sempre a mesma. Com isso, a variação dos parâmetros pode ser melhor estudada, pois uma solução inicial diferente poderia influir no comportamento do algoritmo, invalidando a análise para a variação do parâmetro em estudo.

Os testes foram organizados da seguinte maneira:

1. Comparação de vizinhanças:

2. Construção de gráficos de *Custo* × *iteração* Construção de gráficos de *Custo* × *iteração* para 4 instâncias distintas do problema e para cada uma das duas vizinhanças desenvolvidas, num total de 8 gráficos.

3. Avaliação do tamanho da lista tabu:

   (a) Para duas instâncias: construção dos gráficos de *Custo* × *iteração* variando a lista tabu em três valores: 2, 10 e 20, num total de 6 gráficos.

4. Efeito de modificações na solução de partida:

   (a) Elaboração de um total de 3 gráficos de *Custo* × *iteração*, considerando três partidas distintas para uma instância.
   (b) Análise de duas soluções diferentes obtidas para uma mesma instância.

5. Efeitos da mudança de alguns parâmetros sobre a solução final. Para uma dada instância, é exibida uma tabela com o número de violações e o custo final da solução encontrada, para cada um dos seguintes casos:

   (a) Utilizando o melhor ajuste encontrado.
   (b) Atribuindo menor penalização às violações de preferências.
   (c) Aumentando a penalização por horas-extra.
   (d) Diminuindo o custo de violação do limite de horas-extra por dia.
   (e) Diminuindo a penalização por interstício.

6. Comparação de algumas soluções geradas com o algoritmo de busca tabu e as soluções manuais encontradas pela diretoria do HC:

   (a) Exibir as respectivas soluções e custos associados para 2 escalas.

O custo de cada penalização utilizada no cálculo do custo total de uma solução está definido como abaixo:

```
[PENALIDADES]
FolgasMes = 500                 \\ Numero minimo de folgas que devem ser
                                \\ fornecidas para cada funcionario
FuncionariosDia = 60            \\ Necessidade de funcionarios em cada dia
MinPreferenciasAtendidas = 400  \\ Preferencias atendidas para cada funcionario
PreferenciasAtendidas = 50      \\ Penalizacao por cada preferencia nao atendida
FolgaFinaisSemana = 180         \\ Folga nos finais de semana
DiasConsecutivos = 500          \\ Maximo de dias consecutivos de trabalho
LimiteHorasExtraDia = 20        \\ Limite maximo de horas extras em cada dia
MinDiasIntersticio = 100        \\ Minimo de dias consecutivos de trabalho
PenalidadeSimples = 10          \\ Penalidades relacionadas ao balanceamento
PenalidadeDupla = 10            \\ de folgas simples e duplas em finais de semana
```

Se não houver menção a nenhuma alteração dos parâmetros acima, todos os custos calculados serão relativos a esse conjunto.

6.2 Comparação das vizinhanças

Simulações: Foram escolhidas as escalas de agosto e setembro do período matutino, fornecidas pelo HC. O comportamento da primeira vizinhança pode ser visto nos gráficos das Figuras 3, 4, 5 e 6. As mesmas instâncias, com mesma partida inicial, porém usando-se a segunda vizinhança, são mostradas nas Figuras 7, 8, 9 e 10.

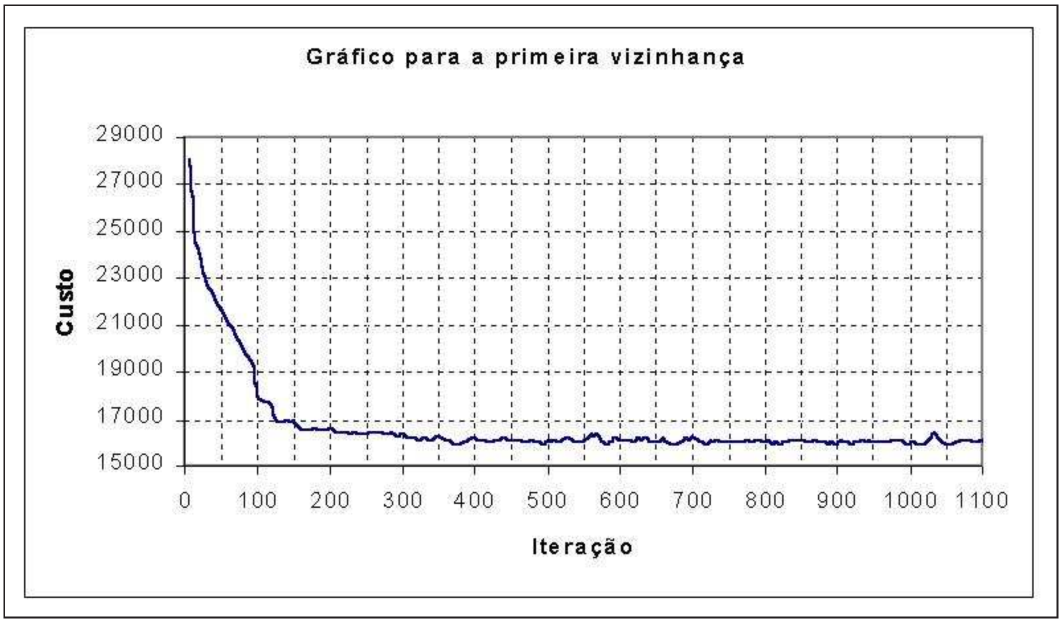


Figura 3: Primeira vizinhança: Execução para a escala de agosto, período da manhã e funcionários auxiliares e técnicos

Análise: Com relação à primeira vizinhança, nota-se que é possível "escapar" do primeiro mínimo local e oscilar em torno de soluções de pior custo, chegando em soluções melhores que a encontrada no primeiro mínimo local. Como será demonstrado adiante, foram obtidas soluções com o mínimo possível de horas extras e sem nenhuma violação para todas as escalas fornecidas pelo HC.

A implementação da segunda vizinhança também mostrou resultados satisfatórios. A partir dos gráficos das Figuras 3 a 10 é possível comparar os resultados obtidos para cada uma das vizinhanças analisadas. A segunda vizinhança apresenta pouca oscilação mas, na maioria dos casos, os melhores custos são obtidos após algumas iterações, i.e., sempre se encontra uma solução de menor custo que àquela dada pelo primeiro mínimo local. A menor oscilação da segunda vizinhança pode ser explicada pela limitação imposta para o conjunto de soluções que o algoritmo pode percorrer. Nessa segunda vizinhança, algumas restrições, tais como o número mínimo de finais de semana, o número mínimo de folgas a serem concedidas e os interstícios, já são automaticamente atendidos. O algoritmo, então, não passa por soluções que violem essas restrições. Desta forma, o espaço de soluções que o algoritmo percorre é mais limitado. Em contrapartida, o espaço de soluções da

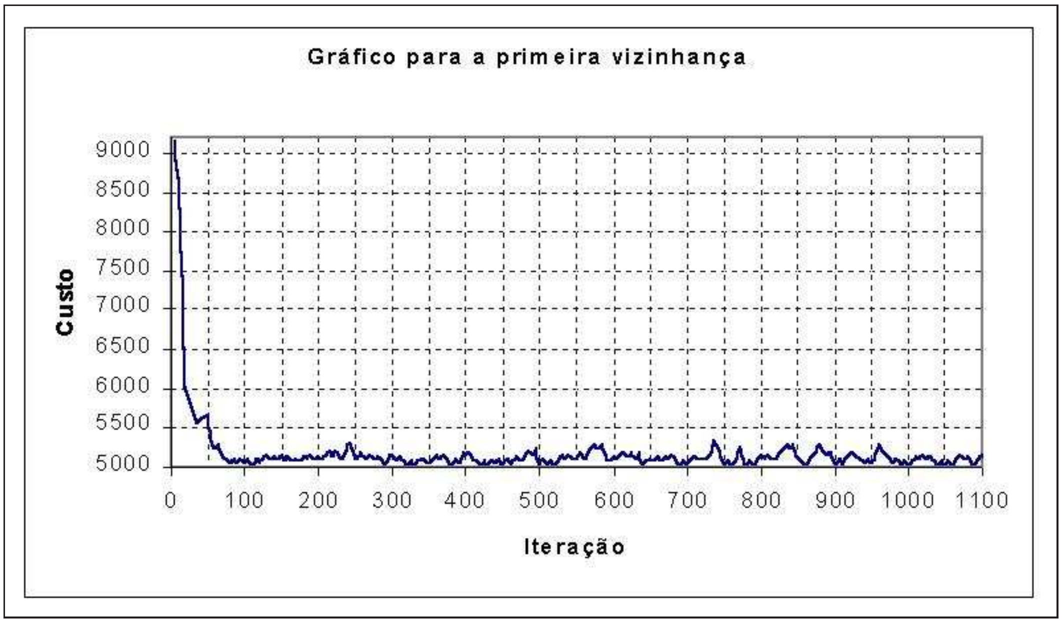


Figura 4: Primeira vizinhança: Execução para a escala de agosto, período da manhã referente aos enfermeiros

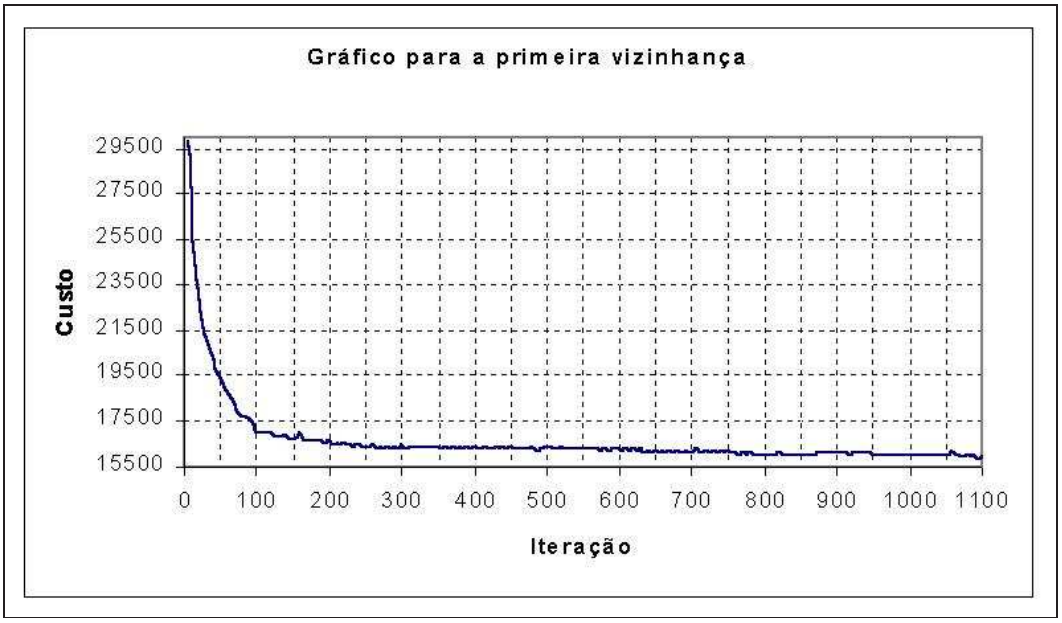


Figura 5: Primeira vizinhança: Execução para a escala de setembro, período da manhã e funcionários auxiliares e técnicos

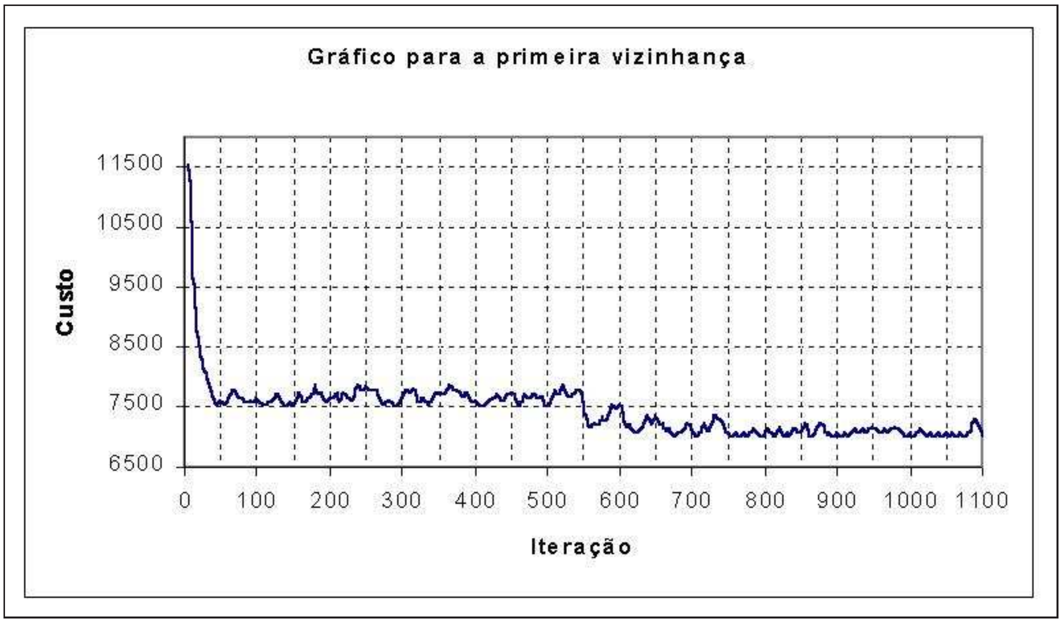


Figura 6: Primeira vizinhança: Execução para a escala de setembro, período da manhã, referente aos enfermeiros

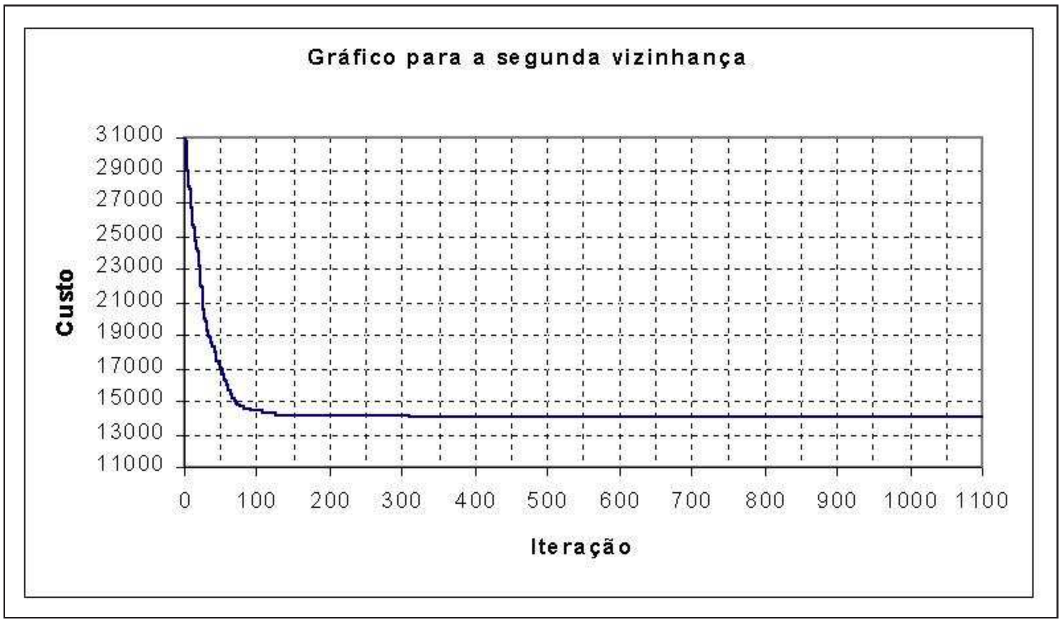


Figura 7: Segunda vizinhança: Execução para a escala de agosto, período da manhã e funcionários auxiliares e técnicos

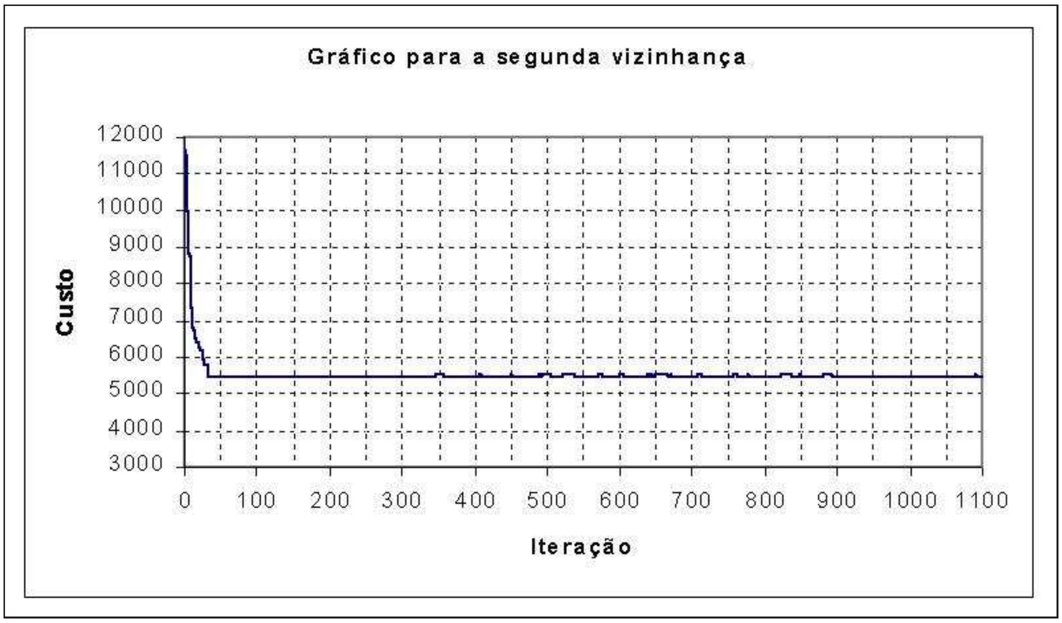


Figura 8: Segunda vizinhança: Execução para a escala de agosto, período da manhã referente aos enfermeiros

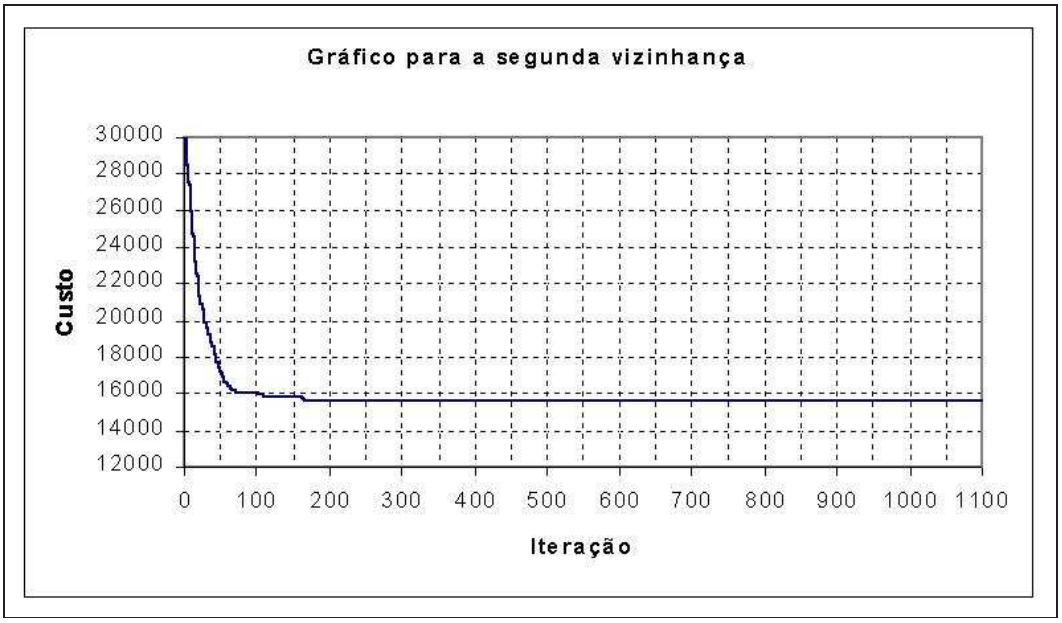


Figura 9: Segunda vizinhança: Execução para a escala de setembro, período da manhã e funcionários auxiliares e técnicos

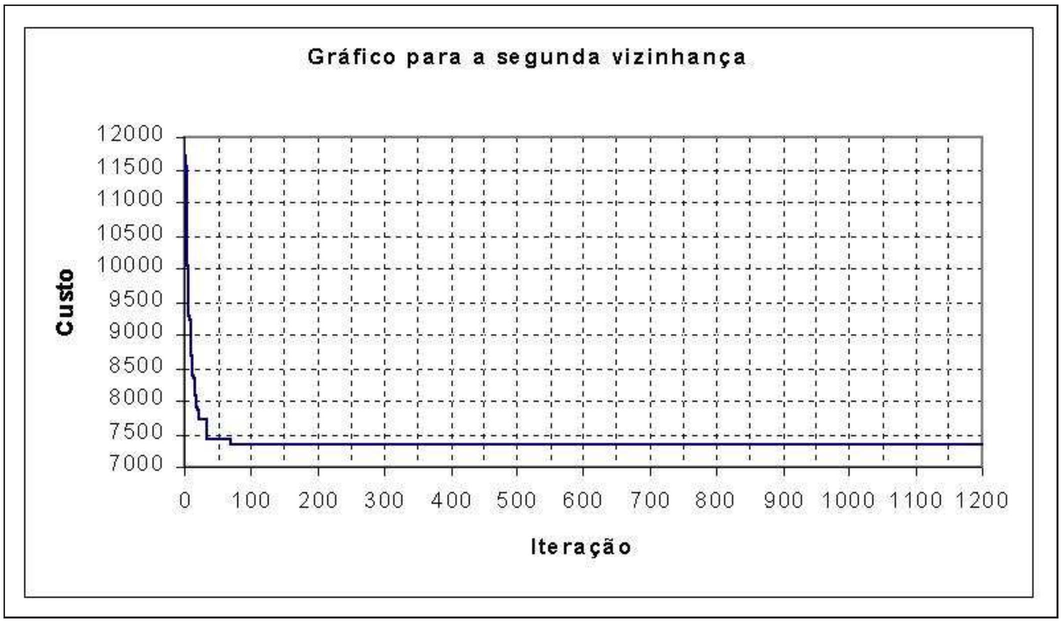


Figura 10: Segunda vizinhança: Execução para a escala de setembro, período da manhã, referente aos enfermeiros

primeira vizinhança é muito mais amplo, pois as soluções obtidas podem violar as restrições mencionadas acima. Esse fato pode justificar a maior oscilação observada.

Essa diferença faz com que a segunda vizinhança, como pode ser observado nos gráficos, atinja uma solução com custo semelhante à obtida com a primeira vizinhança, porém em menor tempo. Ou seja, verifica-se que a segunda vizinhança, na maioria dos casos, dá origem a um algoritmo mais rápido. Também pode ser verificado que os custos finais para ambas as vizinhanças são semelhantes, assim como as respectivas soluções finais.

6.3 Variação da lista tabu

Simulações: Para essa análise foram eleitas as mesmas escalas da seção anterior, porém envolvendo somente auxiliares e técnicos, resultando no total de 6 gráficos. Foi usada a primeira vizinhança pois, segundo os testes anteriores, é a vizinhança que apresenta maior oscilação, devendo ser a mais afetada pela modificação no tamanho da lista tabu. Os gráficos obtidos podem ser visualizados nas Figuras 11, 12 e 13 para a escala de agosto, e nas Figuras 14, 15 e 16 para a escala de setembro. Novamente toma-se o cuidado de ajustar a mesma solução de partida inicial.

Os gráficos mostram a região de oscilação do custo associado à solução após o algoritmo encontrar o primeiro mínimo local. Isso possibilita analisar com mais clareza o efeito do tamanho da lista tabu.

Análise: Observando os gráficos, é possível concluir que o tamanho da lista tabu influencia diretamente o custo da solução obtida. Os gráficos para as duas escalas resultaram em

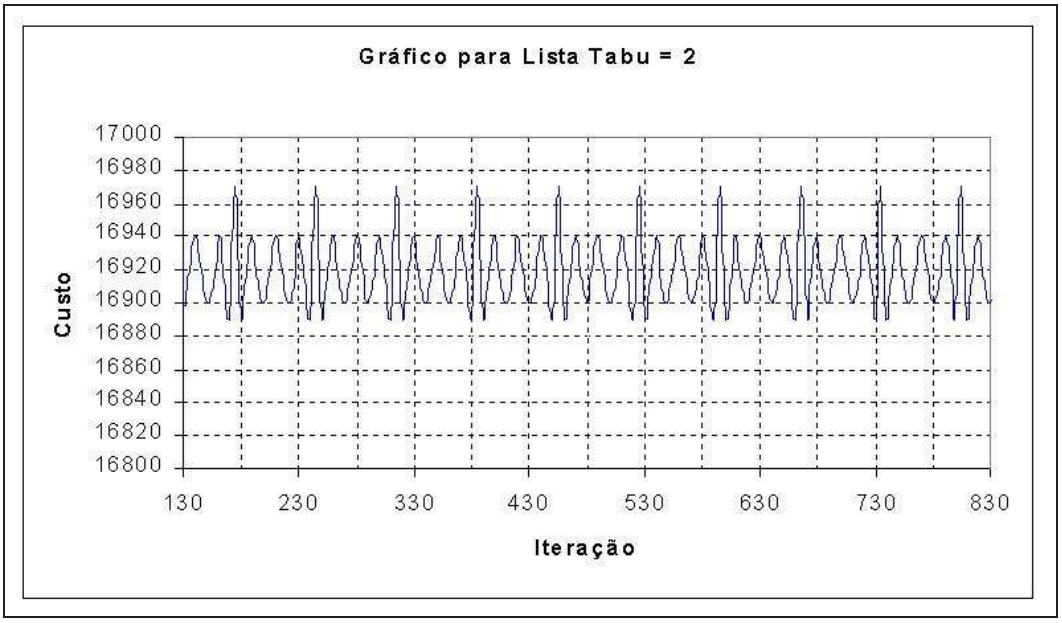


Figura 11: Execução para a escala de agosto com lista tabu de tamanho 2

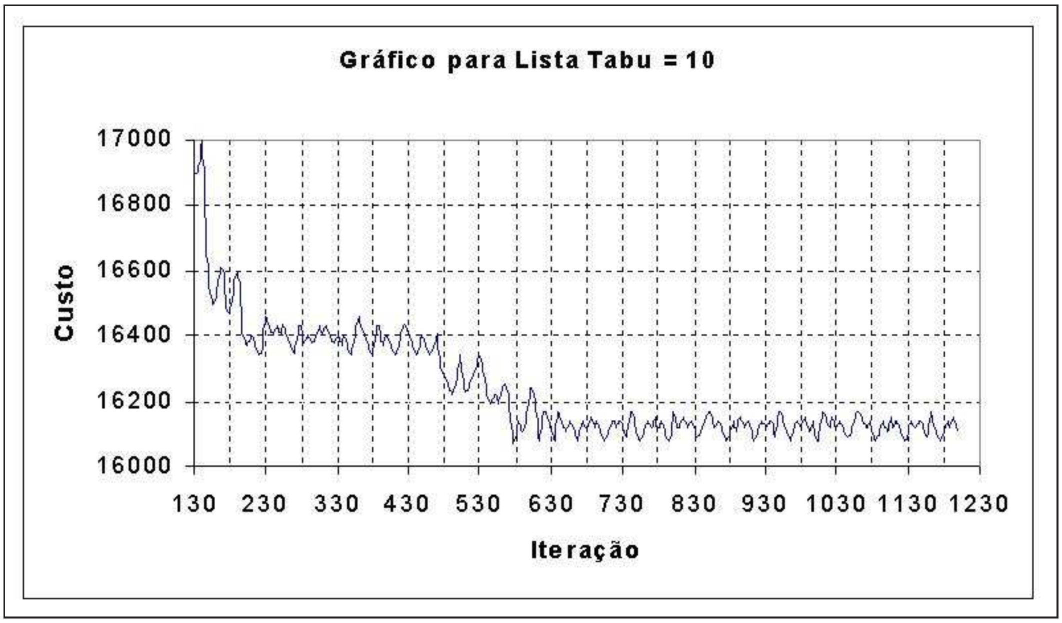


Figura 12: Execução para a escala de agosto com lista tabu de tamanho 10

situações semelhantes. Para uma lista tabu pequena, restringindo apenas 2 movimentos anteriores, o algoritmo praticamente não consegue ir além do primeiro mínimo local en-

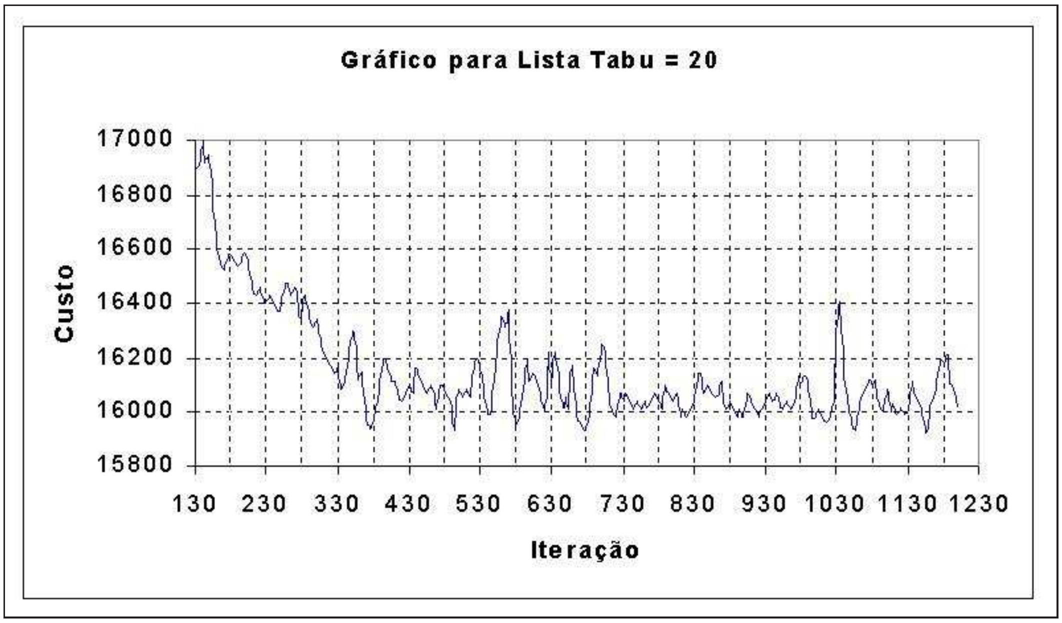


Figura 13: Execução para a escala de agosto com lista tabu de tamanho 20

contrado. O custo oscila entre dois valores até que o máximo de iterações seja atingido. O melhor custo encontrado nesse caso foi de 16890 para a escala de agosto e 17014 para a de setembro. Para esse tamanho de lista tabu pode-se dizer que o algoritmo funciona praticamente como uma busca local, não lançando mão das características da busca tabu que permitem escapar de um mínimo local.

Quando o comprimento da lista tabu é aumentado, como esperado, a oscilação diminui e o algoritmo consegue escapar do primeiro mínimo local atingido. Pode-se verificar esse fato observando as Figuras 12 e 15, quando defini-se uma lista tabu de tamanho 10. Nota-se que com uma lista tabu desse porte atribui-se maior liberdade ao algoritmo que consegue passar por um número maior de soluções com custo mais alto (soluções piores) e então chegar a uma solução de custo menor. Observa-se uma ligeira queda nos custos das soluções obtidas: para a escala de agosto o melhor custo encontrado foi 16080 e para setembro, 15933.

Um outro teste possibilitou aumentar ainda mais a lista tabu. Configurou-se uma lista de tamanho 20. O resultado obtido nas Figuras 13 e 16 revelam, novamente, uma diminuição na oscilação em torno de duas ou três soluções, levando o algoritmo à soluções ainda melhores. Os custos obtidos comprovam essa análise: para a escala de agosto o custo foi de 15920 e para setembro, 15804.

Segundo esses resultados, e outros testes realizados com o tamanho da lista tabu, decidiu-se pela escolha de uma lista tabu de tamanho 20. Nos outros testes reportados nessa seção, o tamanho da lista tabu foi mantido nesse valor.

Uma outra observação interessante é que o tamanho da lista tabu na obtenção da solução de melhor custo também depende da escala analisada. Observa-se que o aumento da lista tabu reduziu o custo final da escala de setembro de forma mais incisiva, quando comparada com a redução do custo final calculado para a escala de agosto. Quando a lista tabu aumentou de 2 para 10, a redução para a escala de setembro foi de 1081 e a de agosto foi de 810.

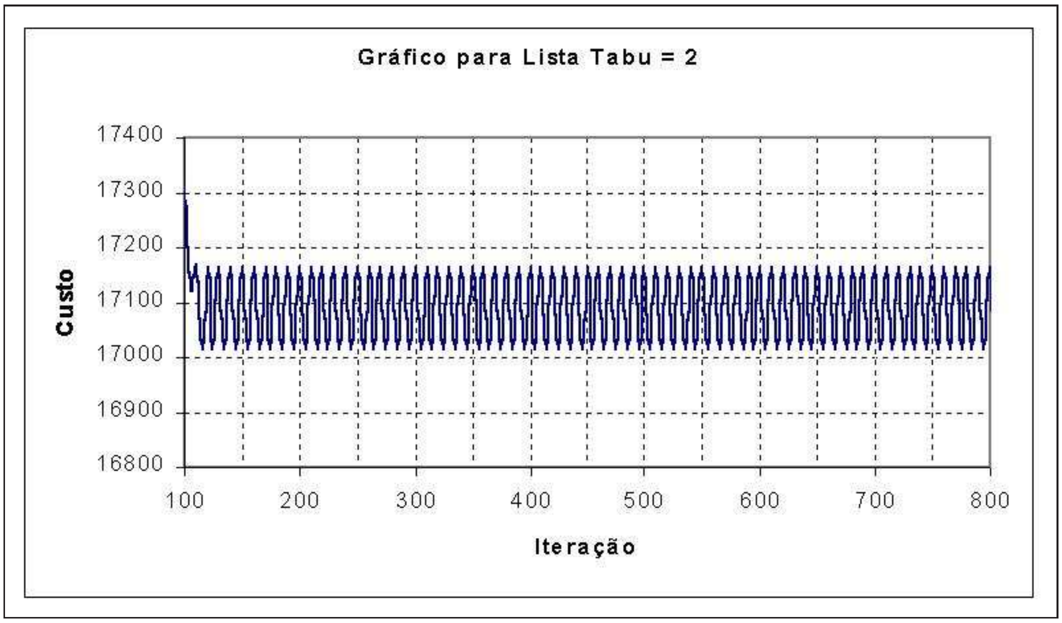


Figura 14: Execução para a escala de setembro com lista tabu de tamanho 2

6.4 Efeito da solução de partida

Simulações: A escala utilizada nesse caso foi a de setembro, período vespertino, referente aos auxiliares e técnicos. Esse teste é importante para verificar a sensibilidade do algoritmo quanto à solução de partida inicial, uma vez que a partida é pseudo-aleatória.

Os gráficos para três partidas distintas são exibidos nas Figuras 17, 18 e 19. Novamente, optou-se por exibir os gráficos após o encontro do primeiro mínimo local, de modo a facilitar sua visualização. As Tabelas 2, 3 e 4 exibem as soluções encontradas para essas partidas. Nessas soluções, a letra T representa um dia de trabalho, F representa uma folga e I um dia de impedimento. Os custos associados aparecem na Tabela 5.

Análise: A primeira observação é que a partida inicial tem influência considerável sobre a solução final obtida. Apesar dos custos associados serem diferentes, as soluções finais obtidas são consideradas muito boas, pois não apresentam violação de nenhuma das restrições estipuladas. Verifica-se que, para cada uma das partidas, foi alcançado o número mínimo de horas-extra possível. A diferença entre os custos obtidos é, portanto, atribuída àquelas restrições leves, como o balanceamento no número de finais de semana concedidos como folga e o limite de horas-extra atribuídos por dia. Nota-se que essas restrições possuem pesos relativamente baixos, e que levam a custos próximos para cada uma das três partidas analisadas.

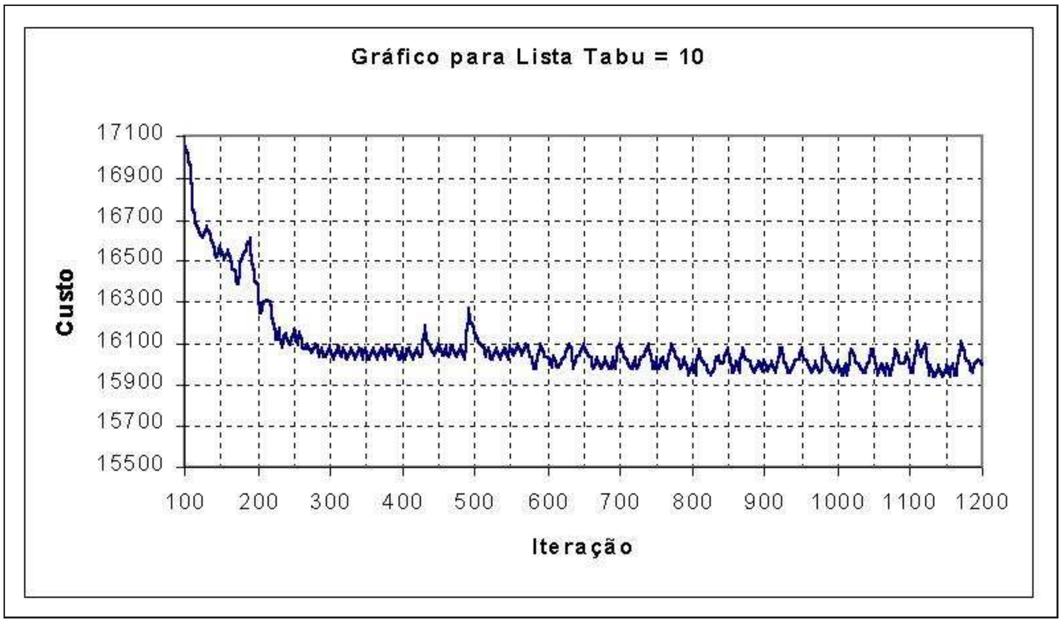


Figura 15: Execução para a escala de setembro com lista tabu de tamanho 10

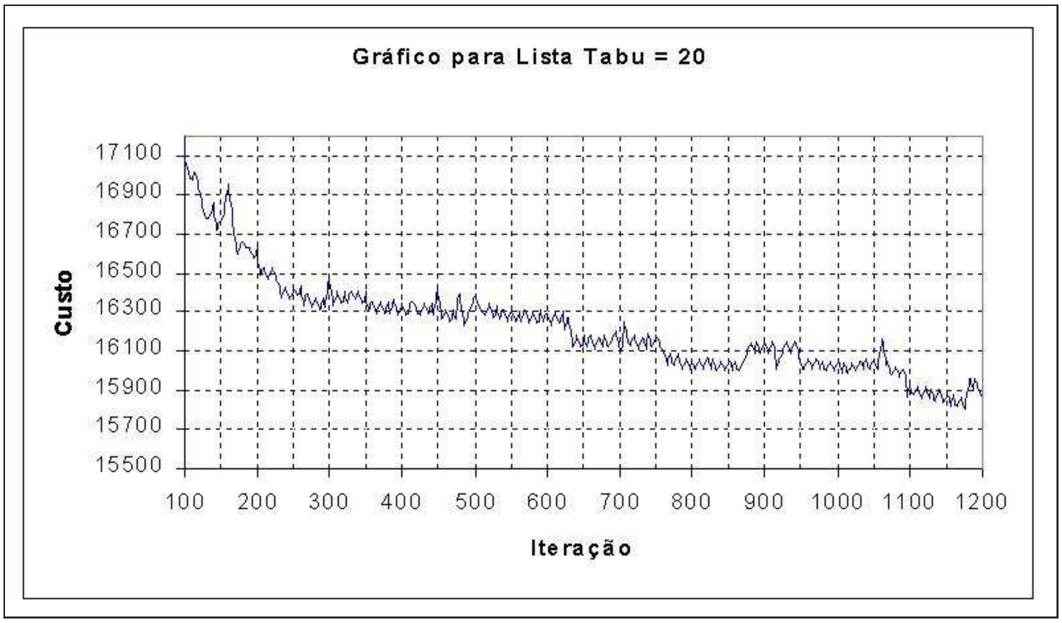
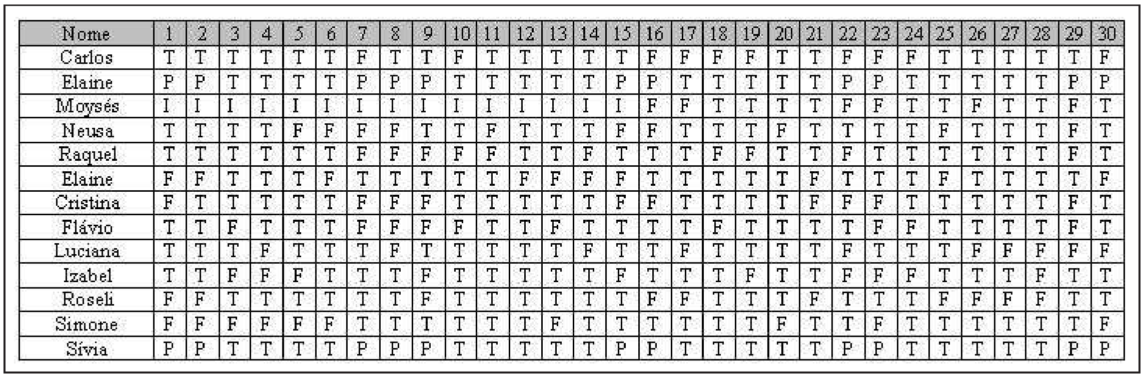


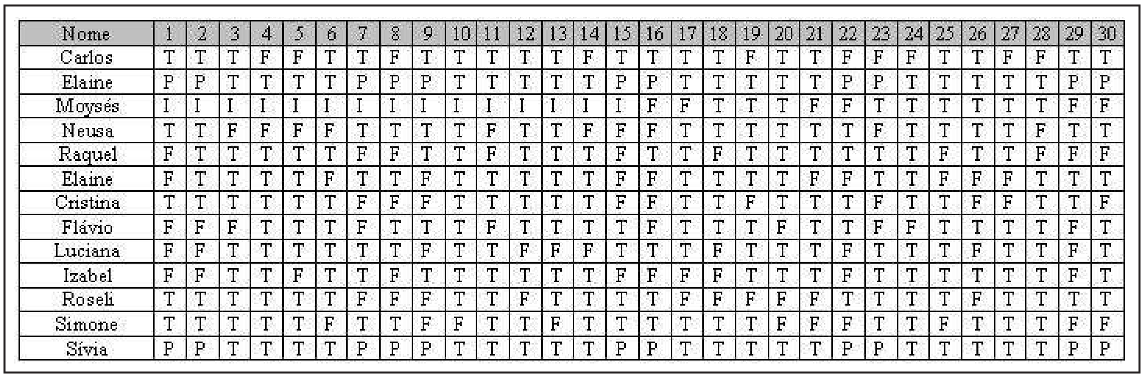
Figura 16: Execução para a escala de setembro com lista tabu de tamanho 20

Os gráficos das Figuras 17, 18 e 19 demonstram que a solução usada como partida inicial define o "caminho" do algoritmo. Ou seja, a partida inicial é importante pois define as



Nome	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Carlos	T	T	T	T	T	T	F	T	T	F	T	T	T	T	T	F	F	F	F	T	T	F	F	F	T	T	T	T	T	F
Elaine	P	P	T	T	T	T	P	P	P	T	T	T	T	T	P	P	T	T	T	T	T	P	P	T	T	T	T	T	P	P
Moysés	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	F	F	T	T	T	T	F	F	T	T	F	T	T	F	T
Neusa	T	T	T	T	F	F	F	F	T	T	F	T	T	T	F	F	T	T	T	F	T	T	T	T	F	T	T	T	F	T
Raquel	T	T	T	T	T	T	F	F	F	F	F	T	T	F	T	T	T	F	F	T	T	F	T	T	T	T	T	T	F	T
Elaine	F	F	T	T	T	F	T	T	T	T	T	F	F	F	F	T	T	T	T	T	F	T	T	T	F	T	T	T	T	F
Cristina	F	T	T	T	T	T	F	F	F	T	T	T	T	T	F	F	T	T	T	T	F	F	F	T	T	T	T	T	F	T
Flávio	T	T	F	T	T	T	F	F	F	F	T	T	F	T	T	T	T	F	T	T	T	T	F	F	T	T	T	T	F	T
Luciana	T	T	T	F	T	T	T	F	T	T	T	T	T	F	T	T	F	T	T	T	T	F	T	T	T	F	F	F	F	F
Izabel	T	T	F	F	F	T	T	T	F	T	T	T	T	T	F	T	T	T	F	T	T	F	F	F	T	T	T	F	T	T
Roseli	F	F	T	T	T	T	T	T	F	T	T	T	T	T	T	F	F	T	T	T	F	T	T	T	F	F	F	F	T	T
Simone	F	F	F	F	F	F	T	T	T	T	T	T	F	T	T	T	T	T	T	F	T	T	F	T	T	T	T	T	T	F
Sívia	P	P	T	T	T	T	P	P	P	T	T	T	T	T	P	P	T	T	T	T	T	P	P	T	T	T	T	T	P	P

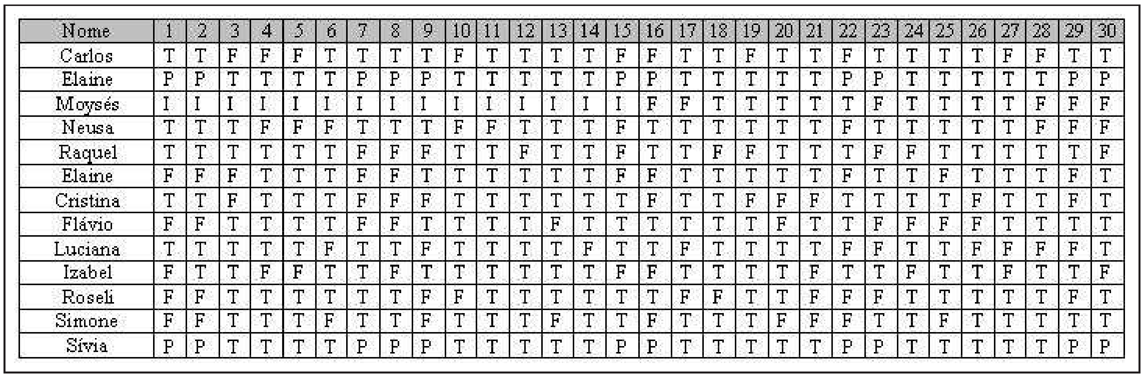
Tabela 2: Comportamento para a primeira partida pseudo-aleatória



Nome	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Carlos	T	T	T	F	F	T	T	F	T	T	T	T	T	F	T	T	T	T	F	T	T	F	F	F	T	T	F	F	T	T
Elaine	P	P	T	T	T	T	P	P	P	T	T	T	T	T	P	P	T	T	T	T	T	P	P	T	T	T	T	T	P	P
Moysés	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	F	F	T	T	T	F	F	T	T	T	T	T	T	F	F
Neusa	T	T	F	F	F	F	T	T	T	T	F	T	T	F	F	F	T	T	T	T	T	T	F	T	T	T	T	F	T	T
Raquel	F	T	T	T	T	T	F	F	T	T	F	T	T	T	F	T	T	F	T	T	T	T	T	T	F	T	T	F	F	F
Elaine	F	T	T	T	T	F	T	T	F	T	T	T	T	T	F	F	T	T	T	T	F	F	T	T	F	F	F	T	T	T
Cristina	T	T	T	T	T	T	F	F	F	T	T	T	T	T	F	F	T	T	F	T	T	T	F	T	T	F	F	T	T	F
Flávio	F	F	F	T	T	T	F	T	T	T	F	T	T	T	T	F	T	T	T	F	T	T	F	F	T	T	T	T	F	T
Luciana	F	F	T	T	T	T	T	T	F	T	T	F	F	F	T	T	T	F	T	T	T	F	T	T	T	F	T	T	F	T
Izabel	F	F	T	T	F	T	T	F	T	T	T	T	T	T	F	F	F	F	T	T	T	F	T	T	T	T	T	T	F	T
Roseli	T	T	T	T	T	T	F	F	F	T	T	F	T	T	T	T	F	F	F	F	F	T	T	T	T	F	T	T	T	T
Simone	T	T	T	T	T	F	T	T	F	F	T	T	F	T	T	T	T	T	T	F	F	F	T	T	F	T	T	T	F	F
Sívia	P	P	T	T	T	T	P	P	P	T	T	T	T	T	P	P	T	T	T	T	T	P	P	T	T	T	T	T	P	P

Tabela 3: Comportamento para a segunda partida pseudo-aleatória

soluções através das quais o algoritmo irá passar até o término de sua execução. Observando



Nome	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Carlos	T	T	F	F	F	T	T	T	T	F	T	T	T	T	F	F	T	T	F	T	T	F	T	T	T	T	F	F	T	T
Elaine	P	P	T	T	T	T	P	P	P	T	T	T	T	T	P	P	T	T	T	T	T	P	P	T	T	T	T	T	P	P
Moysés	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	F	F	T	T	T	T	T	F	T	T	T	T	F	F	F
Neusa	T	T	T	F	F	F	T	T	T	F	F	T	T	T	F	T	T	T	T	T	T	F	T	T	T	T	T	F	F	F
Raquel	T	T	T	T	T	T	F	F	F	T	T	F	T	T	F	T	T	F	F	T	T	T	F	F	T	T	T	T	T	F
Elaine	F	F	F	T	T	T	F	F	T	T	T	T	T	T	F	F	T	T	T	T	T	F	T	T	F	T	T	T	F	T
Cristina	T	T	F	T	T	T	F	F	F	T	T	T	T	T	T	F	T	T	F	F	F	T	T	T	T	F	T	T	F	T
Flávio	F	F	T	T	T	T	F	F	T	T	T	T	F	T	T	T	T	T	T	F	T	T	F	F	F	F	T	T	T	T
Luciana	T	T	T	T	T	F	T	T	F	T	T	T	T	F	T	T	F	T	T	T	T	F	F	T	T	F	F	F	F	T
Izabel	F	T	T	F	F	T	T	F	T	T	T	T	T	T	F	F	T	T	T	T	F	T	T	F	T	T	F	T	T	F
Roseli	F	F	T	T	T	T	T	T	F	F	T	T	T	T	T	T	F	F	T	T	F	F	F	T	T	T	T	T	F	T
Simone	F	F	T	T	T	F	T	T	F	T	T	T	F	T	T	F	T	T	T	F	F	F	T	T	F	T	T	T	T	T
Sívia	P	P	T	T	T	T	P	P	P	T	T	T	T	T	P	P	T	T	T	T	T	P	P	T	T	T	T	T	P	P

Tabela 4: Comportamento para a terceira partida pseudo-aleatória

	Primeira Partida	**Segunda Partida**	**Terceira Partida**
Custo	6809	6769	5905

Tabela 5: Custos assodiados às três partidas pseudo-aleatórias

o primeiro ponto de cada gráfico, que define o primeiro mínimo local encontrado, verifica-se que o custo associado a esse ponto também depende da partida inicial. Pode-se dizer que a definição do primeiro mínimo local é crucial para a determinação do "caminho" que o algoritmo segue até a solução de melhor custo.

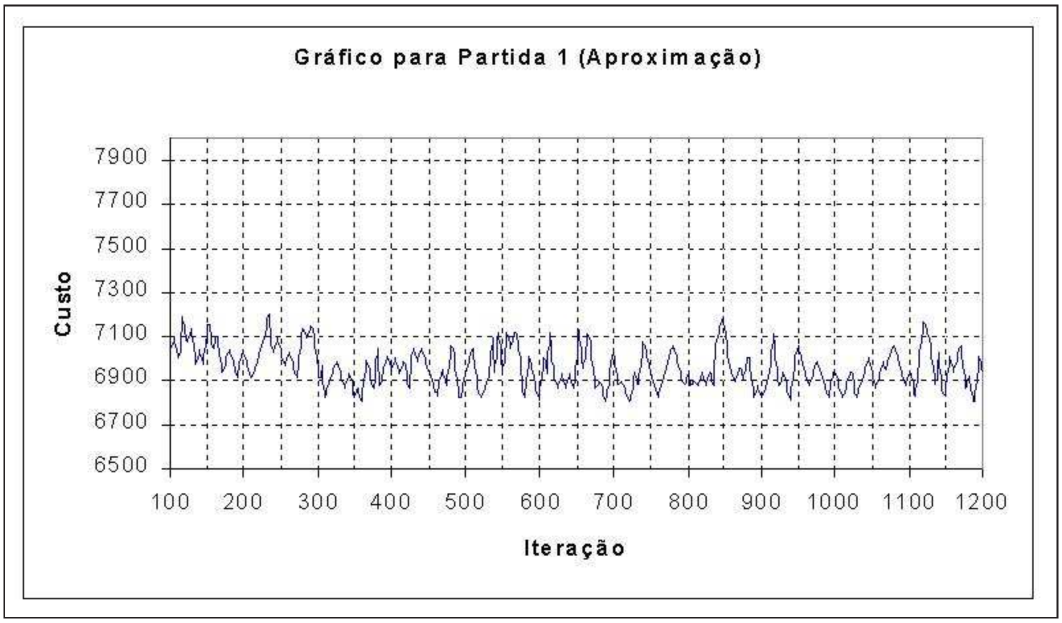


Figura 17: Comportamento para a primeira partida pseudo-aleatória

6.5 Modificando os parâmetros de custo

Simulações: Como já discutido previamente na seção 4.2, o arquivo de define as penalizações é o mesmo exibido no início dessa seção. Os valores atribuídos nesse arquivo base correspondem ao melhor ajuste encontrado à luz dos resultados obtidos até então. A esse primeiro ajuste atribuiu-se o nome de *ajuste zero*. As modificações propostas refletirão alterações em uma ou mais penalizações definidas no arquivo acima, permanecendo as outras penalizações inalteradas.

O arquivo com o melhor ajuste será tomado como base para avaliação da modificação dos parâmetros escolhidos para análise. No entanto, nessa seção, não foram modificados todos os parâmetros, pois a análise tornaria-se muito complexa a ponto de dificultar a obtenção

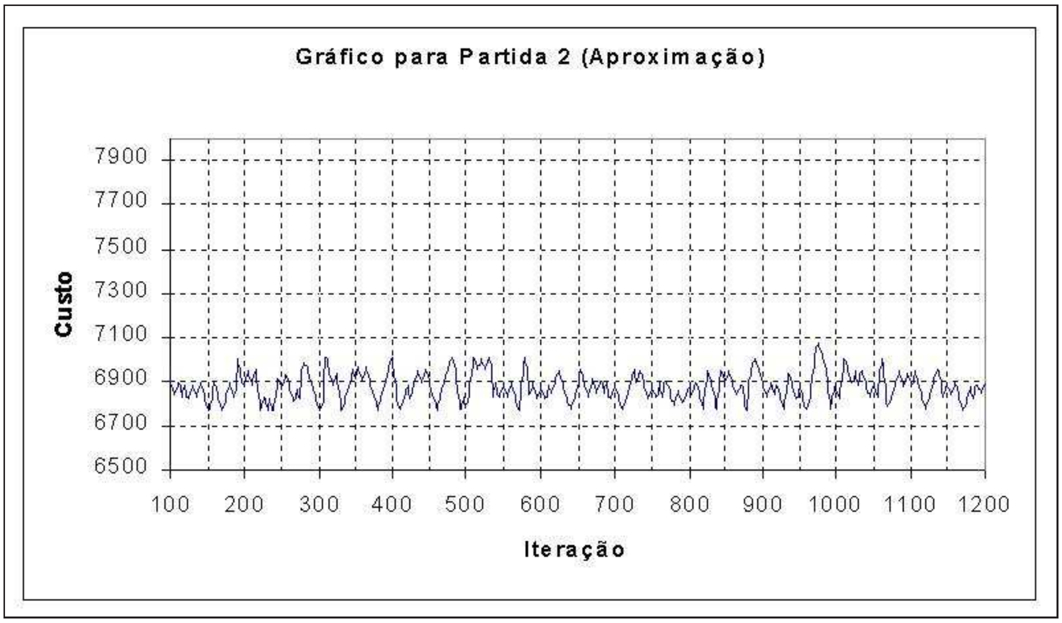


Figura 18: Comportamento para a segunda partida pseudo-aleatória

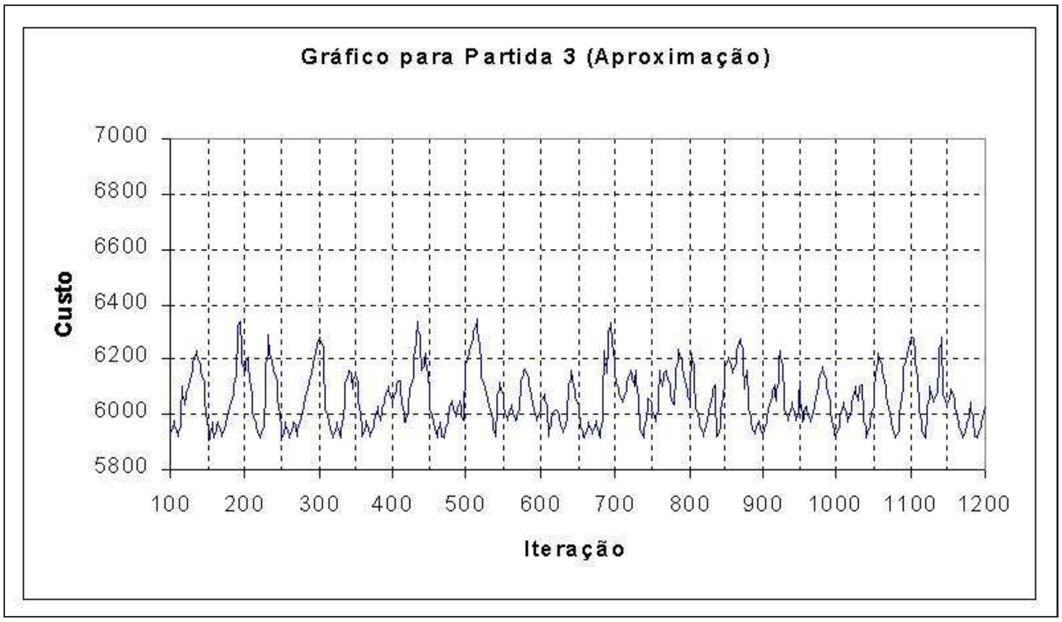


Figura 19: Comportamento para a terceira partida pseudo-aleatória

de conclusões sólidas. Os ajustes testados foram os seguintes:

Ajuste 1: Menor peso para violação das preferências: Foram alteradas as seguintes linhas do arquivo:

```
MinPreferenciasAtendidas = 50 % <---
PreferenciasAtendidas = 10 % <---
```

Ajuste 2: Maior penalização por horas-extra: A linha que define esse custo foi alterada para:

```
FuncionariosDia = 500   % <---
```

Ajuste 3: Menor penalização por violar o limite de horas-extra no dia: O custo de violar o limite de horas-extra por dia já é pequeno. Para analisar o efeito de um custo ainda menor, define-se:

```
LimiteHorasExtraDia = 5  % <---
```

Ajuste 4: Diminuindo a penalização por interstício: A ausência de um determinado interstício entre folgas também é definida como uma característica indesejável na solução, sendo, portanto, uma restrição leve. No entanto o melhor ajuste corresponde a um valor relativamente alto de penalização nesse caso. Reduzindo esse valor para aquele definido abaixo é possível avaliar melhor esse parâmetro:

```
MinDiasIntersticio = 50 % <---
```

Utilizando os novos arquivos gerados a partir de cada uma das modificações acima (5 arquivos ao todo), o algoritmo foi executado sobre cada uma delas, produzindo os resultados mostrados na Tabela 6.

Análise: Da tabela obtida na simulação, observa-se que os parâmetros foram ajustados de maneira correta. Todas as modificações causaram alguma violação das restrições estabelecidas ou provocaram um aumento no número de horas-extra. As violações obtidas para cada caso confirmam o esperado.

Quando a penalização por violação de preferências diminuiu (ajuste 1), o resultado obtido revelou 5 violações de preferências.

No ajuste 2, o aumento da penalização por horas-extra levou à redução do número de horas-extra além do mínimo. Para que isso fosse possível, no entanto, um dos funcionários recebeu uma folga a menos. Ainda houve, nesse caso, um funcionário apresentando um interstício não atendido. Observando a solução gerada (não exibida aqui) constata-se que o interstício não atendido causa a diminuição de uma hora-extra sendo, portanto, mantido na solução final devido à alta penalização atribuída às violações de horas-extra. O elevado custo obtido para o ajuste 2, em detrimento dos demais, pode ser justificado pela alta penalização imposta à violação de restrição de folga de um funcionário no arquivo base, utilizado no ajuste zero. Mesmo assim, devido a alta penalização por horas-extra, o algoritmo acabou por suprimir uma folga de um funcionário para gerar um hora-extra a menos na solução final.

Ajuste	**Num. violações**	**Custo**	**Caract. das violações**	**Horas-extra**[a]
0	0	5763	Não houve violações	39
1	5	5590	Não foram atendidas um total de 5 preferências. Dois funcionários tiveram apenas uma preferência atendida e outro teve apenas 2 das três necessárias.	39
2	2	23852	Um funcionário obteve uma folga a menos das obrigatórias. Outro funcionário apresentou 1 interstício não atendido.	38
3	0	5613	Não houve violações.	40
4	1	5845	Um funcionário apresentou 1 interstício não atendido.	39

---

[a] O número mínimo considerado é de 39 horas extras

Tabela 6: Modificando os parâmetros de custo

O ajuste 3 foi o único que não apresentou violação de restrições, excetuando-se o ajuste zero. No entanto, analisando a solução obtida para esse caso verifica-se uma distribuição irregular das horas-extra por dia. Essa distribuição pode ser melhor visualizada no esquema abaixo:

```
                             +
=**====*===*==*===**=====*====
-  ---- --- -- ---  ----- ---
-  ---- --  -- -     -  -  --
            -        -      -
            -        -
```

Nesse esquema cada coluna corresponde a um dia e ainda:

- `-` – cada traço indica uma hora extra necessária naquele dia.

`*` – esse sinal indica que há número suficiente de funcionários naquele dia.

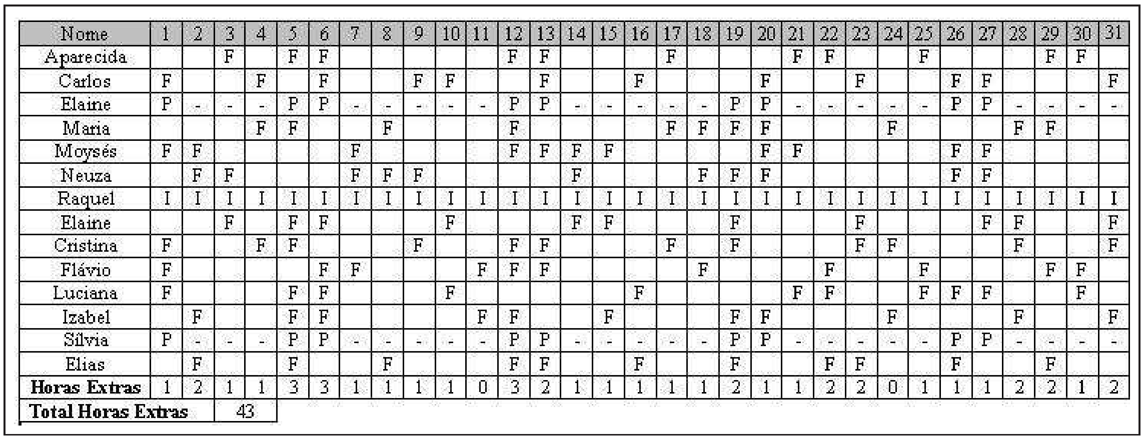
\+ – indica que há excesso de um funcionário.

Sabendo que o limite de horas-extra por dia é 2, verifica-se que existem dias com até 4 horas-extra. Isso pode ser atribuído diretamente à diminuição da penalização por limite de horas-extra por dia.

O ajuste 4 propõe a diminuição da penalização por interstício. Verifica-se a presença de um interstício não atendido na solução final obtida sem, no entanto, alterar o número de horas-extra, que permanece mínimo.

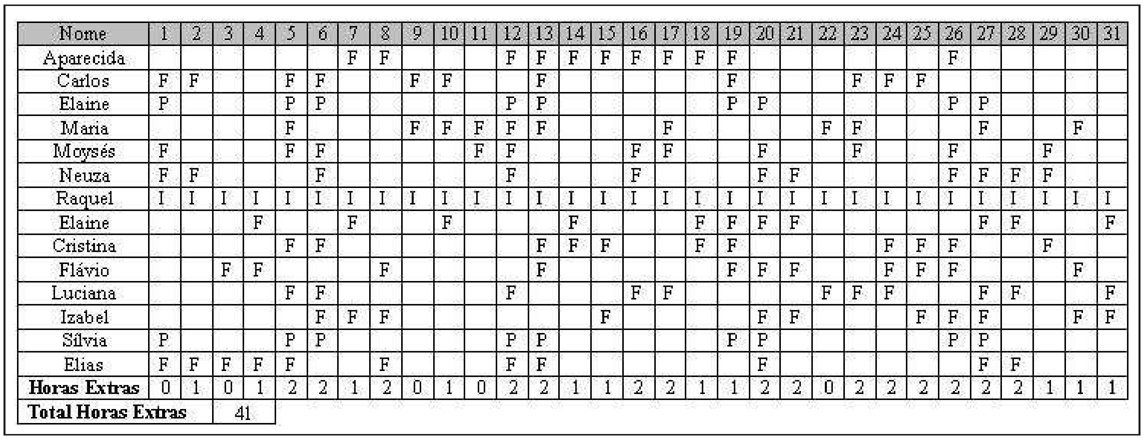
6.6 Comparação das soluções geradas pelo algoritmo e as soluções geradas manualmente

Simulações: Nessa análise utilizaram-se como instâncias a escala do mês de maio de 2001, período vespertino referente aos auxiliares e técnicos, e a escala de agosto de 2001, também para o período vespertino, mas considerando os enfermeiros. Escalas manuais foram obtidas junto ao HC [2]. As soluções exibidas a seguir mostram os resultados gerados manualmente e os resultados obtidos pelo algoritmo de busca tabu.



Nome	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Aparecida			F		F	F						F	F				F				F	F			F				F	F	
Carlos	F			F		F			F	F			F			F				F			F			F	F				F
Elaine	P	-	-	-	P	P	-	-	-	-	-	P	P	-	-	-	-	-	P	P	-	-	-	-	-	P	P	-	-	-	-
Maria				F	F			F				F					F	F	F	F				F				F	F		
Moysés	F	F					F					F	F	F	F					F	F					F	F				
Neuza		F	F				F	F	F					F				F	F	F						F	F				
Raquel	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
Elaine			F		F	F				F				F	F				F				F				F	F			F
Cristina	F			F	F				F			F	F				F		F				F	F				F			F
Flávio	F					F	F				F	F	F					F				F			F				F	F	
Luciana	F				F	F				F						F					F	F			F	F	F			F	
Izabel		F			F	F					F	F			F				F	F				F				F			F
Sílvia	P	-	-	-	P	P	-	-	-	-	-	P	P	-	-	-	-	-	P	P	-	-	-	-	-	P	P	-	-	-	-
Elias		F			F			F				F	F			F			F			F	F			F			F		
Horas Extras	1	2	1	1	3	3	1	1	1	1	0	3	2	1	1	1	1	1	2	1	1	2	2	0	1	1	1	2	2	1	2
Total Horas Extras			43																												

Tabela 7: Mês de maio, solução manual



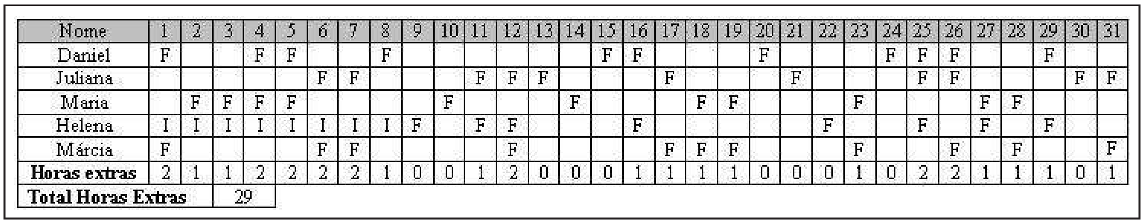
Nome	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Aparecida							F	F				F	F	F	F	F	F	F	F							F					
Carlos	F	F			F	F			F	F			F						F				F	F	F						
Elaine	P				P	P						P	P						P	P						P	P				
Maria					F				F	F	F	F	F				F					F	F				F			F	
Moysés	F				F	F					F	F				F	F			F			F			F			F		
Neuza	F	F				F						F				F				F	F					F	F	F	F		
Raquel	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
Elaine				F			F			F				F				F	F	F	F						F	F			F
Cristina					F	F							F	F	F			F	F					F	F	F			F		
Flávio			F	F				F					F						F	F	F			F	F	F				F	
Luciana					F	F						F				F	F					F	F	F			F	F			F
Izabel						F	F	F							F					F	F				F	F	F			F	F
Sílvia	P				P	P						P	P						P	P						P	P				
Elias	F	F	F	F	F			F				F	F							F							F	F			
Horas Extras	0	1	0	1	2	2	1	2	0	1	0	2	2	1	1	2	2	1	1	2	2	0	2	2	2	2	2	2	1	1	1
Total Horas Extras			41																												

Tabela 8: Mês de maio, solução automática

1. Para o mês de maio, obteve-se:
    (a) Solução manual: na Tabela 7.
    (b) Solução gerada com o algoritmo: na Tabela 8.

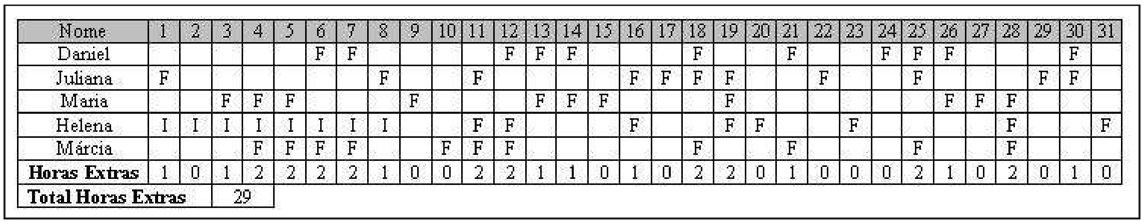
2. Para o mês de agosto:
    (a) Solução manual: na Tabela 9.
    (b) Solução gerada com o algoritmo: na Tabela 10.

A Tabela 11 mostra os custos associados.



Nome	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Daniel	F			F	F			F							F	F				F				F	F	F			F		
Juliana						F	F				F	F	F				F				F				F	F				F	F
Maria		F	F	F	F					F				F				F	F				F				F	F			
Helena	I	I	I	I	I	I	I	I	F		F	F				F						F			F		F		F		
Márcia	F					F	F					F					F	F	F				F			F		F			F
Horas extras	2	1	1	2	2	2	2	1	0	0	1	2	0	0	0	1	1	1	1	0	0	0	1	0	2	2	1	1	1	0	1
Total Horas Extras	29																														

Tabela 9: Mês de agosto, solução manual



Nome	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Daniel						F	F					F	F	F				F			F			F	F	F				F	
Juliana	F							F			F					F	F	F	F			F			F				F	F	
Maria			F	F	F				F				F	F	F				F							F	F	F			
Helena	I	I	I	I	I	I	I	I			F	F				F			F	F			F					F			F
Márcia				F	F	F	F			F	F	F						F			F				F			F			
Horas Extras	1	0	1	2	2	2	2	1	0	0	2	2	1	1	0	1	0	2	2	0	1	0	0	0	2	1	0	2	0	1	0
Total Horas Extras	29																														

Tabela 10: Mês de agosto, solução automática

Mês	**Solução Manual**		**Solução com a Busca tabu**	
	Num. violações	Num. horas-extra	Num. violações	Num. horas-extra
Maio	7	43	0	41
agosto	5	29	0	29

Tabela 11: Custos de soluções manuais e custos de soluções automáticas

Análise: Observando-se as soluções geradas manualmente e as soluções criadas pelo algoritmo, verifica-se de antemão que a distribuição dos dias de trabalho e dias de folga são diferentes. Utilizando-se o programa que gera relatórios criou-se uma tabela que possibilita comparar o número de violações encontrados e o número de horas-extra resultante.

A Tabela 11 mostra que o número de violações das soluções manuais é bem maior que o das soluções geradas com o algoritmo. Observando o relatório gerado para as soluções manuais, verifica-se que todas as violações são relativas a interstícios não atendidos, tanto para o mês de maio quanto para o mês de agosto. Provavelmente, como essa restrição é considerada leve, a diretoria do HC preferiu adotar a solução exibida aqui, ao invés de retrabalhar a escala para suprimir o número de interstícios não atendidos. Nesse ponto, a solução obtida pelo algoritmo de busca tabu, de forma muito mais rápida que a manual (algo entre 10 a 20 segundos), corrige a presença demasiada de interstícios não atendidos, levando a uma solução livre de violações para as duas escalas em análise.

Quanto ao número de horas-extra atribuído a cada escala, percebe-se que a diretoria do HC procura reduzir ao máximo o número de horas-extra para evitar custos muito elevados na operação das escalas. Para o mês de agosto o número de horas-extra obtido pela solução manual e pelo algoritmo foi o menor possível (29 horas-extra). Entretanto, no mês de maio o número de horas-extra exigidos pela solução manual é 43, enquanto que o algoritmo de busca tabu conseguiu obter o mínimo de horas-extra possível (41 horas-extra). Nesse último caso, o ganho para o hospital é evidente. Analisando o conjunto de enfermarias como um todo, se existem ao todo 30 enfermarias com pelo menos 2 turnos passíveis de otimização em cada uma delas (manhã e tarde) e, além disso, em cada turno há uma escala referente aos enfermeiros e outra referente aos funcionários auxiliares e técnicos, calcula-se um ganho de 8 horas-extra em cada enfermaria. No total, tem-se uma redução significativa do custo com recursos humanos na operação das escalas de serviço do hospital.

7 Comparação entre os Algoritmos e Implantação no HC

Essa seção compara os dois métodos de solução implementados, além de comentar brevemente sobre a implantação do sistema no HC.

7.1 Comparativo entre as implementações evolutiva e tabu

Nessa seção, são comparadas as soluções geradas pelos algoritmos de busca tabu contra as soluções geradas pelos algoritmos genéticos, para as mesmas instâncias do problema.

Além do sistema de geração de soluções também foi desenvolvido um sistema que avalia as soluções propostas pelos algoritmos. Esse sistema tem como entrada o arquivo padronizado utilizado pelos algoritmos e pela interface. Se esse arquivo possuir uma solução para o problema, o analisador de solução é acionado e a solução pode ser analisada para posterior comparação com outras soluções.

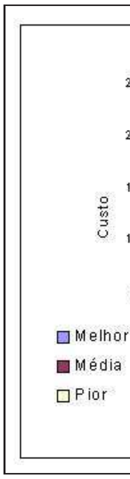
A saída deste analisador é um relatório que apresenta o conjunto de restrições violadas, destacando quais são e onde estão localizadas na matriz de solução. O total de horas-extra utilizados pela solução também é obtido. Este resultado é apresentado através de um gráfico que mostra se há falta, igualdade ou excesso de funcionários de acordo com a necessidade daquele dia. Essa saída gerada pelo analisador também possibilita comparar a solução gerada com a solução que exigiria o menor número de horas-extra. O cálculo do menor número de horas-extra é baseado na necessidade de funcionários no mês, no número de funcionários disponíveis, nos impedimentos e nas folgas que devem ser concedidas.

Para efeito de comparação entre os dois algoritmos, utiliza-se uma função objetivo comum para avaliar as escalas produzidas pelas duas heurísticas. Ao executar, no entanto, essas heurísticas não operavam necessariamente com uma mesma função objetivo. Os resultados serão analisados através do gráfico mostrado na Figura 21. Esse gráfico ilustra um conjunto de execuções de ambos os algoritmos para todas as escalas fornecidas pelo

HC, de forma a contrapor as soluções de mínimo custo, de máximo custo e a solução de custo médio. A média se destaca pois, estatisticamente, representa o custo que deve ser obtido pelos algoritmos em uma execução genérica, considerando sempre o mesmo ajuste de parâmetros. A Tabela 20 mostra os dados a partir dos quais o gráfico da Figura 21 foi construído.

Essas figuras evidenciam que os custos das soluções encontradas pelo algoritmo de busca tabu e pelo algoritmo genético são similares, embora em todos os casos o algoritmo genético gerasse escalas com custos menores. Quanto ao tempo de execução, a busca tabu apresentou leve vantagem.

Através de uma análise dos relatórios gerados a partir dessas soluções é possível verificar ainda que todas as melhores soluções obtidas não contêm nenhuma violação. Além disso, a maioria das soluções encontradas exibe o mínimo de horas-extra possível. Portanto, a utilização de duas heurísticas, conceitualmente bastante diferentes — como é o caso dos algoritmos genéticos e de busca tabu — conduziu à soluções de custos semelhantes.


Custo
Melhor
Média
Pior

Configuração	Ag 20 Ma Aux
Busca Tabu	4 4 5
Algoritmo Genético	4 4 5

F

7.2 Implantação da ferramenta computacional no Hospital das Clínicas

Desde julho de 2001 foram realizadas várias reuniões no HC, encontros esses destinados à troca de idéias, sugestões e informações sobre o funcionamento desejado dos algoritmos.

Também foram realizados treinamentos com os usuários do HC para repassar-lhes o conhecimento necessário para a operação da ferramenta. Como os usuários apresentavam um conhecimento de informática limitado, esses treinamentos exigiram simulações passo a passo da operação da ferramenta. Com mais alguma prática no manuseio da ferramenta, os ganhos provocados pela substituição do processo manual poderão reduzir em algumas ordens de grandeza o tempo necessário para o planejamento das escalas de trabalho nas enfermarias do HC.

Nesta data, o algoritmo encontra-se instalado em diversos computadores nas enfermarias do HC, junto com a interface de operação construída para este fim. As sessões de treinamento já foram realizadas e o sistema está pronto para entrar em operação sustentada.

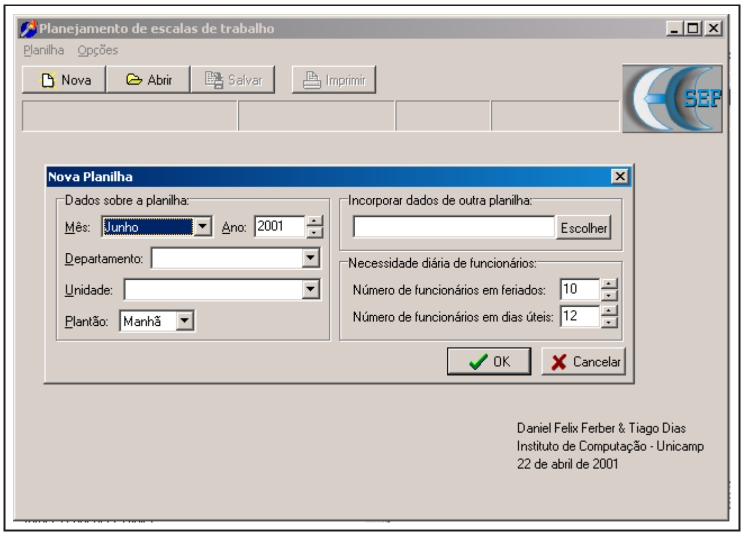
8 A Implementação da Interface

A interface foi desenvolvida com o objetivo de se criar um meio de interação simples e amigável entre o usuário da ferramenta e o algoritmo que realiza a busca de uma solução.

Daniel Felix Ferber & Tiago Dias
Instituto de Computação - Unicamp
22 de abril de 2001

Figura 22: Tela inicial de entrada da Interface

A tela inicial, apresentada a partir da entrada no sistema, mostra um menu principal e um conjunto de botões para abertura ou criação de uma nova escala de dados (vide Figura 22). A partir dessa tela é possível abrir uma escala já previamente digitada clicando-se no botão *Abrir* ou utilizando-se o menu *Planilha/Abrir*. Também é possível criar uma nova escala clicando-se no botão *Nova* ou utilizando-se o menu *Planilha/Criar Nova*. Nesse segundo caso, uma tela como a da Figura 23 é exibida e, após a digitação dos dados, uma tela semelhante a da Figura 24 é apresentada, sem a relação de funcionários, os quais devem



Dados sobre a planilha:

Mês: Junho Ano: 2001

Departamento:

Unidade:

Plantão: Manhã

Incorporar dados de outra planilha:

Necessidade diária de funcionários:

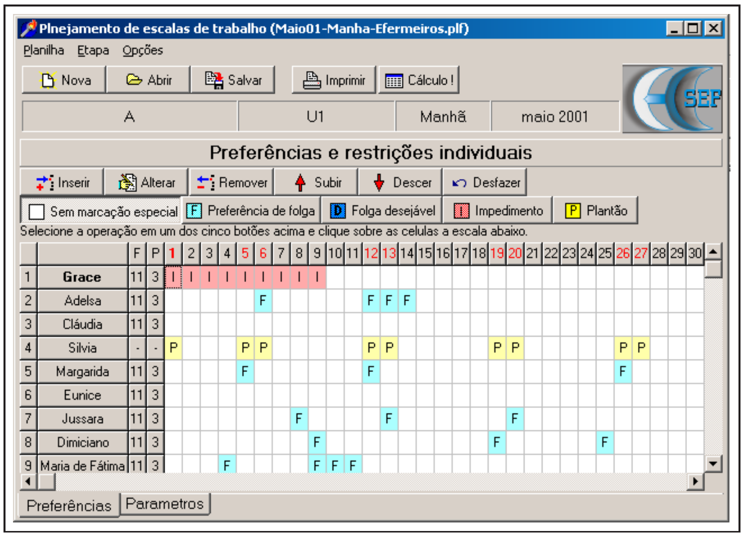
Número de funcionários em feriados: 10

Número de funcionários em dias úteis: 12

Daniel Felix Ferber & Tiago Dias
Instituto de Computação - Unicamp
22 de abril de 2001

Figura 23: Criação de uma nova escala

ser incluídos a partir de um clique no botão *Incluir*. A Figura 25 exibe a tela de inclusão



A | U1 | Manhã | maio 2001

Preferências e restrições individuais

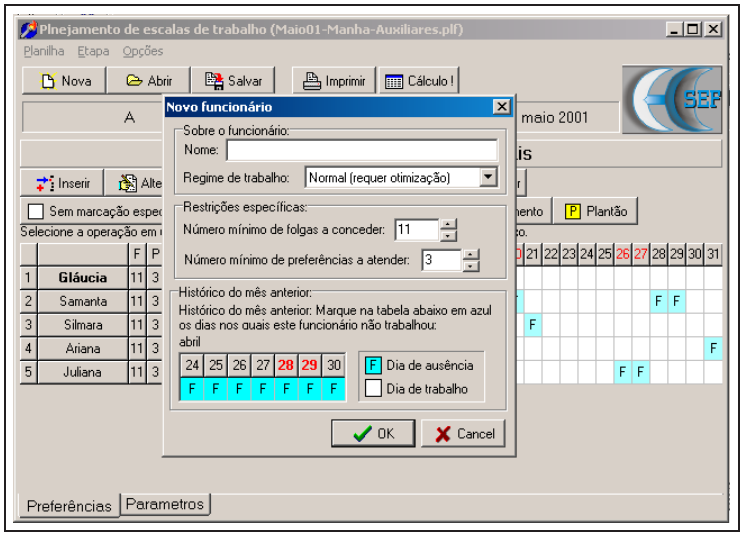
Selecione a operação em um dos cinco botões acima e clique sobre as celulas a escala abaixo.

		F	P	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	**Grace**	11	3	I	I	I	I	I	I	I	I	I																					
2	Adelsa	11	3						F						F	F	F																
3	Cláudia	11	3																														
4	Silvia	-	-	P				P	P						P	P						P	P						P	P			
5	Margarida	11	3					F							F														F				
6	Eunice	11	3																														
7	Jussara	11	3								F					F							F										
8	Dimiciano	11	3									F										F						F					
9	Maria de Fátima	11	3				F					F	F	F																			

Figura 24: Tela que define os funcionários e as preferências

de funcionários.

De posse dos funcionários, os dados necessários para a execução do algoritmo resolvedor podem ser digitados. As restrições e demais dados são definidos em duas áreas básicas que podem ser selecionadas: as preferências e os parâmetros. As preferências (vide Figura 24) definem os funcionários pertencentes àquela escala, possibilitando a sua inclusão, alteração



Novo funcionário

Sobre o funcionário:

Nome:

Regime de trabalho: Normal (requer otimização)

Restrições específicas:

Número mínimo de folgas a conceder: 11

Número mínimo de preferências a atender: 3

Histórico do mês anterior:

Histórico do mês anterior: Marque na tabela abaixo em azul os dias nos quais este funcionário não trabalhou:

abril

24	25	26	27	28	29	30
F	F	F	F	F	F	F

F Dia de ausência

☐ Dia de trabalho

		F	P
1	**Gláucia**	11	3
2	Samanta	11	3
3	Silmara	11	3
4	Ariana	11	3
5	Juliana	11	3

Figura 25: Inserção de um novo funcionário

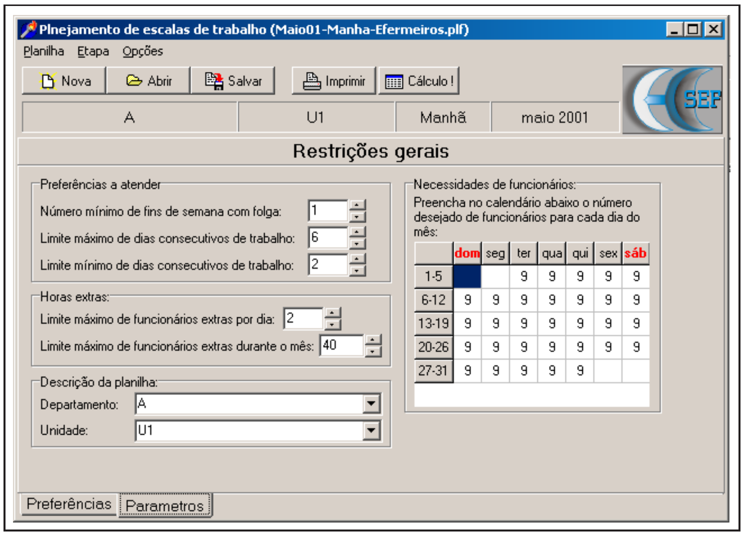
e exclusão a partir de botões auto-explicativos. Além disso, a área de preferências define as preferências de folga (F), os impedimentos (I), folgas desejadas (D) e plantões (P). Esses dados podem digitados na planilha exibida na Figura 24 mediante a escolha de um dos botões: *Sem marcação especial*, *Preferência de Folga*, *Folga desejável*, *Impedimento* e *Plantão*. Isso feito, alguns cliques na planilha definem os dias em que essa opção deve ser aplicada.

A opção por plantões (ícone amarelo) está ligada ao tipo de funcionário. Existem dois tipos de funcionários que trabalham nas enfermarias: o funcionário que trabalha normalmente e obedece padrões de horários fixos (tipo N) e o funcionário que só trabalha nos finais de semana e feriados (tipo P). Esse segundo grupo é formado, normalmente, por residentes e estudantes que não podem trabalhar nos dias de semana, devido a outros compromissos. A opção P (plantão) só é utilizada para o funcionário do tipo P, indicando que, naquele dia, o plantonista está trabalhando. Veja a Figura 24, na qual a funcionária *Silvia* é uma funcionária tipo P. Os seus dias de trabalho se restringem aos finais de semana e são indicados pela letra P.

A tela de definições de parâmetros é mostrada na Figura 26. Nessa tela são informadas algumas restrições para o problema, como o número mínimo de folgas que devem ser concedidas nos finais de semana e o limite máximo de dias consecutivos de trabalho, entre outras. A matriz apresentada nesta tela é a chamada matriz de necessidades, a qual define a necessidade de funcionários em cada um dos dias do mês.

Após digitados todos os dados, inclusive restrições, funcionários, preferências e referências (nomes de departamentos, turno da escala), o sistema está pronto para buscar uma solução para o problema. Porém, antes de executar o algoritmo otimizador, é oferecida ao usuário a oportunidade de salvar os dados digitados.

É importante ressaltar que os dados podem ser salvos a qualquer momento, evitando



A	U1	Manhã	maio 2001

Restrições gerais

Preferências a atender
Número mínimo de fins de semana com folga: 1
Limite máximo de dias consecutivos de trabalho: 6
Limite mínimo de dias consecutivos de trabalho: 2

Horas extras:
Limite máximo de funcionários extras por dia: 2
Limite máximo de funcionários extras durante o mês: 40

Descrição da planilha:
Departamento: A
Unidade: U1

Necessidades de funcionários:
Preencha no calendário abaixo o número desejado de funcionários para cada dia do mês:

	dom	seg	ter	qua	qui	sex	sáb
1-5			9	9	9	9	9
6-12	9	9	9	9	9	9	9
13-19	9	9	9	9	9	9	9
20-26	9	9	9	9	9	9	9
27-31	9	9	9	9	9		

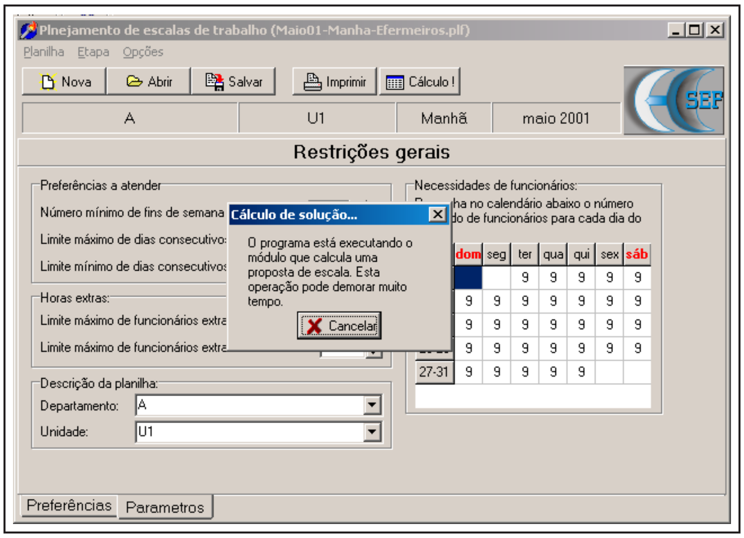
Figura 26: Definição das restrições gerais do problema

perdas. Os dados são salvos em um arquivo (extensão plf — planilha de funcionários) num formato padrão. Esse arquivo é passado como entrada ao algoritmo de resolução antes desse último iniciar sua execução. O arquivo de dados é divido em grupos que congregam as informações digitadas. Um exemplo de dois grupos desse arquivo pode ser visto abaixo:

```
⋮

[ FERIADOS ]
1 "Dia do Trabalho"

[ PREFERENCIAS ]
N IIIIIIIII----------------------
N -----F-----FFF-----------------
N -------------------------------
P P---PP-----PP-----PP-----PP----
N ----F------F-------------F-----
N -------------------------------
N -------F----F------F-----------
N --------F---------F-----F------
N ---F----FFF--------------------
N -----------------FFF-----------
N -------FF--F-------------------
P P---PP-----PP-----PP-----PP----
N -------------------------------
N -------------------------------

⋮
```

Nesse exemplo, o primeiro grupo, *FERIADOS*, define os feriados daquele mês. O segundo grupo, *PREFERENCIAS*, armazena a matriz de preferências dos funcionários e também o seu tipo. Nessa matriz os funcionários estão representados pelas linhas e os dias pelas colunas. A letra F significa folga, a letra I impedimento, a letra D uma folga desejada e a letra P indica dia de trabalho para um funcionário tipo P. O tipo de cada funcionário é definido no início de cada linha (tipo P ou N). As outras restrições do problema também estão armazenadas neste arquivo, organizadas em diferentes grupos.

Nesse instante, o algoritmo resolvedor já pode ser executado, com um clique no botão *Cálculo!* (vide Figura 26). Após esse clique o sistema aguarda a execução e obtenção das soluções pelo algoritmo de otimização, como indicado na Figura 27. Quando a execução do algoritmo termina, esse último cria um novo grupo, chamado *PROPOSTA*, e o armazena no arquivo de entrada. Este grupo contém a solução obtida. Quando a interface lê esse arquivo novamente a solução é exibida, como mostrado na Figura 28. Nessa figura, os círculos ao redor do dia indicam uma das preferências que foi digitada (Figura 24).



A | U1 | Manhã | maio 2001

Restrições gerais

Preferências a atender

Número mínimo de fins de semana

Limite máximo de dias consecutivo

Limite mínimo de dias consecutivos

Horas extras:

Limite máximo de funcionários extra

Limite máximo de funcionários extra

Descrição da planilha:

Departamento: A

Unidade: U1

Necessidades de funcionários:

ha no calendário abaixo o número
do de funcionários para cada dia do

	dom	seg	ter	qua	qui	sex	sáb
			9	9	9	9	9
	9	9	9	9	9	9	9
	9	9	9	9	9	9	9
	9	9	9	9	9	9	9
27-31	9	9	9	9	9		

Cálculo de solução...

O programa está executando o módulo que calcula uma proposta de escala. Esta operação pode demorar muito tempo.

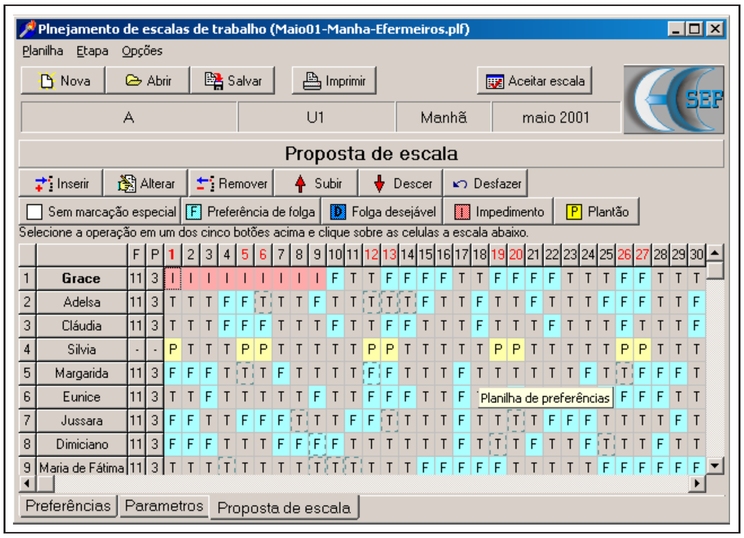
Cancelar

Figura 27: Aguardando a execução do algoritmo de otimização

Nesse momento, o usuário pode analisar a proposta de solução e decidir se aceita a solução, se modifica algum dado e executa novamente o algoritmo otimizador, ou se modifica a própria solução encontrada. No primeiro caso, basta clicar no botão *Aceitar Escala* e uma nova área é aberta (Figura 30) com a solução obtida, a qual é regravada em outro grupo, denominado *ESCALA*, no arquivo texto de entrada padronizado.

Se o usuário desejar alterar algum dado do problema inicial, deve abrir o menu *Etapa* e selecionar a opção *Alterar preferências/parâmetros*. Com isso uma janela é exibida informando que, após as alterações, um novo cálculo deve ser realizado. Então, mediante a aceitação do usuário, a solução obtida anteriormente é descartada e o sistema retorna ao estado exibido na Figura 26.

O usuário pode ainda modificar a solução obtida a seu próprio risco, com possibilida-



Selecione a operação em um dos cinco botões acima e clique sobre as celulas a escala abaixo.

		F	P	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	**Grace**	11	3	I	I	I	I	I	I	I	I	I	F	T	T	F	F	F	F	T	T	F	F	F	F	T	T	T	F	F	T	T	T
2	Adelsa	11	3	T	T	T	F	F	T	T	T	F	T	T	T	T	T	F	T	T	F	T	T	F	T	T	T	F	F	F	T	T	F
3	Cláudia	11	3	T	T	T	F	F	F	T	T	T	F	T	T	F	F	T	T	T	F	T	T	T	F	T	T	T	F	T	T	T	F
4	Silvia	-	-	P	T	T	T	P	P	T	T	T	T	T	P	P	T	T	T	T	T	P	P	T	T	T	T	T	P	P	T	T	T
5	Margarida	11	3	F	F	F	T	T	T	F	T	T	T	T	F	F	T	T	T	F	T	T	T	T	T	T	F	T	T	F	F	F	T
6	Eunice	11	3	T	T	F	T	T	T	T	T	F	T	T	F	F	F	T	T	F	Planilha de preferências								F	F	F	T	T
7	Jussara	11	3	F	F	T	T	F	F	F	T	T	T	F	F	T	T	T	T	F	T	T	T	T	F	F	F	T	T	T	T	F	T
8	Dimiciano	11	3	F	F	F	T	T	T	F	F	F	F	T	T	T	T	T	T	F	T	T	T	F	T	T	F	T	T	T	F	T	T
9	Maria de Fátima	11	3	T	T	T	T	T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	T	T	T	T	T	F	F	F	F	F	F

Figura 28: Proposta de soluções encontrada pelo algoritmo

des de violar restrições estabelecidas. Essa alteração pode ser feita da mesma forma que a digitação da planilha de preferências, atribuindo folgas (F), folgas desejáveis (D), impedimentos (I), plantões (P) ou excluindo qualquer marcação especial. Após essas modificações o usuário pode aceitar a solução clicando em *Aceitar Escala* e seguir para a tela mostrada na Figura 30. Essa possibilidade de alteração é importante para o hospital, pois permite adaptar manualmente a escala contruída. Além disso, na ocorrência de pedidos de licença ou qualquer tipo de impedimento após a obtenção da solução, é possível alterar a escala ora em uso sem a necessidade de se executar o algoritmo otimizador novamente.

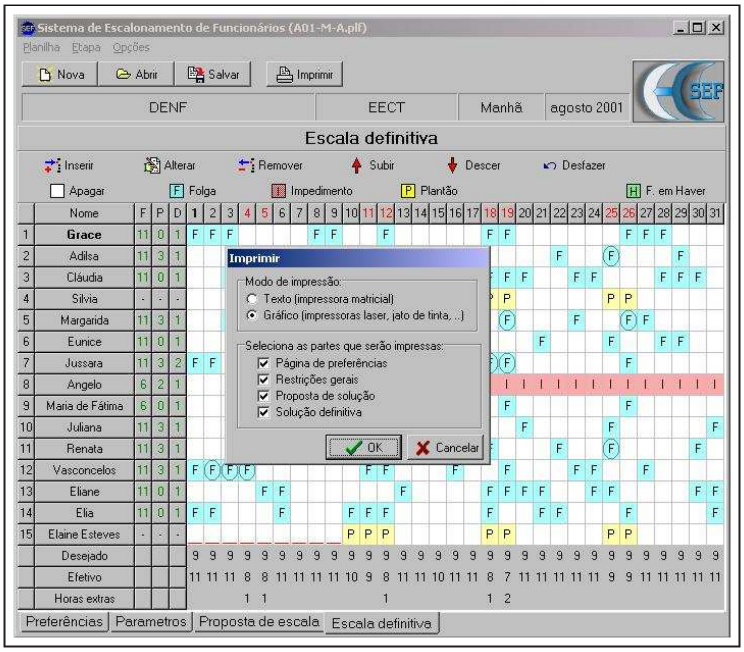
Após o usuário aceitar uma das soluções propostas, a planilha pode ser impressa acionando-se o botão *Imprimir* ou através do menu *Planilha/Imprimir*, como ilustrado na Figura 29.

Para que o usuário possa acompanhar com facilidade as conseqüências de suas ações, a interface implementa algumas facilidades a mais, tais como:

- Um rodapé com uma linha onde aparece o número de funcionários disponibilizados a cada dia, além de outras duas linhas contendo o número mínimo desejado e o número de funcionários extras necessários em cada dia.

- À direita da escala, várias colunas contabilizam, para cada funcionário, o número de folgas, o número de preferências atendidas e o número de finais de semanas concedidos como folgas. Quando um dos números extrapola um de seus limites, o mesmo é realçado em vermelho.

- Seqüências de dias de folgas e dias de trabalho indesejáveis são marcadas com um traço vermelho. Por exemplo, são marcados os períodos nos quais o funcionário trabalha por mais de seis dias consecutivos, ou quando recebe folgas intercaladas com um único dia



DENF | EECT | Manhã | agosto 2001

Escala definitiva

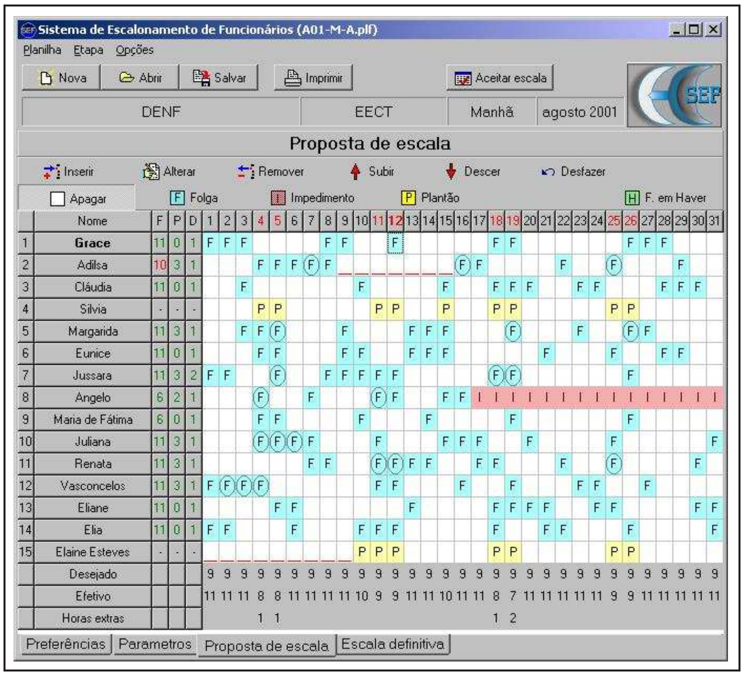
Imprimir

Modo de impressão:
○ Texto (impressora matricial)
● Gráfico (impressoras laser, jato de tinta, ..)

Seleciona as partes que serão impressas:
☑ Página de preferências
☑ Restrições gerais
☑ Proposta de solução
☑ Solução definitiva

OK
Cancelar

Figura 29: Tela de impressão



DENF | EECT | Manhã | agosto 2001

Proposta de escala

☐ Apagar | F Folga | I Impedimento | P Plantão | H F. em Haver

	Nome	F	P	D	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	**Grace**	11	0	1	F	F	F					F	F			F						F	F							F	F	F			
2	Adilsa	10	3	1				F	F	F	Ⓕ	F								Ⓕ	F					F			Ⓕ				F		
3	Cláudia	11	0	1			F							F					F			F	F	F			F	F				F	F	F	
4	Silvia	-	-	-				P	P						P	P			P			P	P						P	P					
5	Margarida	11	3	1			F	F	Ⓕ				F				F	F	F				Ⓕ				F			Ⓕ	F				
6	Eunice	11	0	1				F	F				F	F			F	F	F						F				F			F	F		
7	Jussara	11	3	2	F	F			Ⓕ			F	F	F	F	F						Ⓕ	Ⓕ							F					
8	Angelo	6	2	1				Ⓕ			F				Ⓕ	F			F	F	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
9	Maria de Fátima	6	0	1				F	F					F				F					F							F					
10	Juliana	11	3	1				Ⓕ	Ⓕ	Ⓕ	F				F				F	F	F			F					F						F
11	Renata	11	3	1							F	F			Ⓕ	Ⓕ	F	F			F	F				F			Ⓕ					F	
12	Vasconcelos	11	3	1	F	Ⓕ	Ⓕ	Ⓕ							F	F				F			F				F	F			F				
13	Eliane	11	0	1					F	F							F					F	F	F	F			F	F					F	F
14	Elia	11	0	1	F	F				F				F	F	F						F			F	F				F					F
15	Elaine Esteves	-	-	-										P	P	P						P	P						P	P					
	Desejado				9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
	Efetivo				11	11	11	8	8	11	11	11	11	10	9	9	11	11	10	11	11	8	7	11	11	11	11	11	9	9	11	11	11	11	11
	Horas extras							1	1													1	2												

Figura 30: Solução definitiva aceita pelo usuário

de trabalho. Na Figura 30, uma das folgas atribuídas à funcionária *Adilsa* foi retirada, ilustrando a marcação quando se tem uma violação de dias consecutivos de trabalho. Nesse caso, o número máximo de dias consecutivos de trabalho foi configurado como 6 e a funcionária está trabalhando 7 dias consecutivos.

As setas de deslocamento *Sobe* e *Desce*(Figura 24) podem ser utilizadas para redistribuir os funcionários na matriz de preferências, deslocando-os para baixo ou para cima. O botão *Desfazer* desfaz a última alteração ou digitação comandada pelo usuário.

Outras funcionalidades implementadas pela interface incluem a possibilidade de salvar a planilha digitada com outro nome, utilizando-se, para isso, a opção *Planilha/Salvar Como*, e também ajustar e configurar a impressora utilizando-se a opção *Opções/Ajustar impressora* ou *Opções/Configurar*.

9 Conclusões

As duas heurísticas desenvolvidas resolveram de forma muito satisfatória o problema de gerar escalas de trabalho adequadas para as enfermarias do HC. As escalas são geradas em tempo relativamente pequeno, da ordem de 30 a 50 segundos no pior caso, e são passíveis de utilização imediata pela administração do hospital.

A ferramenta computacional desenvolvida e implementada permite a efetiva substituição da metodologia manual de construção de escalas de serviço no HC, em uso até o presente momento, por um processo (semi-)automatizado de construção dessas escalas. Se utilizada adequadamente, a ferramenta computacional permite reduzir de forma significativa o tempo necessário para o planejamento de escalas de trabalho nas enfermarias do HC.

Já está em curso o processo para disponibilizar a ferramenta para uso em todas as enfermarias do HC.

O projeto foi também reconhecido pela Sociedade Brasileira de Pesquisa Operacional, que o classificou entre os 5 finalistas na avaliação de trabalhos de iniciação científica desenvolvidos no ano de 2001, e estendeu um convite aos autores para que o expusessem em uma sessão técnica específica no XXXIII Simpósio de Pesquisa Operacional, com todas as despesas pagas pelos organizadores daquele envento.

Referências

[1] Reuniões com a Diretoria de Enfermagem do Hospital das Clínicas, agosto 2000.

[2] Reuniões com a Diretoria de Enfermagem do Hospital das clínicas, Agosto de 2001.

[3] Azuma Ohuchi Ahmad Jan, Masahito Yamamoto. Genetic evolutionary algotithms fur nurse scheduling problem. *Hokkaido University*, 1994.

[4] Vittorio Maniezzo Alberto Colorni, Marco Dorigo. Genetic algorithms and highly constrained problems – the time–table case. *Dipartamento di Elettronica — Politecnico de Milano.*

[5] Emma Collingwood. Investigation of a multiple chromosome evolutionary algorithm for bus driver scheduling and other problems. *MS Information Technology*, 1994.

[6] A. Colorni, M. Dorigo, and V. Maniezzo. A genetic algorithm to solve the timetable problem. Technical Report 90-060, Politecnico di Milano, Milan, Italy, 1992.

[7] Alberto Colorni, Marco Dorigo, and Vittorio Maniezzo. Genetic algorithms and highly constrained problems: the time-table case. In *Proceedings of the First International Conference on Paralel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, pages 55– 59, Dortmund, Germany. Springer-Verlag.

[8] Alberto Colorni, Marco Dorigo, and Vittorio Maniezzo. Genetic algorithms: a new approach to the time-table problem. In Agkül M., editor, *Combinatorial Optimization*, volume F 82 of *Lecture Notes in Computer Science, NATO ASI Series*, pages 235– 239. Springer-Verlag, 1990.

[9] Lawrence Davis, editor. *Handbook of genetic algorithms.* International Thomson Computer Press, Boston, Massachusetts, 1996.

[10] Lawrence Davis. *Handbook of genetic algorithms.* London, 1996.

[11] M. Dorigo. Genetic algorithms: the state of the art and some research proposals. Technical Report 89-058, Politecnico di Milano, 1989.

[12] Hsiao-Lan Fang. Genetic algorithms in timetabling and scheduling. *Department of Artificial Intelligence — University of Edinburgh*, 1994.

[13] Michael R. Garey and Johnson David S. *Computers and intractability: A guide to the theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, California, 1979.

[14] Fred Glover. Tabu search: a tutorial. Technical report, Center for Applied Artificial Inteligence, University of Colorado, 1990.

[15] Fred Glover, James P. Kelly, and Manuel Laguna. Genetic algorithms and tabu search: hybrids for optimization. *Computers and Operations Research*, 22(1):111–134, 1995.

[16] David E. Goldberg, editor. *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1989.

[17] David E. Goldberg. *Genetic algorithms in search optimization and machine learning.* Reading, Mass., 1989.

[18] A. Haerf. A survey of automated timetalbing. *CWI*, 1995.

[19] John H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, Michigan, 1975.

[20] C.L. Huntley and Brown D.E. Paralel genetic algorithms with local search.

[21] P.C. Chu J. E. Beasley. A genetic algorithm for the covering set problem. *Imperial College*, 1994.

[22] Scott Ladd. *Genetic algorithms in C++*. New York, 1996.

[23] H. Mühlenbein, M. Georges-Schleute, and O. Krämer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7:65– 88, 1988.

[24] V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith. *Modern Heuristic Search Methods*. Wiley, 1996.

[25] Colin R. Reeves. Genetic algorithms for the operations researcher. *INFORMS Journal pn Computing*, 1997.

[26] Darrel Whitley. A genetic algorithm tutorial. *Department of Computer Science — Colorado State university*, 1993.

[27] Laurence A. Wolsey. *Integer Programming*. Wiley-Interscience, 1998.

[28] Anthony Wren and David O. Wren. A genetic algorithm for public transport driver scheduling. *Computers and Operations Research*, 22(1):101–110, 1995.