UMA APLICAÇÃO DA METAHEURÍSTICA ITERATED LOCAL SEARCH AO SHIFT DESIGN PROBLEM

Cynthia da Silva Barbosa (CEFET-MG)
cysb@terra.com.br
Sérgio Ricardo de Souza (CEFET-MG)
sergio@dppg.cefetmg.br
Gray Farias Moita (CEFET-MG)
gray@dppg.cefetmg.br

Este trabalho apresenta a aplicação da Metaheurística Iterated Local Search (ILS) à solução do Shift Design Problem (SDP) aplicado à criação de turnos de trabalho de uma empresa de Call Center. O objetivo deste trabalho é determinar um coonjunto de soluções factíveis que contenham turnos e o número de funcionários por turno que minimizem o número de turnos distintos, o excesso e a escassez de funcionários, e as diferenças do número médio de tarefas executadas por funcionários, por semana, respeitando as restrições das leis trabalhistas. Este problema pertence à classe de problemas NP-difíceis, justificando-se o uso de técnicas heurísticas para sua solução. O método de busca local utilizado para a geração da solução inicial para o ILS é o Método da Descida, que apresenta baixo custo computacional na implementação realizada. Os resultados mostram que o método proposto é capaz de gerar soluções viáveis, tanto na qualidade da solução final, quanto na rapidez.

Palavras-chaves: Shift Design Problem, Call Center, Metaheurística, Iterated Local Search.

1. Introdução

A criação de horários de trabalho de funcionários é uma das etapas do processo de planejamento de uma empresa de Call Center, segundo Bhulai et al. (2007). É nesta etapa que são gerados os turnos de trabalho para a designação dos funcionários.

Neste trabalho é estudado o problema de criação de turnos de trabalho conhecido na literatura como *Shift Design Problem* (SDP) apresentado em Musli et al. (2004) aplicado ao planejamento dos funcionários de uma empresa de *Call Center*. O SDP consiste em determinar um conjunto de soluções factíveis que contenham turnos e o número de funcionários por turno que minimizem o número de turnos distintos, o excesso e a escassez de funcionários, e as diferenças do número médio de tarefas executadas por funcionários, por semana. Este problema considera a alocação dos funcionários após as mudanças geradas nos turnos de trabalho utilizando métodos de busca local. Para a geração das mudanças dos turnos é definido um algoritmo para a geração de uma boa solução inicial e um conjunto de movimentos para a geração da vizinhança. A utilização de métodos exatos pode se tornar inviável, devido a um grande número de restrições ao aplicadas ao SDP motivando a utilização de heurísticas e metaheurísticas na resolução do problema, que podem garantir bons resultados.

Um problema similar ao SDP, relatado na literatura no início da década de 50 como *Shift Scheduling Problem* através do trabalho de Dantzig (1954) foi resolvido por programação inteira. Nesta formulação, encontra-se um conjunto ótimo de trocas (permutação) de horários. As possíveis mudanças são enumeradas com base no início e na duração do turno e no intervalo de tempo para descanso. Quando o número de turnos aumenta, essa formulação logo se torna impraticável.

Bechtold and Jacobs (1990) apresentaram uma nova formulação implícita de programação linear com a inclusão de intervalo de descanso para a programação de turnos. Os autores encontraram resultados superiores aos obtidos em Dantzig (1954), porém esta abordagem é limitada às organizações que operam com menos de 24 horas de trabalho por dia.

Thompson (1995) apresenta um modelo de programação inteira para o desenvolvimento de horários de turnos de trabalho, permitindo uma grande flexibilidade em relação a turnos de trabalhos alternados, a comprimento dos turnos e aos intervalos de descanso. O modelo está relacionado aos trabalhos de Moondra (1976) e Bechtold e Jacobs (1990), em que o comprimento dos turnos e os intervalos de pausas são modelados implicitamente.

Em Aykin (1996) é proposto um modelo de programação inteira sem limitações e em Aykin (2000) são comparados diferentes tipos de abordagem para o *Shift Scheduling Problem*.

Este trabalho trata o SDP para a criação dos turnos de trabalho dos funcionários de uma empresa de Call Center após as trocas de horários nos turnos de trabalho, utilizando um algoritmo guloso para a definição de uma solução inicial e o método heurístico *Iterated Local Search* (ILS) para refinar a solução final, com o objetivo de obter soluções que contenham a quantidade de funcionários por turno e que minimize a quantidade de turnos distintos e a quantidade de funcionários por turno, de acordo com as leis trabalhistas brasileiras.

Na literatura, são encontradas algumas abordagens para resolver a fase de criação de horários de trabalho de funcionários de uma empresa. Em Glover e McMillan (1986), os problemas de geração dos turnos e a atribuição dos deslocamentos de horários para a designação dos

empregados são resolvidos como um único problema. A abordagem conhecida como *Shift Design Problema* (SDP) é apresentada em Musliu et al. (2004) e considera a alocação real dos funcionários após as mudanças geradas nos turnos de trabalho. Segundo estes autores, a principal desvantagem da última abordagem é que ela não garante que uma boa solução para a atribuição das mudanças dos turnos seja encontrada. Por outro lado, resolvendo-se o problema global das várias fases distintas, a solução do SDP torna-se mais fácil de ser obtida. Ainda segundo os autores, o problema de minimização do número de turnos é um problema NP - difícil e a definição da estrutura de vizinhança adequada é uma das características mais importantes das técnicas de busca local.

Em Musliu et al. (2008), os autores consideram os intervalos de descanso (pausas) para a geração dos turnos de trabalho dos funcionários. O grande desafio para esta classe de problema é satisfazer os requisitos legais e garantir que um certo número de funcionários estejam presentes em todo o turno. A quantidade de pausas em um turno deve ser programada de tal forma que o número de violações e restrições em relação às pausas e o excesso ou a escassez dos funcionários sejam minimizadas.

Em Tellier e White (2006) e Canon (2007), os autores apresentam o problema através do planejamento de pausas, com dias de descanso e férias dos funcionários. O problema foi resolvido utilizando-se técnicas de busca local para encontrar soluções com qualidade aceitável baseados em heurística para refinar a solução final.

No presente trabalho, o planejamento dos turnos de trabalho com dias de descanso, as horas extras e as férias dos funcionários não serão contemplados.

Este trabalho está organizado como segue. Na Seção 2, são descritas as características do problema estudado e a formulação matemática do problema. Na Seção 3, é detalhado o método da descida, enquanto na Seção 4, é descrito o método ILS. Na Seção 5, são apresentadas a metodologia adotada e a aplicação do método da descida e ILS. Na Seção 6 são apresentados e discutidos os resultados computacionais. A Seção 7 conclui o trabalho.

2. Descrição do problema estudado

O problema estudado neste trabalho é o *Shift Design Problem* (SDP) aplicado ao planejamento dos funcionários de uma empresa de *Call Center*. A solução para este problema consiste em encontrar a quantidade de funcionários por turno de trabalho, que minimize a quantidade de turnos distintos e a quantidade de funcionários por turno, de acordo com as leis trabalhistas brasileiras.

Neste trabalho, os requisitos das tarefas de trabalho por um determinado período de tempo, juntamente com as restrições sobre os possíveis horários de início e da duração dos turnos e a quantidade média de ligações atendidas por funcionário por semana são conhecidos.

2.1 Formulação matemática

A formulação matemática do SDP adotada neste artigo foi adaptada do trabalho de Musliu et al. (2004) e é descrita a seguir. Considere, então, que:

– *n* representa o número de intervalos de tempo consecutivos ($[a_1,a_2]$, $[a_2,a_3]$, ..., $[a_n, a_{n+1}]$) , todos com o mesmo comprimento, e representados em minutos. Cada intervalo $[a_i, ai_{+1}]$ está relacionado com o número w_i indicando a quantidade ideal de funcionários para um determinado intervalo de tempo. Cada intervalo $[a_i, ai_{+1}]$ possui uma duração de 30 minutos. O instante de tempo a_1 representa o início do turno de trabalho e o instante de tempo a_{n+1} representa o fim do turno de trabalho;

– y representa o tipo do turno v_1,..., v_y , conforme apresentados na Tabela 1. Cada tipo de turno v_j possui os seguintes parâmetros:
  - v_i . início-min e v_i.início-max representam a faixa de tempo para o início-mínimo e o início-máximo em que o turno poderá iniciar;
  - v_i. min-comp e v_i.max-comp representam o comprimento mínimo e o comprimento máximo do turno de trabalho. Neste trabalho, considera-se que a duração máxima do turno não poderá ultrapassar 6 horas de trabalho por dia.

Tipo Turno	Início-Min	Início-Max	Min-Comp	Max-Comp
Manhã	06:00	09:00	06:00	08:00
Tarde	12:00	15:00	06:00	08:00
Noite	18:00	21:00	06:00	08:00

Tabela 1 – Tipos de turnos

Conhecidas as variáveis aplicadas a este trabalho, o objetivo é gerar um conjunto k de turnos s_1,...,s_k, minimizando a quantidade de turnos distintos e a quantidade de funcionários em cada turno.

Cada turno s_l possui parâmetros de início s_1. *inicio* e de duração s_1 .*duração*, sendo o horizonte de tempo de planejamento de uma semana.

O objetivo, então, é minimizar os quatro componentes abaixo:

– F_1: soma dos excessos de trabalhadores em cada intervalo de tempo, durante o período de planejamento.
– F_2: soma da escassez de trabalhadores em cada intervalo de tempo, durante o período de planejamento.
– F_3: número de turnos.
– F_4: média da carga de trabalho por semana, caso esteja acima do limite.

Em Musliu et al. (2004) este problema é apresentado como um problema de otimização multi-critério. Os critérios de decisão possuem importâncias diferentes, dependentes da situação específica abordada, de modo que a função objetivo tratada é a soma ponderada dos quatro componentes citados acima, tendo os pesos como função dos dados da instância.

A carga de trabalho l_d , para um determinado intervalo de tempo d, utilizada para definir o excesso ou a escassez de trabalhadores, representa a quantidade de turnos de trabalho que um funcionário poderá trabalhar por semana, sendo dada por: :

$$l_d = \sum_{p=1}^{k} X_{p,d}$$

para:

$$X_{pd} = \begin{cases} s_p.\, w_i & \text{se o intervalo de tempo } d \text{ pertence ao turno } s_p \text{ no dia } i, \\ 0 & \text{caso contrário} \end{cases}$$

Esta restrição garante que o número de funcionários trabalhando em um intervalo de tempo d não pode ser inferior à quantidade necessária em um turno de trabalho.

O excesso F1 e a escassez F2 (ambos dados em minutos) dos funcionários em todos os intervalos de tempo durante o período de planejamento é definido como:

$$F_1 = \sum_{d=1}^{n} (\max(l_d - wd, 0) * duracaoturno)$$

$$F_2 = \sum_{d=1}^{n} (\max(wd - l_d, 0) * duracaoturno)$$

sendo que *wd* representa a quantidade de funcionários em cada intervalo de tempo.

A penalidade associada à média da carga de trabalho, referente ao número de turnos que um funcionário poderá trabalhar por semana, é definida como:

$$F_4 = \max(AvD - AS, 0)$$

sendo:

- *AvD* representa a média do número de turnos de trabalho por semana por funcionário;
- *AS* representa o limite superior para a média do número de turnos de trabalho por semana, por funcionário.

A penalidade F_4 não se aplica ao presente artigo, pois o limite máximo da média do número de turnos de trabalho é 6 horas por dia, conforme a legislação trabalhista brasileira para o setor de atendimento telefônico. Portanto, a média por semana será sempre um e a penalidade F_4 será sempre zero.

Assim, a função objetivo a ser minimizada no caso do presente artigo é:

$$FO = \alpha F_1 + \alpha F_2 + \alpha F_3$$

sendo as penalizações definidas conforme já posto.

3. Método da descida

É um método de busca local que analisa todos os possíveis vizinhos de uma solução *s* em sua vizinhança *N(s),* escolhendo, a cada passo, aquele que tem menor valor para a função de avaliação, ele também é chamado de método guloso, segundo Costa (2003). É importante observar, que para efetivar a mudança o vizinho candidato deve melhorar estritamente o valor da melhor solução obtida até o momento. O critério de parada se dá quando um mínimo local é encontrado. O mínimo local é a solução *s* em que nenhum de seus vizinhos $s' \in N(s)$ tem o valor de função de avaliação menor.

Entende-se por vizinho de uma solução *s* a solução *s'* que sofreu a modificação *m*, Costa (2003). Representa-se essa operação por $s' \leftarrow s \oplus m$. Mais especificamente seja *S* o espaço de pesquisa de um problema de otimização. A função *N*, a qual depende da estrutura do problema tratado, associa a cada solução viável $s \in S$, sua vizinhança $N(s) \in S$, ou seja, todas as soluções *s'* geradas a partir de um movimento m. Cada solução $s' \in N(s)$ é chamada de vizinho de *s*.

4. *Iterated Local Search*

O método *Iterated Local Search* (ILS), apresentado em Lourenço et al. (2003), é baseado na idéia de que um procedimento de busca local pode ser melhorado, gerando-se novas soluções de partida, as quais são obtidas por meio de perturbações na solução ótima local. A perturbação precisa ser suficientemente forte para permitir que a busca local explore

diferentes soluções, mas também fraca o suficiente para evitar um reinício aleatório.

Para aplicar um algoritmo *ILS*, quatro componentes têm que ser especificados, segundo Lourenço et al. (2003):

- *Procedimento GeraSolucaoInicial()*, que gera uma solução inicial *so* para o problema;
- *Procedimento BuscaLocal*, que retorna uma solução melhorada *s''*;
- *ProcedimentoPerturbacao*, que modifica a solução corrente *s* guiando a uma solução intermediária *s'* e;
- *Procedimento CriterioAceitacao*, que decide de qual solução a próxima perturbação será aplicada.

A Figura 1, encontrada em Lourenço et al. (2003) apresenta o pseudo-código do método *ILS.*

```
Procedimento ILS()
1.    s0 ← GeraSolucaoInicial();
2.    s ← BuscaLocal(s0);
3.    iter ← 0;
4.    enquanto (iter < itermax)
5.          iter ←  iter + 1;
6.          s' ← Perturbacao(nível, s);
7.          s" ← BuscaLocal(s');
8.          s ← CriterioAceitacao(s");
9.    fim-enquanto;
10.   retorne s;
fim ILS;
```

Figura 1: Pseudocódigo do método *ILS*

5. Metodologia de solução do problema

5.1 Representação de uma solução

Uma solução inicial *s* para o SDP é gerada através de um método guloso, definindo a quantidade necessária de funcionários para cada intervalo de tempo, para o atendimento da demanda. O algoritmo guloso escolhe uma solução adequada uma por vez, fazendo uma escolha ótima local.

Para explorar o espaço de soluções do problema, são aplicados dois tipos diferentes de movimentos, para definir as estruturas de vizinhança:

- - Movimento da quantidade de funcionários: Nesse movimento, a vizinhança da solução é obtida alterando a quantidade de funcionários em um determinado intervalo de tempo, acrescida ou decrescida de uma unidade, retornando-se como vizinho aquela solução que apresentar o melhor valor da função objetivo.
- - Movimento de início do turno: Nesse movimento, a vizinhança da solução é obtida alterando o início do turno, acrescida ou decrescida de um intervalo de tempo (30 minutos), retornando-se como vizinho aquela solução que apresentar o melhor valor da função objetivo.

5.2. Método da descida aplicado problema

A aplicação do método da descida ao problema está descrita a seguir.

Seja *s* uma solução do problema e seja uma solução *s'* pertencente a uma vizinhança de *s* definida pela quantidade necessária de funcionários por turno. Assim, *s'* é gerado a partir do movimento *m* realizado em *s*. Um movimento *m* em *s* é definido como acrescentar ou diminuir a quantidade de funcionários em um determinado intervalo de tempo. Por exemplo, para o turmo que se inicia às 06:30hs são necessários 22 funcionários para o atendimento da demanda. Assim, move-se um funcionário para o turno anterior, 06:00hs e outro funcionário para às 07:00 hs. Esta forma de seleção de funcionários, aplica-se também aos movimentos do início dos turnos de trabalho. Estes movimentos implicam uma chance maior da solução corrente se tornar viável. O critério de parada consiste no número máximo de iterações sem melhora.

O método da descida aplicado a este trabalho realiza sempre a melhor troca de posições no vetor de entrada que contém a quantidade de funcionários. Todas as possíveis trocas são avaliadas, mas somente a melhor é realizada. A função ainda retorna o menor valor entre os melhores índices (**iMelhor** e **jMelhor**), ou seja, os índices que participaram da melhor troca.

A função recebe como entrada os seguintes parâmetros:

- *matriz s* – matriz que armazena sempre a melhor solução encontrada;
- *int nSlots* – quantidade de intervalos de tempo a cada 30 minutos;
- *int slotsPessoa* - quantidade de intervalos de tempo que uma pessoa ocupa de forma contínua, ou seja, o comprimento do turno de trabalho.

A Figura 2, apresenta o pseudo-código do método da descida aplicado ao SDP.

```
Algoritmo metodoDescida()
    int iMelhor
    int jMelhor
    para i ← 1 até (nSlots - slotsPessoa) faça
    {
        para j ← (i+1) até (nSlots – slotsPessoa + 1) faça
        {
            troca de posições i e j
            avalia se a função objetivo melhorou
            se melhorou, armazena as posições em iMelhor e jMelhor
            desfaz a troca
         }
     }
     faz a troca entre as posições iMelhor e jMelhor
     retorne iMelhor
fim metodoDescida;
```

Figura 2: Pseudocódigo do método da descida aplicado ao SDP

5.3. *ILS* aplicado ao SDP

Para resolver o problema, o método *ILS* foi adaptado da seguinte forma: como método de busca local utilizou-se o método da descida descrito na seção anterior, recebendo o valor do menor índice referente a melhor troca. Este índice é armazenado na variável **iMelhor**. Logo após, são realizados dois tipos de pertubação:

- - Perturbação 1: neste movimento um intervalo de tempo é selecionado e o horário de

início do turno é acrescido ou decrescido de uma unidade, retornando-se como vizinho aquele que apresentar o melhor valor da função objetivo atual.

– - Pertubação 2: neste movimento a quantidade de funcionários por intervalo de tempo é acrescida ou decrescida de uma unidade, retornando-se como vizinho aquele que apresentar o melhor valor da função objetivo atual.

Portanto, essas perturbações consistem em aumentar ou diminuir a quantidade de funcionários e o horário de início do turno, em um intervalo de tempo. O critério de aceitação define que uma solução gerada pelo método de busca local é aceita, isto é, $s \leftarrow s'$, se s' apresentar valor da função objetivo menor que a da melhor solução s encontrada até o momento, isto é, se $f(s') < f(s)$. Caso a função objetivo apresente a melhor solução, são armazenados os melhores resultados e o processo é reiniciado para o nível de perturbação igual a 1, como uma nova posição de referência definido pelo execução do *metodoDescida* com a nova solução. O segundo nível de pertubação, consiste em realizar dois movimentos, ou seja, são trocados dois funcionários, e assim por diante. A cada iteração sem melhora, o nível de perturbação é modificado de acordo com o seguinte esquema:

– Nível 1 de perturbação: consiste em realizar um único movimento.
– Nível 2 de perturbação: são realizados dois movimentos, e assim sucessivamente, até o nível máximo definido.

Estas perturbações são realizadas pela função *perturbacaoLocal*,descrita a seguir.

A função *perturbacaoLocal* gera perturbações de uma posição específica referente a quantidade de funcionários em um intervalo de tempo. As perturbações têm como referência, a posição calculada a partir das variáveis **iMelhor** e **distPertub**. São feitas as alterações para mais e para menos começando com o acréscimo (ou decréscimo) de 1 e indo até a quantidade especificada em **pertubMax**. A função recebe como entrada os seguintes parâmetros:

– *matriz s* : matriz que armazena sempre a melhor solução encontrada;
– *matriz s'* : matriz que realiza cálculos;
– *int distPertub* : distância a ser pertubada de acordo com o resultado armazenado em **iMelhor**;
– *int pertubMax* : limite superior para a quantidade de perturbações;
– *int *iMelhor* : posição de referência para calcular a posição a ser perturbada;
– *int nSlots* : quantidade de intervalos de tempo a cada 30 minutos;
– *int slotsPessoa* : quantidade de intervalos de tempo que uma pessoa ocupa de forma contínua, ou seja, o comprimento do turno de trabalho.

A Figura 3, apresenta o pseudocódigo da função *perturbacaoLocal.*

```
Algoritmo perturbacaoLocal ()
  int perturb ← 1
  indicePerturbacao ← iMelhor + distPertub
  enquanto (não atinge pertubação máxima) faça
  {   s' ← s
      pertuba s' na posição indicePertubacao com pertub
      iMelhor ← metodoDescida(s')
      se fo (s') < fo (s) então faça
      {   s ← s'
```

```
            pertub ← 1
          }
        senão faça {    s' ← s
                        perturba s' na posição índicePertubacao com –pertub
                        iMelhor ← metodoDescida (s')
                        se fo (s') < fo (s) então faça
                           {     s ← s'
                                 pertub ← 1
                            }
                    }
   }
fim perturbacaoLocal;
```

Figura 3: Função *perturbacaoLocal*

Uma perturbação no método ILS consiste em trocar um funcionário de um intervalo de tempo para outro, ou seja, na perturbação de nível 1, são trocados, um funcionários por intervalo de tempo , na perturbação de nível 2, são trocados dois funcionários, e assim por diante. O mesmo processo é realizado com as trocas realizadas no início do turno de trabalho, ou seja, na perturbação de nível 1, são trocados o horário que se inicia um turno, na perturbação de nível 2, são trocados dois intervalos de tempo, nos quais se inicia um turnos, e assim por diante. O *metodoILS* chama a função *pertubacaoLocal* para realizar estes movimentos.

Sempre que uma solução *s*' é aceita, a perturbação volta ao nível 1. O critério de parada do ILS é o número máximo de perturbações feitas sem melhora durante a execução do método, representado pela variável **distPertubMax**.

O método *ILS* recebe como entrada os seguintes parâmetros:

- matriz s – matriz que armazena sempre a melhor solução encontrada;
- int nSlots – quantidade de intervalos de tempo a cada 30 minutos;
- int slotsPessoa - quantidade de intervalos de tempo que uma pessoa ocupa de forma contínua, ou seja, o comprimento do turno de trabalho.

A Figura 4, encontrada apresenta o pseudocódigo do método *ILS* aplicado ao SDP.

```
Procedimento metodoILS()

  int distPertubMax ← valor definido de acordo com o número máximo de perturbações
  int perturbMax ← valor definido de acordo com as perturbações a serem realizadas
  int iMelhor
  int distPerturb
  matriz s'
  iMelhor ← metodoDescida(s)
  aloca s' na memória
  para distPerturb ← 1 até distPertubMax  faça
  {
    perturbacaolocal(s, s', -distPerturb, perturbMax, iMelhor, nSlots, slotsPessoa)
    perturbacaolocal(s, s', distPerturb, perturbMax, iMelhor, nSlots, slotsPessoa)
fim metodoILS;
```

Figura 4: Pseudocódigo do método *ILS* aplicado ao SDP

6. Apresentação e Análise dos Resultados

O algoritmo foi implementado na linguagem C, e compilado em DEV C++. Os testes foram realizados em um computador Intel Core i5 430M, com 4GB de memória RAM DDR 3, sob o sistema operacional Windows 7. Para avaliá-lo, utilizou-se um arquivo contendo dados de teste, com a quantidade total de atendentes necessários em cada intervalo de tempo, conforme a Tabela 2.

Horário	Qtde Atendentes	Horário	Qtde Atendentes	Horário	Qtde Atendentes
06:00	15	12:00	126	18:00	121
06:30	22	12:30	112	18:30	99
07:00	38	13:00	112	19:00	83
07:30	62	13:30	122	19:30	77
08:00	118	14:00	141	20:00	57
08:30	157	14:30	157	20:30	57
09:00	169	15:00	161	21:00	49
09:30	169	15:30	161	21:30	45
10:00	169	16:00	164	22:00	33
10:30	169	16:30	160	22:30	15
11:00	169	17:00	146	23:00	15
11:30	147	17:30	141	23:30	15

Tabela 2: Quantidade total de atendentes por intervalo de tempo.

Na Tabela 3, são apresentados os conjuntos de soluções antes de processar o método *ILS*. Na Tabela 4, são apresentados os conjuntos de soluções gerados na melhor iteração do *ILS*. As colunas representam:

- DEM : representa a demanda total de funcionários por intervalo de tempo;
- FUNC: representa a quantidade de funcionários que iniciaram o turno de trabalho no intervalo de tempo;
- ALOC: representa a quantidade de funcionários no intervalo de tempo;
- ESC : representa a escassez de funcionários no intervalo de tempo;
- EXC : representa o excesso de funcionários no intervalo de tempo.

O resultado da melhor iteração mostra que, após o refinamento da solução inicial, os resultados apresentados em cada coluna melhoraram consideravelmente a solução final, tanto na quantidade de funcionários (FUNC) necessários para o atendimento da demanda, quanto no excesso (EXC) e na escassez (ESC) de funcionários em cada intervalo de tempo.

Tabela 3: Resultado das iterações antes do método *ILS*

ANTES ILS											
HORÁRIO	DEM	FUNC	ALOC	ESC	EXC	HORÁRIO	DEM	FUNC	ALOC	ESC	EXC
06:00	15	15	15	0	0	15:00	161	16	161	0	0
06:30	22	7	22	0	0	15:30	161	0	161	0	0
07:00	38	16	38	0	0	16:00	164	3	164	0	0
07:30	62	24	62	0	0	16:30	160	0	164	0	4
08:00	118	56	118	0	0	17:00	146	0	164	0	18
08:30	157	39	157	0	0	17:30	141	0	164	0	23

09:00	169	12	169	0	0	18:00	121	0	164	0	43
09:30	169	0	169	0	0	18:30	99	0	164	0	65
10:00	169	0	169	0	0	19:00	83	0	164	0	81
10:30	169	0	169	0	0	19:30	77	0	149	0	72
11:00	169	0	169	0	0	20:00	57	0	74	0	17
11:30	147	0	169	0	22	20:30	57	0	19	38	0
12:00	126	0	154	0	28	21:00	49	0	3	46	0
12:30	112	0	147	0	35	21:30	45	0	3	42	0
13:00	112	0	131	0	19	22:00	33	0	0	33	0
13:30	122	15	122	0	0	22:30	15	0	0	15	0
14:00	141	75	141	0	0	23:00	15	0	0	15	0
14:30	157	55	157	0	0	23:30	15	0	0	15	0

Tabela 3: Resultado das iterações antes do método ILS

DEPOIS ILS											
HORÁRIO	**DEM**	**FUNC**	**ALOC**	**ESC**	**EXC**	**HORÁRIO**	**DEM**	**FUNC**	**ALOC**	**ESC**	**EXC**
06:00	15	15	15	0	0	15:00	161	0	144	17	0
06:30	22	7	22	0	0	15:30	161	0	144	17	0
07:00	38	16	38	0	0	16:00	164	3	147	17	0
07:30	62	24	62	0	0	16:30	160	0	147	13	0
08:00	118	56	118	0	0	17:00	146	0	147	0	1
08:30	157	39	157	0	0	17:30	141	0	147	0	6
09:00	169	12	169	0	0	18:00	121	15	162	0	41
09:30	169	0	169	0	0	18:30	99	0	162	0	63
10:00	169	0	169	0	0	19:00	83	0	162	0	79
10:30	169	0	169	0	0	19:30	77	0	146	0	69
11:00	169	0	169	0	0	20:00	57	0	72	0	15
11:30	147	0	169	0	22	20:30	57	0	18	39	0
12:00	126	0	154	0	28	21:00	49	0	18	31	0
12:30	112	0	147	0	35	21:30	45	0	18	27	0
13:00	112	0	131	0	19	22:00	33	0	15	18	0
13:30	122	16	123	0	1	22:30	15	0	15	0	0
14:00	141	74	141	0	0	23:00	15	0	15	0	0
14:30	157	54	156	1	0	23:30	15	0	15	0	0

Tabela 4: Resultado das Iterações após o método *ILS*

Na Tabela 5 são apresentados os resultados referente à comparação dos valores totais em cada coluna antes e após a execução do método *ILS*.

ANTES ILS					DEPOIS ILS				
DEM	**FUNC**	**ALOC**	**ESC**	**EXC**	**DEM**	**FUNC**	**ALOC**	**ESC**	**EXC**

3773	333	3996	204	427	3773	331	3972	180	379

Tabela 5: Comparação das Iterações antes e após a execução do método *ILS*

Na Tabela 6, são apresentados os resultados da função objetivo gerada na melhor iteração do método *ILS*. A variável ***distPertub*** representa a pertubação de uma posição para cima ou para baixo do vetor e a variável **p*ertub*** representa o nível de pertubação.

fo solucao gulosa = 1151.40	
distPertub = -1 pertub = 1	foMelhor = 1057.70
distPertub = 1 pertub = 1	foMelhor = 1055.80

Tabela 6: Comparação das Iterações antes e após a execução do método *ILS*

Observa-se que a heurística *ILS* produz soluções de qualidade para a otimização da troca de turnos. Após este resultado, basta alocar os atendentes para o trabalho, levando em consideração as preferências individuais de cada um.

7. Conclusões

Este trabalho apresentou o método *ILS* aplicado ao *Shift Design Problem*. O método heurístico *ILS* é uma técnica de busca local, retornando boas soluções. Para a geração da solução inicial foi utilizado o método da descida, também conhecido como método guloso, que analisa todos os possíveis vizinhos de uma solução, escolhendo a cada passo, aquele que tem o menor valor para a função objetivo. Foi desenvolvido um método para definir os níveis de perturbações para a geração da vizinhança em cada iteração. Os experimentos realizados neste trabalho, utilizando critérios de pertubações aleatórios, apresentaram os piores resultados obtidos através da função objetivo, em relação as pertubações definidas para a geração da vizinhança. O algoritmo *ILS* foi facilmente implementado e o tempo de processamento foi baixo, produzindo ótimos resultados. Isto mostra que é um método simples e de alta precisão e pode ser aplicado para diferentes classes de problemas.

Referências

Aykin, T. *Optimal shift scheduling with multiple break windows*, Managament Science, Vol. 42, n. 4, p. 591–602, 1996.

Aykin, T. *A comparative evaluation of modeling approaches to the labor shift scheduling problem*, European Journal of Operational Research, Vol.125, p. 381–397, 2000.

Bechtold, S.E. & Jacobs L.W. *Implicit modeling of flexible break assignments in optimal shift scheduling*, Management Science, Vol.36, n. 11, p. 1339–1351, 1990.

Bhulai, S., Koole, G & Pot, A.. *Simple methods for shift scheduling in multi-skill call centers*. Manufacturing & Service Operations Management, 2007.

Canon, C. *Personnel scheduling in the call center industry*. 4OR: A Quarterly Journal of Operations Research, Vol. 5, n. 1, p. 89–92, 2007.

Costa, F.P. *Programação de Horários em Escolas via GRASP e Busca Tabu*, UFOP, 2003.

Danzig, G.B. *A comment on eddie's traffic delays at toll booths*, Operations Research, Vol. 2, p. 339–341, 1954.

Glover, F & . McMillan, C. *The general employee scheduling problem*: *An integration of MS and AI*,

Computers and Operations Research, Vol. 13, n. 5, p. 563–573, 1986.

Glover, F. & Kochenberger, G. A.. *Handbook of Metaheuristics*, Kluwer Academic Publishers, Norwell, MA, 2003.

Lourenço, H. R., Martin, O.& Stützle, T. *Iterated Local Search*. *In* F.Glover & G. Kochenberger (eds), *Handbook of Metaheuristics*, p. 321 - 353, Kluwer Academic Publishers, Norwell, MA, 2003.

Moondra, L. . *An LP model for work force scheduling for banks*, Journal of Bank Research, Vol. 7, p. 299–301, 1976.

Musliu, N., Schaerf, A., & Slany, W. *Local search for shift design*. European Journal of Operational Research, Vol. 153, n.1, p.51–64, 2004.

Musliu, N., Beer, A., Schafhauser W., Gartner J. & Slany, W. *Scheduling Breaks in Shift Plans for Call Centers*, 2008.

Tellier, P. & White, G. *Generating personnel schedules in an industrial setting using a tabu search algorithm*. *In* E. K. Burke & H. Rudov (Eds.): PATAT 2006, p. 293–302, 2006.

Thompson, G. (1995). *Improved implicit modeling of the labor shift scheduling problem*, Management Science Vol. 41, n. 4, p. 595–607.