DEPARTAMENTO DE COMPUTAÇÃO

D E C O M

Resolução do problema de programação de tripulações de um sistema de transporte público via *Simulated Annealing*

Geraldo Regis Mauri – 99.1.4040

Marcone Jamilson Freitas Souza
Gustavo Peixoto Silva

Relatório Técnico 02/2003

U F O P

UNIVERSIDADE FEDERAL DE OURO PRETO

GERALDO REGIS MAURI

RESOLUÇÃO DO PROBLEMA DE PROGRAMAÇÃO DE TRIPULAÇÕES DE UM SISTEMA DE TRANSPORTE PÚBLICO VIA *SIMULATED ANNEALING*

MONOGRAFIA SUBMETIDA AO DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO DA UNIVERSIDADE FEDERAL DE OURO PRETO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Prof. Dr. Marcone Jamilson Freitas Souza
Doutor pela COPPE/Universidade Federal do Rio de Janeiro
Departamento de Ciência da Computação, UFOP

Prof. Dr. Luiz de Siqueira Martins Filho
Doutor pela Universidade Paul Sabatier – Toulouse (França)
Departamento de Ciência da Computação, UFOP

Msc. Marcelo Henrique Peixoto Caetano Chaves
Mestre pelo Instituto Nacional de Pesquisas Espaciais

OURO PRETO, MG – BRASIL
FEVEREIRO DE 2003

Resumo

Este trabalho tem seu enfoque no Problema da Programação da Tripulação (PPT) de uma empresa de transporte público urbano. Este problema consiste na atribuição da tarefa de condução dos veículos aos motoristas e cobradores (tripulações), de tal forma que as viagens das diferentes linhas atendidas pela empresa sejam executadas com o menor custo possível.

O número elevado de viagens e de suas possíveis combinações para constituir as jornadas de trabalho, associado com a necessidade de cumprimento da legislação trabalhista e das regras operacionais das empresas, torna altamente complexa a atividade de elaboração de escalas das tripulações. Na literatura o PPT é considerado NP-difícil, o que dificulta, ou até mesmo impossibilita a resolução de casos reais por métodos de programação matemática, dito exatos. Desta forma, o problema é normalmente abordado por técnicas heurísticas.

Nesse trabalho, o PPT foi tratado levando-se em consideração a metaheurística *Simulated Annealing* (SA), utilizando como movimentos de vizinhança a realocação de tarefas das tripulações (para a escala diária), e a troca de jornadas entre as tripulações (para a escala mensal).

A função objetivo utilizada pelo método visa a eliminação de inviabilidades, tais como a sobreposição de tarefas de uma mesma tripulação, o excesso de tempo trabalhado e outras, e ainda visa a redução dos custos operacionais, que consistem em minimizar o número de tripulações e utilizar adequadamente a mão-de-obra disponível.

O sistema computacional desenvolvido foi implementado utilizando-se a linguagem de programação C++, com a ferramenta C++ Builder 6.0.

Para melhor tratamento do problema, tentou-se fazer com que o sistema abordasse também uma boa solução para a interface com os usuários, fazendo assim com que o sistema além de eficiente fosse também funcional e prático em termos de usabilidade. Com o intuito de validar o sistema implementado, testou-se este com dados reais obtidos de uma empresa que opera no município de Belo Horizonte – MG.

Agradecimentos

A Deus, pela saúde e oportunidade de chegar até aqui.

À minha família, em especial à minha avó Tereza Rotta Mauri, pelo apoio e incentivo durante a realização do curso de Ciência da Computação.

Ao professor Marcone Jamilson Freitas Souza, pela orientação para a realização deste trabalho, e principalmente pelo companheirismo e amizade.

Aos professores Marcelo de Almeida Maia e Marcelo Luís Silva, pelo apoio e confiança.

A todos os professores do DECOM, em especial ao professor Tiago Garcia de Senna Carneiro, pelos ensinamentos transmitidos.

À Larice, pelo carinho, companheirismo e pelos momentos inesquecíveis em Ouro Preto.

Aos irmãos CANALHAS, pela amizade e por compartilharem todos os momentos durante a realização do curso de graduação.

À grandiosa e eterna República VIRA SAIA, por me ajudar a ser o que eu sou hoje, e por tudo que aprendi morando nela.

A todos os demais, que de alguma forma contribuíram para a realização deste trabalho.

Índice

RESUMO **3**

AGRADECIMENTOS **4**

ÍNDICE **5**

LISTA DE FIGURAS **7**

LISTA DE TABELAS **8**

1 INTRODUÇÃO **9**

1.1 OBJETIVOS DO TRABALHO 9
1.2 IMPORTÂNCIA DO TRABALHO 9
1.3 ESTRUTURA DO TRABALHO 9

2 REVISÃO BIBLIOGRÁFICA **10**

2.1 MÉTODOS UTILIZADOS 10
2.1.1 Busca local *10*
2.1.2 Método de descida *11*
2.1.3 Simulated Annealing (SA) *11*

3 METODOLOGIA **13**

3.1 DESCRIÇÃO DO PROBLEMA 13
3.1.1 Programação diária da tripulação *13*
3.1.2 Programação mensal da tripulação *14*
3.2 SA APLICADO À PROGRAMAÇÃO DIÁRIA DA TRIPULAÇÃO 14
3.2.1 A Função Objetivo *15*
3.2.2 Estrutura de Vizinhança Utilizada no Algoritmo *17*
3.2.3 Funcionamento básico do Algoritmo *17*
3.3 SA APLICADO À PROGRAMAÇÃO MENSAL DA TRIPULAÇÃO 18
3.3.1 A Função Objetivo *18*
3.3.2 Estrutura de Vizinhança Utilizada no Algoritmo *19*
3.3.3 Funcionamento básico do Algoritmo *21*

4 SISTEMA DESENVOLVIDO **22**

4.1 DESCRIÇÃO DO SISTEMA 22
4.2 SISTEMA APLICADO AO PPT 23
4.3 INTERAÇÃO SISTEMA X SA 30
4.4 RECURSOS OFERECIDOS PELO SISTEMA 31
4.4.1 Reaquecimento do SA *31*
4.4.2 Alteração de parâmetros *31*
4.4.3 Alteração dos valores padrão *36*
4.4.4 Iniciar, interromper e continuar a execução do SA *36*
4.4.5 Alterar manualmente as escalas diária e mensal *38*
4.4.6 Inserir e remover tripulações manualmente *39*
4.4.7 Agrupar linhas *39*
4.4.8 Selecionar um mês para a escala mensal *40*

4.5 TELAS DO SISTEMA 40
5 RESULTADOS OBTIDOS 45
6 CONCLUSÃO 49
6.1 ANÁLISE DOS RESULTADOS E CONCLUSÕES 49
6.1.1 Programação diária 49
6.1.2 Programação mensal 50
6.2 SUGESTÕES PARA TRABALHOS FUTUROS 51
7 REFERÊNCIAS BIBLIOGRÁFICAS 52
ANEXO A. CÓDIGO FONTE DO SISTEMA 53
ANEXO B. SAÍDAS DO SISTEMA 94

Lista de Figuras

Figura 2.1 - Algoritmo *Simulated Annealing*. 12
Figura 3.1 - Movimento Realocar tarefa. 17
Figura 3.2 - Agrupamentos de jornadas de dias úteis por semana. 20
Figura 3.3 - Movimento *Trocar jornada* 20
Figura 4.1 - Representação de uma tarefa. 23
Figura 4.2 - Representação de uma tarefa de tripulação. 24
Figura 4.3 - Representação de uma jornada. 24
Figura 4.4 - Representação de um dia. 25
Figura 4.5 - Representação de uma estrutura do mês 25
Figura 4.6 - Representação de uma tripulação diária. 26
Figura 4.7 - Representação de uma tripulação mensal. 27
Figura 4.8 - Representação de uma escala diária. 28
Figura 4.9 - Representação de uma escala mensal. 29
Figura 4.10 - Algoritmo utilizado no sistema. 30
Figura 4.11 - Tela de reaquecimento do SA. 31
Figura 4.12 - Tela de parâmetros do SA. 31
Figura 4.13 - Tela de parâmetros para a programação diária da tripulação. 32
Figura 4.14 - Tela de parâmetros para a programação mensal da tripulação 33
Figura 4.15 - Tela das penalizações para a programação diária. 34
Figura 4.16 - ela das penalizações para a programação mensal. 35
Figura 4.17 - Tela de alteração dos valores padrão. 36
Figura 4.18 - Tela de execução do SA e visualização da escala diária. 37
Figura 4.19 - Tela de execução do SA e visualização da escala mensal. 37
Figura 4.20 - Tela de alteração manual de escalas diária. 38
Figura 4.21 - Tela de alteração manual de escalas mensal. 39
Figura 4.22 - Tela para agrupar as linhas. 39
Figura 4.23 - Tela para seleção de um mês. 40
Figura 4.24 - Tela de entrada do arquivo da programação de veículos. 40
Figura 4.25 - Tela de entrada das escalas diárias para criação da mensal. 41
Figura 4.26 - Tela de visualização das tarefas de uma tripulação 41
Figura 4.27 - Tela de visualização das jornadas de uma tripulação. 42
Figura 4.28 - Tela de verificação do resultado obtido para a escala diária. 42
Figura 4.29 - Tela de verificação do resultado obtido para a escala mensal. 43
Figura 4.30 - Tela de alteração das cores do sistema. 43
Figura 4.31 - Tela principal do sistema. 44

Lista de Tabelas

Tabela 5.1 - Parâmetros do *S.A.* utilizados nos testes da programação diária. 45
Tabela 5.2 - Parâmetros utilizados nos testes da programação diária.....46
Tabela 5.3 - Pesos utilizados nos testes da programação diária.....46
Tabela 5.4 - Resultados obtidos nos testes da programação diária. 47
Tabela 5.5 - Parâmetros do S.A. utilizados nos testes da programação mensal. 47
Tabela 5.6 - Parâmetros utilizados nos testes da programação mensal. 48
Tabela 5.7 - Pesos utilizados nos testes da programação mensal. 48
Tabela 5.8 - Resultados obtidos nos testes da programação mensal.....48
Tabela 6.1 - Função objetivo dos testes realizados para a programação diária 49
Tabela 6.2 - Desempenho do método heurístico para a programação diária. 49
Tabela 6.3 - Comparação dos resultados obtidos para a programação diária.....49
Tabela 6.4 - Função objetivo dos testes realizados para a programação mensal. 50
Tabela 6.5 - Desempenho do método heurístico para a programação mensal. 50

1 Introdução

1.1 Objetivos do trabalho

Neste trabalho é proposta uma aplicação do método *Simulated Annealing* para solucionar o Problema da Programação da Tripulação (PPT) de uma empresa de transporte público urbano. Além disso, é proposta uma solução para a interface dessa aplicação com o intuito de torná-la utilizável no dia-a-dia de uma empresa de transporte público.

1.2 Importância do trabalho

O Problema da Programação da Tripulação (PPT) em sistemas de transporte público usando computadores teve seu início por volta de 1960 (Shen & Kwan 2000) e tem sido objeto de estudo desde então.

Apesar de o PPT ter sido largamente estudado e aplicado nos países mais desenvolvidos, suas técnicas de resolução são ainda pouco difundidas e raramente aplicadas à nossa realidade. Isso se deve às condições em que atuam as empresas de transporte público no Brasil e, principalmente, às restrições quanto ao cumprimento das legislações trabalhistas e às próprias normas impostas pelas empresas do setor, que impedem que um sistema desenvolvido em outro país possa ser aplicado à nossa realidade.

Com a crescente competitividade, as empresas, hoje em dia, devem buscam aprimorar cada vez mais seus processos produtivos, tentando manter a qualidade de seus produtos e/ou serviços e conseqüentemente reduzindo seus custos. Com esse intuito, elas devem procurar utilizar, de maneira eficiente, seus recursos materiais e humanos para se manterem lucrativas, sem, no entanto, comprometer a qualidade dos serviços oferecidos. É sabido que nas empresas de transporte público urbano a mão-de-obra operacional representa um dos maiores custos (Bouzada 2002). Assim, uma pequena redução neste item pode significar um ganho considerável no custo total, justificando qualquer trabalho no sentido de minimizar os custos com mão-de-obra.

Finalmente, a necessidade das empresas terem à sua disposição um sistema que solucione o PPT, que seja fácil de operar e que permita ao seu usuário uma maior interatividade, torna mais prática e menos trabalhosa a resolução do problema.

1.3 Estrutura do trabalho

O presente trabalho está dividido em seis capítulos, incluindo esta introdução.

No capítulo II faz-se uma breve revisão bibliográfica sobre as técnicas de resolução do PPT e descreve-se o método heurístico considerado neste trabalho para resolvê-lo.

O capítulo III descreve o PPT abordado, bem como o modelo heurístico de otimização desenvolvido para resolvê-lo.

No capítulo IV são mostrados detalhes do sistema computacional implementado.

O capítulo V mostra os resultados obtidos pela aplicação do sistema.

No capítulo VI o trabalho é concluído com a análise dos resultados obtidos e sugestões para trabalhos futuros.

2 Revisão Bibliográfica

Desde a década de 60, vários autores vêm se dedicando ao desenvolvimento de modelos capazes de gerar escalas de tripulações que satisfaçam às restrições impostas, e que possuam o menor custo possível. Este é um problema permanentemente estudado uma vez que as realidades dos sistemas de transporte estão em contínua transformação e exigem cada vez mais uma gerência eficiente dos recursos disponíveis. Prova disso é a existência de congressos especializados sobre o tema nas últimas décadas (Voβ & Daduna 2001, Wilson 1999, Daduna et al. 1995, Desrochers & Rousseau 1992).

Os sistemas heurísticos foram os primeiros a serem utilizados na resolução do PPT (Elias 1964). Tais sistemas consistiam na automação do trabalho antes realizado manualmente. Entretanto eles não eram capazes de detectar possibilidades de otimização. Manington e Wren (1975) iniciaram a inclusão de procedimentos de otimização neste tipo de sistema (Wren & Rousseau 1995).

Com o surgimento de metaheurísticas tais como *Algoritmos Genéticos* (Goldberg 1989), *Busca Tabu* (Glover 1989, Glover 1990), *Simulated Annealing* (Kirkpatrick et al., 1983) entre outras, abriu-se um novo horizonte na resolução de problemas *NP-difíceis* como o PPT. Embora tais métodos não garantam a obtenção do ótimo global, eles permitem incluir com facilidade qualquer tipo de restrição. Dentre os trabalhos metaheurísticos mais relevantes que envolvem a programação da tripulação destacamos: Clement & Wren (1995), Wren & Wren (1995), Kwan et al. (1999) e Shen & Kwan (2001).

Embora o PPT tenha sido largamente estudado e aplicado nos países mais desenvolvidos, suas técnicas de resolução são pouco difundidas e raramente aplicadas à nossa realidade. Isso se deve, em parte, pelo estágio primário em que se encontram as empresas do setor de transporte público no Brasil. Por outro lado, as principais restrições presentes neste problema estão relacionadas ao cumprimento das legislações trabalhistas e às normas operacionais vigentes nas empresas que atuam no sistema. Tais fatores impedem que um modelo desenvolvido no Reino Unido, por exemplo, seja aplicado no Brasil.

2.1 Métodos utilizados

As metaheurísticas são procedimentos destinados a encontrar uma boa solução, eventualmente a ótima, consistindo na aplicação, em cada passo, de uma heurística subordinada, a qual tem que ser modelada para cada problema específico. As metaheurísticas utilizam buscas locais para obtenção das soluções, e contrariamente às heurísticas convencionais podem escapar de ótimos locais.

2.1.1 Busca local

Buscas locais em problemas de otimização são baseadas na noção de vizinhança. Para definirmos o que é uma vizinhança, seja S o espaço de pesquisa de um problema de otimização e f a função objetivo a minimizar. O conjunto $N(s) \subseteq S$, o qual depende da estrutura do problema tratado, reúne um número determinado de soluções s', denominado *vizinhança* de s. Cada solução $s' \in N(s)$ é chamada de *vizinho* de s e é obtido de s a partir de uma operação chamada de *movimento*.

Em linhas gerais, uma busca local, começando de uma solução inicial s_0, navega pelo espaço de pesquisa, através do movimento, passando de uma solução para outra que seja sua vizinha.

2.1.2 Método de descida

É um método de busca local que se caracteriza por analisar todos os possíveis vizinhos de uma solução s em sua vizinhança N(s), escolhendo, a cada passo, aquele que tem o menor valor para a função objetivo. Nesse método, o vizinho candidato somente é aceito se ele melhorar estritamente o valor da melhor solução até então obtida. Dessa forma, o método pára tão logo um mínimo local seja encontrado.

2.1.3 *Simulated Annealing* (SA)

SA é um método de busca local que aceita movimento de piora para escapar de ótimos locais. Ele foi proposto originalmente por (Kirkpatrick et al., 1983), e se fundamenta em uma analogia com a termodinâmica, ao simular o resfriamento de um conjunto de átomos aquecidos.

Esta técnica começa sua busca a partir de uma solução inicial qualquer. O procedimento principal consiste em um *loop* que gera aleatoriamente, em cada iteração, um único vizinho *s'* da solução corrente *s*.

A cada geração de um vizinho *s'* de *s*, é testada a variação Δ do valor da função objetivo, isto é, $\Delta = f(s') - f(s)$. Se $\Delta < 0$, o método aceita a solução e *s'* passa a ser a nova solução corrente. Caso $\Delta \geq 0$ a solução vizinha candidata também poderá ser aceita, mas neste caso, com uma probabilidade $e^{-\Delta/T}$, onde *T* é um parâmetro do método, chamado de *temperatura* e que regula a probabilidade de aceitação de soluções com custo pior.

A temperatura *T* assume, inicialmente, um valor elevado T_0. Após um número fixo de iterações (o qual representa o número de iterações necessárias para o sistema atingir o equilíbrio térmico em uma dada temperatura), a temperatura é gradativamente diminuída por uma razão de resfriamento α, tal que $T_n \leftarrow \alpha * T_{n-1}$, sendo $0 < \alpha < 1$. Com esse procedimento, dá-se, no início uma chance maior para escapar de mínimos locais e, à medida que *T* aproxima-se de zero, o algoritmo comporta-se como o método de descida, uma vez que diminui a probabilidade de se aceitar movimentos de piora ($T \rightarrow 0 \Rightarrow e^{-\Delta/T} \rightarrow 0$).

O procedimento pára quando a temperatura chega a um valor próximo de zero e nenhuma solução que piore o valor da melhor solução é mais aceita, isto é, quando o sistema está estável. A solução obtida quando o sistema encontra-se nesta situação evidencia o encontro de um mínimo local.

Os parâmetros de controle do procedimento são a razão de resfriamento α, o número de iterações para cada temperatura (*SAmax*) e a temperatura inicial T_0. A Figura 2.1 apresenta o algoritmo *Simulated Annealing* básico.

```
    Procedimento S.A. (f(.), N(.), α, SAmax, T0, s)
1.  s* ← s;                  {Melhor solução obtida até então}
2.  IterT ← 0;               {Número de iterações na temperatura T}
3.  T ← T0;                  {Temperatura corrente}
4.  enquanto (T > 0) faça
5.     enquanto (IterT < SAmax) faça
6.        IterT ← IterT + 1;
7.        Gere um vizinho qualquer s' ∈ N(s);
8.        Δ = f(s') – f(s);
9.        se (Δ < 0)
10.           então s ← s';
11.                 se (f(s') < f(s*)) então s* ← s';
12.           senão Tome x ∈ [0,1];
13.                 se (x < e-Δ/T) então s ← s';
14.       fim-se;
15.    fim-enquanto;
16.    T ← α * T;
17.    IterT ← 0;
18. fim-enquanto;
19. s ← s*;
20. Retorne s;
    Fim S.A.:
```

Figura 2.1 - Algoritmo *Simulated Annealing*.

3 Metodologia

3.1 Descrição do problema

O problema da programação da tripulação consiste na atribuição da tarefa de condução dos veículos aos motoristas e cobradores (tripulações), de tal forma que as viagens das diferentes linhas atendidas pela empresa sejam executadas com o menor custo possível. Este problema pode ser dividido em dois "sub-problemas", o da programação diária e o da programação mensal da tripulação.

3.1.1 Programação diária da tripulação

No transporte público, usualmente a programação diária da tripulação é feita após a programação dos veículos. Nesta as viagens são reunidas em *Blocos*. Um bloco apresenta a seqüência de viagens que um determinado veículo tem que realizar em um dia, começando e terminando na garagem. Cada bloco mostra também as *Oportunidades de Troca* (OT). A OT é um intervalo de tempo suficiente para haver a troca das tripulações. A partir do bloco de um veículo são criadas as *Tarefas*. Cada tarefa é um conjunto de viagens reunidas de maneira que haja apenas duas OT: uma no início e outra no final da tarefa. Assim, durante sua realização não é possível que haja troca de tripulação.

A programação diária de uma tripulação é formada por um conjunto de tarefas, chamado de *Jornada diária*. As jornadas diárias são divididas em dois tipos: *Pegada Simples* ou *Dupla Pegada*. No primeiro tipo as tarefas são realizadas de uma única vez e os intervalos de tempo entre as tarefas são iguais ou menores que 2 horas. Caso ocorra um intervalo maior do que duas horas a jornada é classificada como dupla pegada. Este intervalo não é contabilizado na remuneração da tripulação.

A programação diária pode ser classificada como:

- *Para dias úteis*: as tarefas são formadas a partir das viagens de uma programação de veículos para dias úteis;
- *Para sábados*: as tarefas são formadas a partir das viagens de uma programação de veículos para sábados;
- *Para domingos e feriados*: as tarefas são formadas a partir das viagens de uma programação de veículos para domingos e feriados.

Nos dois últimos casos, as jornadas diárias são, obrigatoriamente, do tipo *Pegada Simples*.

Ao se reunir as tarefas formando as jornadas diárias deve-se levar em conta inúmeras restrições operacionais e trabalhistas, tais como:

a) A troca da tripulação só pode ocorrer em pontos onde houver OT;

b) A jornada diária de trabalho é de 6:40 horas, podendo ser acrescida de no máximo 2:00 horas extras;

c) Uma tripulação tem direito a 30 minutos de descanso durante sua jornada diária de trabalho (*Intervalo total para repouso e/ou alimentação*), podendo este período ser fracionado em intervalos menores, desde que um deles seja maior ou igual a 15

minutos (*Sub-intervalo contínuo para repouso e/ou alimentação*). Esses 30 minutos não são computados na jornada de trabalho;

d) Numa jornada com dupla pegada, os 30 minutos citados acima não são considerados.

3.1.2 Programação mensal da tripulação

Já a programação mensal da tripulação é feita após a programação diária. Nessa as jornadas diárias são reunidas em *Jornadas mensais*. Uma jornada mensal apresenta uma seqüência de jornadas diárias distribuídas ao longo de um determinnado mês. A partir desse mês, determinam-se quais são os dias úteis, os sábados, os domingos e os feriados, tornando-se possível a distribuição correta das jornadas diárias para esses dias. Outro fator que deve ser levado em conta são os dias de folga da tripulação.

Na programação mensal da tripulação, as jornadas diárias terão algumas características próprias de acordo com o tipo do dia que está sendo programado. As jornadas de sábados, domingos e feriados, em menor número, só poderão ser do tipo *Pegada Simples* de acordo com as restrições operacionais da empresa de transporte público em questão.

Visando facilitar a abordagem do problema da programação mensal da tripulação, foi usada como estratégia a replicação de jornadas ao longo do mês, ou seja, de acordo com o tipo do dia, as jornadas dentro do mês seriam iguais. Dessa forma, a estrutura usada para a realização da distribuição de jornadas aos tripulantes seria um *Agrupamento* de jornadas.

Ao se reunir as jornadas diárias formando as jornadas mensais deve-se levar em conta principalmente, além das restrições consideradas na programação diária, as seguintes restrições:

a) O intervalo de tempo entre jornadas que deve ser igual ou superior a 11 horas;

b) O número de trocas de tipos de pegadas entre jornadas, que deve ser reduzido;

c) Deve haver uma uniformidade na construção da escala mensal das tripulações de forma que exista um baixo desvio das horas trabalhadas no mês, por uma tripulação específica, em relação à média das horas totais trabalhadas pelas tripulações.

Obs: As restrições contidas tanto na programação diária quanto na mensal constam na Convenção Coletiva de Trabalho 2002/2003 celebrada entre o Sindicato das Empresas de Transporte de Passageiros de Belo Horizonte (SETRABH) e o Sindicato dos Trabalhadores em Transporte Rodoviário de Belo Horizonte (STTRBH).

Restrições deste tipo tornam problemas reais intratáveis computacionalmente, justificando a utilização de metaheurísticas na resolução dos mesmos.

3.2 SA aplicado à programação diária da tripulação

Como foi dito anteriormente a técnica SA começa a partir de uma solução inicial, caminha por uma estrutura de vizinhança minimizando a função objetivo. Nesta seção

descreveremos como esta técnica foi empregada para solucionar o problema da programação diária da tripulação.

3.2.1 A Função Objetivo

A função a ser minimizada, que avalia a programação diária da tripulação, tem os seguintes requisitos:

i) **Requisitos essenciais –** Esta parte da função objetivo procura eliminar completamente todas as inviabilidades da programação diária da tripulação. Essas inviabilidades são descritas pelos seguintes requisitos:

(a) *Sobreposição de Tarefas*: uma tripulação não pode realizar mais de uma tarefa ao mesmo tempo;

(b) *Troca de Pontos Proibida*: uma tripulação não pode realizar duas tarefas consecutivas na qual o ponto do final da primeira seja diferente do ponto do início da segunda, e o intervalo entre elas seja menor do que o tempo necessário para ir de um ponto a outro;

(c) *Troca de Linhas Proibida*: uma tripulação não pode realizar duas tarefas consecutivas na qual a linha final da primeira seja diferente da linha inicial da segunda, e essas linhas pertençam a diferentes grupos;

(d) *Horas Excedentes*: por determinações trabalhistas uma tripulação não pode exercer mais que 9:10 horas de atividades diárias;

(e) *Folga Acumulada Deficiente*: durante a realização de uma jornada de trabalho a tripulação tem direito a uma folga de 30 minutos destinada à refeição. Por acordo feito pelos trabalhadores, esta folga pode ser dividida e distribuída no decorrer do dia, conquanto que uma das frações seja de pelo menos 15 minutos.

(f) *Tempo entre jornadas*: o tempo que falta para completar 11:00h entre o horário de fim e o horário de início de uma jornada.

A função que melhor representa os requisitos essenciais é a seguinte:

$$f_D(s) = \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_i B_{ij} \tag{1}$$

em que
- s: solução corrente;
- $f_D(s)$: função que avalia o não atendimento dos requisitos essenciais da solução s;
- n: número de tripulações;
- m: número de tipos de penalidades;
- α_i: peso que reflete a importância do requisito i;
- B_{ij}: quantificação do requisito i na programação da tripulação j.

ii) **Requisitos não essenciais –** Esta segunda parte da função objetivo tem por finalidade minimizar os custos operacionais e aproveitar ao máximo o tempo de cada tripulação. Esses custos são descritos pelos seguintes requisitos:

(a) *Duplas Pegadas*: o algoritmo procura reduzir o número de tripulações que têm jornadas do tipo dupla pegada;

(b) *Horas Extras*: o número de horas que superam 7:10 horas (6:40 + 30min) e não ultrapassam 9:10 horas diárias é considerado hora extra. Este montante é minimizado;

(c) *Troca de Pontos Permitida*: uma troca de pontos é permitida quando uma tripulação realiza duas tarefas consecutivas, cujo ponto de troca do final da primeira é diferente do ponto de início da segunda, e o tempo para ir de um para o outro é menor que o intervalo entre elas. Apesar de permitida é desejável que não haja muitas trocas desse tipo;

(d) *Troca de Linhas Permitida*: uma troca de linhas é permitida quando uma tripulação realiza duas tarefas consecutivas cuja linha final da primeira é diferente da linha inicial da segunda, e essas linhas pertençam ao mesmo grupo;

(e) *Troca de Veículos*: quando uma tripulação realiza uma tarefa em um veículo e, após o término desta, troca de veículo e realiza uma outra tarefa.

(f) *Número de Tripulantes*: o número total de tripulações deve ser minimizado até o ponto que o número de horas extras não seja excessivamente grande.

(g) *Ociosidade:* a soma dos tempos que uma tripulação deixa de trabalhar nas 7:10 horas diárias, descontado o intervalo de 30 minutos para refeição.

A função que melhor representa os requisitos não essenciais é a seguinte:

$$g_D(s) = \sum_{i=1}^{p} \sum_{j=1}^{n} \beta_i C_{ij} + n \times peso1 + ndp \times peso2 \qquad (2)$$

em que

s: solução corrente;

$g_D(s)$: função que avalia o não atendimento dos requisitos não essenciais da solução *s*;

n: número de tripulações;

ndp: número de tripulações com dupla pegada;

p: número de requisitos não-essenciais;

β_i: peso que reflete a importância do requisito *i*;

C_{ij}: valor do requisito *i* na programação da tripulação *j*;

peso1: peso que reflete a importância do número de tripulações;

peso2: peso que reflete a importância do nº de tripulações com dupla pegada.

A fórmula representativa da função objetivo total é a seguinte:

$$FO_D(s) = f_D(s) + g_D(s) \qquad (3)$$

em que

s: solução corrente;

$f_D(s)$: função que avalia o não atendimento dos requisitos essenciais da solução s;

$g_D(s)$: função que avalia o não atendimento dos requisitos não essenciais da solução s;

3.2.2 Estrutura de Vizinhança Utilizada no Algoritmo

O *Simulated Annealing* pode ser convenientemente caracterizado como uma forma de busca em uma vizinhança, onde cada solução $s \in S$ tem um conjunto associado de vizinhos, $N(s) \subset S$ chamado de vizinhança de *s*. Qualquer solução $s' \in N(s)$ pode ser alcançada diretamente de *s* a partir de um *movimento.*

O algoritmo trabalha com o movimento *Realocar Tarefa*. Este movimento consiste em atribuir uma tarefa de uma tripulação *i* a uma tripulação *j*. Para gerar esse movimento são sorteadas duas tripulações quaisquer, *i* e *j*, com a condição que a tripulação *i* tenha pelo menos uma tarefa. A seguir, sorteia-se uma tarefa da tripulação *i*, sobre a qual faz-se o movimento. Na Figura 3.1 o movimento é explicado detalhadamente. Em (a) temos as duas tripulações antes do movimento formando uma solução *s*. Em (b) ocorre o movimento. Nesta figura mostra-se que a tarefa escolhida passa da tripulação *i* para a *j* e que, portanto, as ligações anteriores entre as tarefas deixam de existir (linhas tracejadas da figura). Já em (c) podemos ver as tripulações *i* e *j* constituindo a nova solução *s'* vizinha de *s*.

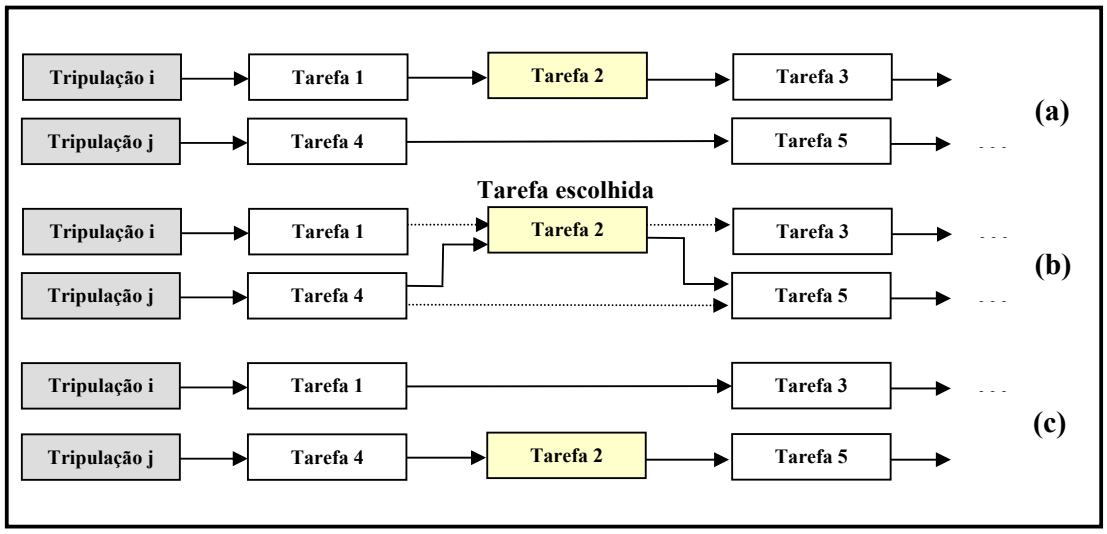


Figura 3.1 - Movimento Realocar tarefa.

3.2.3 Funcionamento básico do Algoritmo

Passo1 - *Selecionar a programação de veículos adequada*.

Selecionar uma programação de veículos referente a dias úteis ou sábados ou domingos ou feriados.

Passo2 - *Gerar as tarefas*.

Tomando como base a programação dos veículos selecionada, o algoritmo particiona os blocos nas OT's, agrupando as viagens em tarefas.

Passo3 - *Obtenção da solução inicial*.

Tomando como base um número máximo de tripulações indicado pelo usuário, a eles são distribuídas, aleatoriamente, as tarefas que foram obtidas no passo anterior.

Passo4 - *Minimizar a função objetivo usando o método Simulated Annealing.*

A técnica do *Simulated Annealing* é usada para minimizar a função objetivo, eliminando as inviabilidades operacionais, minimizando os custos da programação e maximizando a utilização da mão-de-obra, ou seja, é usada para tentar atender os requisitos essenciais e não essenciais, fazendo com que o valor da função objetivo seja o menor possível.

3.3 SA aplicado à programação mensal da tripulação

Como foi dito anteriormente a técnica SA começa a partir de uma solução inicial, caminha por uma estrutura de vizinhança minimizando a função objetivo. Nesta seção descreveremos como esta técnica foi empregada para solucionar o problema da programação mensal da tripulação.

3.3.1 A Função Objetivo

A função a ser minimizada, que avalia a programação mensal da tripulação, tem os seguintes requisitos:

i) **Requisitos essenciais –** Esta parte da função objetivo procura eliminar completamente todas as inviabilidades da programação mensal da tripulação. Como a maioria desses requisitos já foi atendida na programação diária, resta apenas uma inviabilidade que deve ser reavaliada na programação mensal. Essa inviabilidade é descrita pelo seguinte requisito:

(a) *Tempo entre jornadas*: o tempo que falta para completar 11:00h entre o horário de fim e o horário de início de uma jornada diária.

A função que melhor representa os requisitos essenciais é a seguinte:

$$f_M(s) = \sum_{i=1}^{n} \alpha \; B_i \tag{4}$$

em que
- s: solução corrente;
- $f_M(s)$: função que avalia o não atendimento dos requisitos essenciais da solução s;
- n: número de tripulações;
- α: peso que reflete a importância desse requisito;
- B_i: quantificação desse requisito na programação da tripulação i.

ii) **Requisitos não essenciais –** Esta segunda parte da função objetivo tem por finalidade minimizar os custos operacionais e aproveitar ao máximo o tempo de cada tripulação. Além disso, esses requisitos visam também a uniformização das *jornadas mensais* das tripulações. Esses custos são descritos pelos seguintes requisitos:

(a) *N° de trocas de tipo de pegadas*: é o número de vezes que uma tripulação troca de uma jornada diária do tipo *Pegada Simples* para *Pegada Dupla* ou vice-versa;

(b) *N° de trocas do período de trabalho*: considerando um horário que indica o horário máximo de início de um período de trabalho, todas as jornadas diárias cujo início seja antes desse horário estão em um período de trabalho, as outras que iniciam após esse horário, estão em outro período de trabalho. Assim, o número de vezes que isso acontece é contabilizado, visando uniformizar os períodos de trabalho das tripulações;

(c) *N° de jornadas diferentes*: é o número de jornadas diárias diferentes contidas na jornada mensal da tripulação;

(d) *Diferença do tempo médio de trabalho*: é a diferença entre o tempo de trabalho da tripulação e o tempo médio de trabalho de todas as tripulações.

A função que melhor representa os requisitos não essenciais é a seguinte:

$$g_M(s) = \sum_{i=1}^{p} \sum_{j=1}^{n} \beta_i C_{ij} \tag{5}$$

em que

s: solução corrente;

$g_M(s)$: função que avalia o não atendimento dos requisitos não essenciais da solução s;

n: número de tripulações;

p: número de requisitos não-essenciais;

β_i: peso que reflete a importância do requisito i;

C_{ij}: quantificação do requisito i na programação da tripulação j;

A fórmula representativa da função objetivo total é a seguinte:

$$FO_M(s) = f_M(s) + g_M(s) \tag{6}$$

em que

s: solução corrente;

$f_M(s)$: função que avalia o não atendimento dos requisitos essenciais da solução s;

$g_M(s)$: função que avalia o não atendimento dos requisitos não essenciais da solução s;

3.3.2 Estrutura de Vizinhança Utilizada no Algoritmo

Para simplificar o tratamento do problema da Programação Mensal, as jornadas de dias úteis foram agrupadas por semana (vide Fig. 3.2). Por exemplo, se um mês, sem feriados, começa em uma quarta-feira, então, os dias de quarta, quinta e sexta pertencerão a um mesmo agrupamento, o qual receberá uma mesma jornada diária. Nesse exemplo o próximo agrupamento envolveria os dias compreendidos entre a segunda e a sexta-feira da semana subseqüente.

Aqui, na programação mensal, o algoritmo trabalha com o movimento *Trocar Jornada (ou agrupamento)*, este movimento consiste em trocar uma jornada diária ou agrupamento de jornadas diárias de uma tripulação i com a jornada ou agrupamento correspondente de uma

tripulação *j*. Para gerar esse movimento são sorteadas duas tripulações quaisquer, *i* e *j*. A seguir, sorteia-se uma jornada ou agrupamento de uma das tripulações *i* ou *j*, sobre o qual, e seu respectivo na outra tripulação, faz-se o movimento. Deve-se garantir que tanto a jornada (ou agrupamento) selecionada quanto a sua respectiva não sejam *folgas*.

Na Figura 3.3 o movimento é explicado detalhadamente. Em (a) temos as duas tripulações antes do movimento formando uma solução *s*. Em (b) ocorre o movimento, nesta figura mostra-se que o agrupamento de jornadas diárias escolhido passa da tripulação *i* para a *j* e o respectivo em j passa para i. Já em (c) podemos ver as tripulações *i* e *j* constituindo a nova solução *s'* vizinha de *s*.

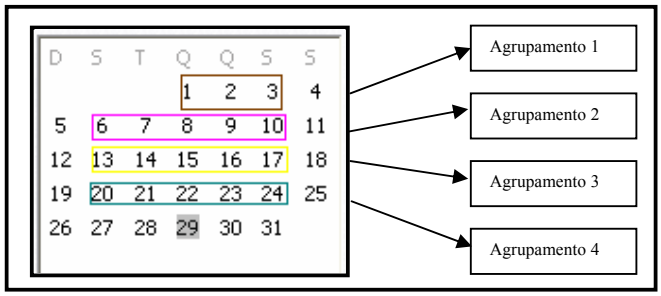


Figura 3.2 - Agrupamentos de jornadas de dias úteis por semana.

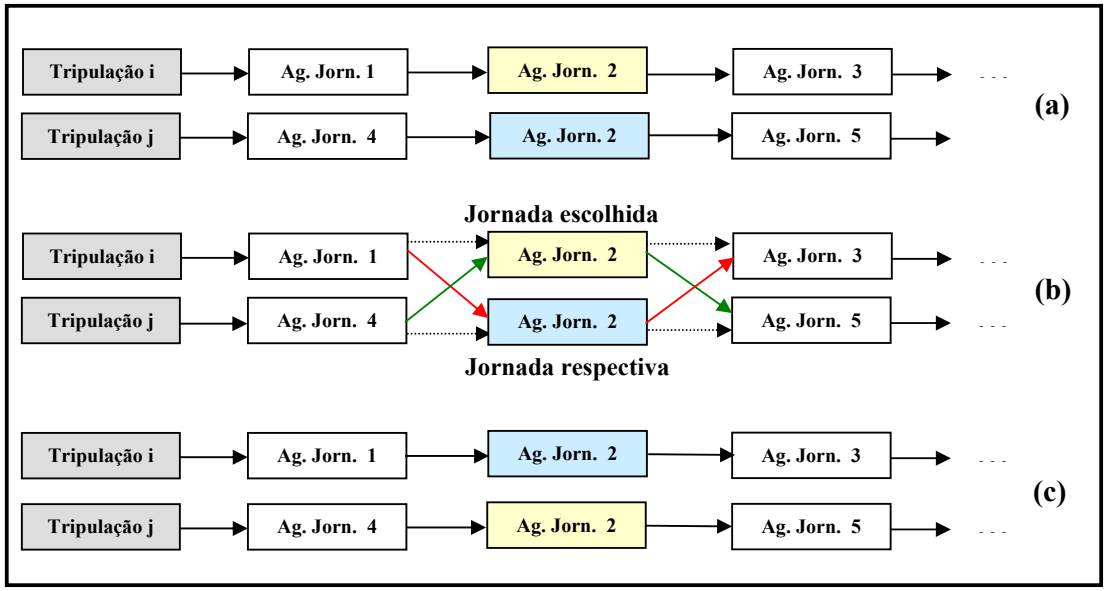


Figura 3.3 - Movimento *Trocar jornada*.

3.3.3 Funcionamento básico do Algoritmo

Passo1 – *Ler as programações diárias*.

Tomando como base uma programação diária para dias úteis, uma para sábados e uma para domingos e feriados, o algoritmo "carrega" as jornadas diárias.

Passo2 – *Seleção do mês*.

Tomando como base um mês qualquer do calendário, definem-se os dias úteis, os sábados, os domingos e os feriados.

Passo3 - *Obtenção da solução inicial*.

Considerando u o número de jornadas diárias para dias úteis, s o número de jornadas diárias para sábados e f para domingos e feriados, e ainda a soma $u = s + f$, são criadas u tripulações para as quais são atribuídas seqüencialmente as u jornadas diárias para dias úteis a todos os dias úteis dessas tripulações. Feito isso, são atribuídas, também seqüencialmente, as s jornadas diárias para sábados a todos os sábados das tripulações (da tripulação 0 até a s-1) e *folga para os domingos e feriados*, e por fim, da mesma forma, distribui-se as jornadas diárias para domingos e feriados para todos os domingos e feriados (da trip. s até a u-1) e folga para os sábados.

Passo4 - *Minimizar a função objetivo usando o método Simulated Annealing*.

A técnica do *Simulated Annealing* é usada para minimizar a função objetivo, eliminando as inviabilidades operacionais, minimizando os custos da programação mensal e uniformizando as jornadas mensais, ou seja, é usada para tentar atender os requisitos essenciais e não essenciais, fazendo com que o valor da função objetivo seja o menor possível.

4 Sistema desenvolvido

4.1 Descrição do sistema

O sistema desenvolvido, chamado "PPT Solver", tem o intuito de tentar solucionar o PPT, e ao mesmo tempo ser utilizado por qualquer empresa de transportes. Ele foi desenvolvido na linguagem C++ com a ferramenta C++ Builder 6.0. Isso devido ao poder e desempenho da linguagem e a disponibilidade e recursos da ferramenta. O mesmo foi implementado segundo o paradigma de Orientação a Objetos.

O sistema tenta oferecer um ambiente de uso agradável, flexível e aplicável no dia-a-dia de uma empresa de transportes. Para isso, o sistema foi desenvolvido com uma interface bem parecida com a de "softwares comuns", de maneira que não necessite de um usuário "especial" para operá-lo. Ele permite ainda que o usuário faça alterações de valores que mudam freqüentemente como, por exemplo, os valores determinados pela Convenção Coletiva de Trabalho, que mudam, praticamente, todo ano.

Esse sistema está dividido em dois módulos independentes, descritos como:

- *Diário*: Módulo onde são obtidas as escalas diárias, ou seja, a programação diária das tripulações. Como entrada para esse módulo utiliza-se a programação de veículos.
- *Mensal*: Módulo onde são obtidas as escalas mensais, ou seja, a programação mensal das tripulações. Já nesse módulo, as entradas são três escalas diárias (dias úteis, sábados e domingos e feriados), ou seja, saídas do módulo anterior.

O funcionamento básico do módulo diário pode ser melhor descrito em quatro etapas:

(1) *Parâmetros da programação diária da tripulação*: Nessa etapa o usuário preenche os parâmetros referentes à programação diária como, por exemplo, o tempo para troca das mesmas, o número de tripulações, etc.

(2) *Programação de veículos*: Terminada a etapa anterior, o usuário seleciona um arquivo que contém a programação de veículos referente ao tipo de programação a ser gerado (dias úteis, sábados ou domingos e feriados).

(3) *Agrupamento de linhas*: Aqui, o usuário determina os grupos nos quais as linhas pertencem.

(4) *Execução do SA*: Nessa etapa o usuário executa o *Simulated Annealing*.

O funcionamento básico do módulo mensal também pode ser descrito em quatro etapas:

(1) *Parâmetros da programação mensal da tripulação*: Nessa etapa o usuário preenche os parâmetros referentes à programação mensal como, por exemplo, o tempo mínimo entre as jornadas diárias, etc.

(2) *Escalas diárias*: Aqui, o usuário seleciona os três arquivos que contém as escalas diárias para dias úteis, sábados e domingos e feriados.

(3) *Seleção do mês*: A partir de um calendário, o usuário seleciona o mês e o ano para o qual será construída a escala mensal.

(4) *Execução do SA*: Nessa etapa o usuário executa o *Simulated Annealing*.

Obs: Tanto no caso do módulo diário quanto mensal, em qualquer momento antes da etapa 3, o usuário pode alterar os valores dos parâmetros do SA e das penalizações. E, terminada a execução do SA, será exibido o resultado encontrado, no qual o usuário poderá ver a escala (diária ou mensal) encontrada.

4.2 Sistema aplicado ao PPT

Visando modelar da melhor forma possível o sistema em relação ao PPT, a implementação do mesmo é fundamentada basicamente em nove entidades: Tarefas, Tarefas das tripulações, Jornadas, Dias, Estrutura do mês, Tripulações diária e mensal e Escalas diária e mensal.

i) **Tarefa:** Essa entidade tem a finalidade de representar as tarefas das tripulações diárias. Essa representação é feita a partir de uma classe chamada *TTarefa* cujos atributos representam os valores que compõem uma tarefa. Essa representação é feita segundo o modelo descrito na Figura 4.1.

Tarefa	
N° do veículo	Folga acumulada
Horário inicial	Horário final
Ponto inicial	Ponto final
Linha inicial	Linha final

Figura 4.1 - Representação de uma tarefa.

onde:

(a) *Número do veículo*: número do veículo que realiza a tarefa;

(b) *Folga acumulada*: intervalos obtidos durante a realização da tarefa;

(c) *Horário inicial*: horário de início da tarefa;

(d) *Horário final*: horário de término da tarefa;

(e) *Ponto inicial*: ponto no qual o veículo inicia a tarefa;

(f) *Ponto final*: ponto no qual o veículo termina a tarefa;

(g) *Linha inicial*: linha na qual o veículo inicia a tarefa;

(h) *Linha final*: linha na qual o veículo termina a tarefa.

ii) **Tarefa de tripulação:** Essa entidade tem a finalidade de representar as tarefas de tripulações específicas. Essa representação é feita a partir de uma classe chamada *TTar_Trip* cujos atributos representam os valores destas tarefas. Essa representação é feita segundo o modelo descrito na Figura 4.2.

Tarefa de tripulação	
N° do veículo	Folga acumulada
Horário inicial	Horário final
Ponto inicial	Ponto final
Linha inicial	Linha final
N° da tripulação	N° da tarefa

Figura 4.2 - Representação de uma tarefa de tripulação.

onde:

(a) *N° da tripulação*: número que representa a tripulação "dona" da tarefa;

(b) *N° da tarefa*: número que representa a tarefa;

Os demais atributos são semelhantes às tarefas citadas anteriormente.

iii) **Jornada:** Essa entidade tem a finalidade de representar as jornadas diárias das tripulações mensais. Essa representação é feita a partir de uma classe chamada *TJornada* cujos atributos representam os valores que compõem uma jornada diária. Essa representação é feita segundo o modelo descrito na Figura 4.3.

Jornada	
Tipo	Tempo de trabalho
Horário inicial	Horário final
Ponto inicial	Ponto final
Linha inicial	Linha final
Hora extra	Ociosidade

Figura 4.3 - Representação de uma jornada.

onde:

(a) *Tipo*: indica se a jornada diária é do tipo *Pegada Simples* ou *Dupla Pegada*;

(b) *Tempo de trabalho*: tempo trabalhado na jornada diária;

(c) *Horário inicial*: horário de início da jornada diária;

(d) *Horário final*: horário de término da jornada diária;

(e) *Ponto inicial*: ponto no qual o veículo inicia a jornada diária;

(f) *Ponto final*: ponto no qual o veículo termina a jornada diária;

(g) *Linha inicial*: linha na qual o veículo inicia a jornada diária;

(h) *Linha final*: linha na qual o veículo termina a jornada diária;

(i) *Hora extra*: hora extra contida na jornada diária;

(j) *Ociosidade*: tempo ocioso contido na jornada diária.

iv) **Dia:** Essa entidade tem a finalidade de representar os "dias" da estrutura do mês. Essa representação é feita a partir de uma classe chamada *TDias* cujos atributos representam os valores que compõem um "agrupamento de jornadas diárias" na estrutura. Essa representação é feita segundo o modelo descrito na Figura 4.4.

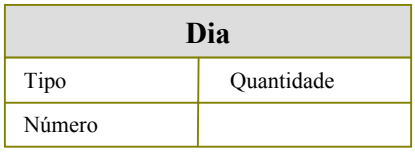


Figura 4.4 - Representação de um dia.

onde:

(c) *Tipo*: tipo do dia (U → dia útil; S → sábado; D → domingo ou feriado.);

(d) *Quantidade*: quantidade de dias consecutivos;

(e) *Número*: número referente a um agrupamento de jornadas diárias na estrutura.

v) **Estrutura do mês:** Essa entidade tem a finalidade de representar a estrutura a partir da qual a escala mensal será construída. Essa representação é feita a partir de uma classe chamada *TEstruturaMes* cujos atributos representam os valores que compõem esta estrutura. Essa representação é feita segundo o modelo descrito na Figura 4.5.

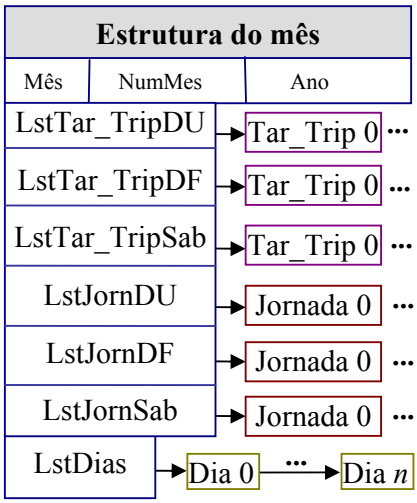


Figura 4.5 - Representação de uma estrutura do mês.

onde:

(a) *Mês*: nome do mês selecionado (Janeiro, Fevereiro, etc);

(b) *NumMes*: número do mês selecionado (1, 2, ..., 12);

(c) LstTar_TripDU: lista de tarefas de tripulação para dias úteis;

(d) LstTar_TripDF: lista de tarefas de tripulação para domingos e feriados;

(e) LstTar_TripSab: lista de tarefas de tripulação para sábados;

(f) LstJornDU: lista de jornadas para dias úteis;

(g) LstJornDF: lista de jornadas para domingos e feriados;

(h) LstJornSab: lista de jornadas para sábados;

(i) LstDias: lista com os dias referentes ao mês selecionado;

vi) **Tripulação diária:** Essa entidade tem a finalidade de representar as tripulações diárias. Essa representação é feita a partir de uma classe chamada *TTripED* cujos atributos representam os valores que compõem uma tripulação da escala diária. Essa representação é feita segundo o modelo descrito na Figura 4.6.

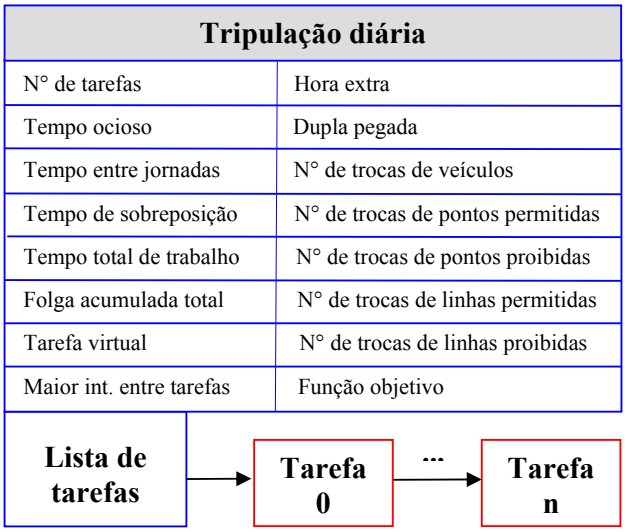


Figura 4.6 - Representação de uma tripulação diária.

onde:

(a) *Número de tarefas*: é a quantidade de tarefas da tripulação;

(b) *Hora extra*: tempo superior a 7:10 horas, caso não haja dupla pegada, ou superior a 6:40 caso haja;

(c) *Tempo ocioso*: tempo total entre as tarefas, mais a folga acumulada total e mais o que tempo que faltar para completar as 7:10 horas (caso não seja dupla pegada) ou 6:40horas menos o tempo da dupla pegada (caso haja dupla pegada);

(d) *Dupla pegada*: caso a jornada possua um intervalo maior do que 2 horas entre duas tarefas, ela é classificada como *dupla pegada*;

(e) *Tempo entre jornadas*: tempo entre o horário de fim e o horário de início de uma jornada diária;

(f) *Número de trocas de veículos*: número de vezes em que a tripulação troca de veículos entre duas tarefas;

(g) *Tempo de sobreposição*: tempo total em que uma tripulação realiza mais de uma tarefa ao mesmo tempo;

(h) *Número de trocas de pontos permitidas*: número de vezes em que o ponto final de uma tarefa é diferente do ponto inicial da tarefa seguinte e o intervalo entre essas tarefas é o tempo da dupla pegada (caso haja);

(i) *Tempo total de trabalho*: tempo total trabalhado menos o tempo da dupla pegada (caso haja);

(j) *Número de trocas de pontos proibidas*: número de vezes em que o ponto final de uma tarefa é diferente do ponto inicial da tarefa seguinte e o intervalo entre essas tarefas não é o tempo da dupla pegada;

(k) *Folga acumulada total*: folga acumulada de todas as tarefas da tripulação;

(l) *Número de trocas de linhas permitidas*: número de vezes em que a linha final de uma tarefa é diferente da linha inicial da tarefa seguinte e essas linhas pertencem a diferentes grupos;

(m) *Tarefa virtual*: tempo necessário para contemplar o intervalo e o sub-intervalo para repouso e/ou alimentação;

(n) *Número de trocas de linhas proibidas*: número de vezes em que a linha final de uma tarefa é diferente da linha inicial da tarefa seguinte e essas linhas pertencem ao mesmo grupo;

(o) *Maior int. entre tarefas*: valor do maior intervalo entre as tarefas.

(p) *Função objetivo*: função que avalia a jornada diária da tripulação;

(q) *Lista de tarefas*: é uma lista que contém as tarefas da tripulação. Esta lista foi implementada com uma lista duplamente encadeada que contém objetos do tipo *TTarefa*. Essa lista está representada em vermelho na Figura 4.6.

vii) **Tripulação mensal:** Essa entidade tem a finalidade de representar as tripulações mensais. Essa representação é feita a partir de uma classe chamada *TTripEM* cujos atributos representam os valores que compõem uma tripulação da escala mensal. Essa representação é feita segundo o modelo descrito na Figura 4.7.

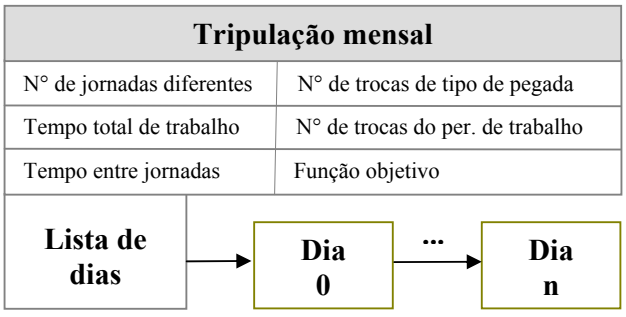


Figura 4.7 - Representação de uma tripulação mensal.

onde:

(a) *N° de jornadas diferentes*: é o número de jornadas diferentes dessa tripulação;

(b) *N° de trocas de tipo de pegada*: é o número de vezes em que essa tripulação passa de uma jornada de Pegada Simples para uma de Dupla Pegada ou vice-versa;

(c) *Tempo total de trabalho*: tempo total trabalhado;

(d) *N° de trocas do período de trabalho*: n° de vezes em que a tripulação troca o seu período de trabalho;

(e) *Tempo entre jornadas*: tempo entre o horário de fim e o horário de início de uma jornada diária;

(f) *Função objetivo*: função que avalia a jornada mensal da tripulação;

(f) *Lista de dias*: é uma lista que contém os dias que referenciam as jornadas diárias da tripulação. Esta lista foi implementada com uma lista duplamente encadeada que contém objetos do tipo *TDias*.

viii) **Escala diária:** Essa entidade tem a finalidade de representar as escalas diárias com suas respectivas tripulações e tarefas. Essa representação é feita a partir de uma classe chamada *TEscDiaria*. Essa representação é feita segundo o modelo descrito na Figura 4.8.

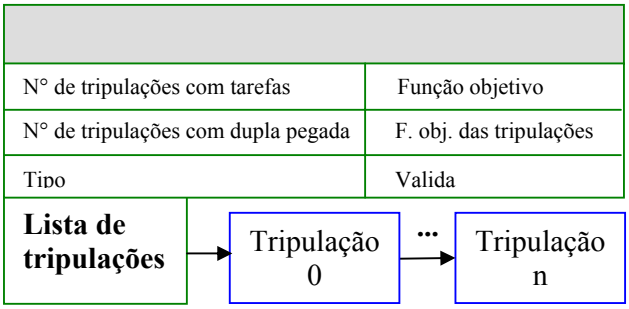


Figura 4.8 - Representação de uma escala diária.

onde:

(a) *N° de tripulações com tarefas*: número de tripulações que possuem pelo menos uma tarefa;

(b) *Função objetivo*: função que avalia escala diária da programação da tripulação;

(c) *N° de tripulações com dupla pegada*: número de tripulações cuja jornada é do tipo dupla pegada;

(d) *F*. obj. das tripulações: somatório das funções objetivos das tripulações;

(e) *Tipo*: indica o tipo da escala diária (dia útil, sábado ou domingo e feriado);

(f) *Valida*: indica se a escala diária é válida ou não;

(g) *Lista de tripulações*: é uma lista que contém as tripulações e suas respectivas tarefas. Esta lista também foi implementada com uma lista duplamente encadeada que contém objetos do tipo *TTripED*. Essa lista está representada em azul na Figura 4.8.

ix) **Escala mensal:** Essa entidade tem a finalidade de representar as escalas mensais com suas respectivas tripulações e jornadas mensais. Essa representação é feita a partir de uma classe chamada *TEscMensal*. Essa representação é feita segundo o modelo descrito na Figura 4.9.

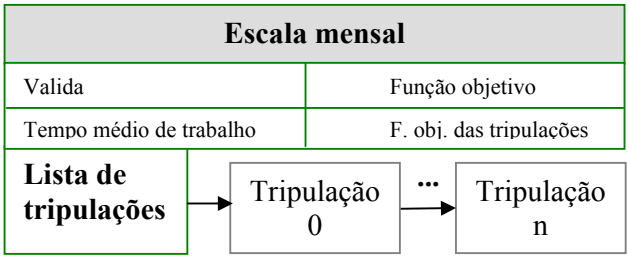


Figura 4.9 - Representação de uma escala mensal.

onde:

(a) *Valida*: indica se a escala diária é válida ou não;

(b) *Função objetivo*: função que avalia escala mensal da programação da tripulação;

(c) *Tempo médio de trabalho*: tempo (mensal) médio trabalhado pelas tripualções;

(d) *F*. obj. das tripulações: somatório das funções objetivos das tripulações;

(e) *Lista de tripulações*: é uma lista que contém as tripulações e seus respectivos dias. Esta lista também foi implementada com uma lista duplamente encadeada que contém objetos do tipo *TTripEM*. Essa lista está representada em cinza na Figura 4.9.

Obs: As tripulações possuem uma lista de "dias" e não de jornadas diárias, isso devido ao grande número de réplicas das jornadas diárias que seriam necessárias caso essas fossem alocadas diretamente às tripulações. Então, da maneira como o sistema foi modelado, a estrutura do mês possui as listas de jornadas diárias, e as tripulações possuem listas de dias, os quais referenciam as jornadas diárias nas listas da estrutura do mês.

4.3 Interação Sistema x SA

No sistema, o algoritmo do SA sofre algumas modificações em relação ao básico, citado na Figura 3.1. Uma dessas é possibilitar que o algoritmo execute por um tempo determinado pelo usuário, e a principal é a interrupção da execução do algoritmo a qualquer momento.

Assim, o algoritmo utilizado no sistema está descrito na Figura 4.10.

```
    Procedimento S.A. (f(.), N(.), α, SAmax, T0, s, TMP, TC)
1.  s* ← s;                    {Melhor solução obtida até então}
2.  IterT ← 0;                 {Número de iterações na temperatura T}
3.  T ← T0;                    {Temperatura corrente}
4.  TempoTermino ← TempoCorrente + TMP;
5.  enquanto (T > TC) ou (TempoTermino < TempoCorrente) faça
6.      enquanto (IterT < SAmax) faça
7.          Ler(Parar);
8.          se (Parar = true)
9.            então Retorne s;
10.         IterT ← IterT + 1;
11.         Gere um vizinho qualquer s' ∈ N(s);
12.         Δ = f(s') – f(s);
13.         se (Δ < 0)
14.           então s ← s';
15.                 se (f(s') < f(s*)) então s* ← s';
16.           senão Tome x ∈ [0,1];
17.                 se (x < e-Δ/T) então s ← s';
18.         fim-se;
19.     fim-enquanto;
20.     T ← α * T;
21.     IterT ← 0;
22.  fim-se
23. fim-enquanto;
24. s ← s*;
25. Retorne s;
    Fim S.A.;
```

Figura 4.10 - Algoritmo utilizado no sistema.

Principais alterações:

- *Linha 4*: O tempo de término do SA recebe o tempo corrente mais o tempo máximo de processamento (*TMP*);
- *Linha 5*: Verifica se o sistema congelou ($T < TC$) ou se o tempo máximo de processamento já foi atingido (*TempoTermino* < *TempoCorrente*). *TC* é a temperatura de congelamento, *TempoTermino* é o momento em que o algoritmo pára de executar e *TempoCorrente* é o momento corrente do sistema;
- *Linha 8*: Verifica se o usuário interrompeu a execução do SA;
- *Linha 9*: O algoritmo retorna a melhor solução encontrada até o momento.

4.4 Recursos oferecidos pelo sistema

4.4.1 Reaquecimento do SA

Caso o critério de parada Temperatura de Congelamento não seja selecionado, no momento em que o SA congelar (Temperatura atual < Temperatura de congelamento) o sistema dará a oportunidade do usuário reaquecer o SA, ou seja, elevar a temperatura atual e continuando a execução. Esse recurso é disponível na tela representada pela Figura 4.11.

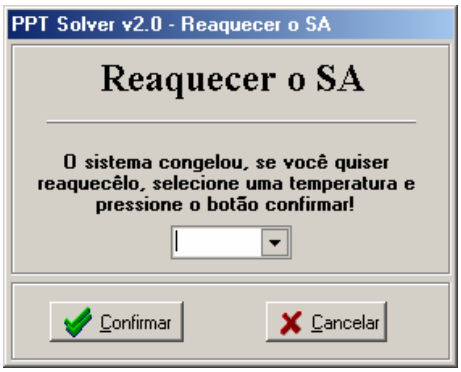


Figura 4.11 - Tela de reaquecimento do SA.

4.4.2 Alteração de parâmetros

O sistema permite ao usuário alterar os valores do SA, para a programação diária e mensal das tripulações e das penalizações (para as programações diária e mensal). Isso é feito através das telas representadas pelas Figuras 4.12, 4.13, 4.14, 4.15 e 4.16 respectivamente.

Parâmetros do SA:

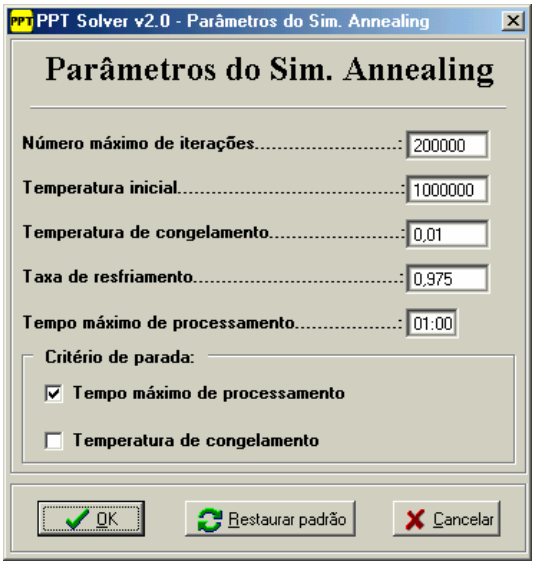


Figura 4.12 - Tela de parâmetros do SA.

(a) *N° máximo de iterações*: número de iterações que o SA vai executar em cada temperatura. Na Figura 4.10 esse número é representado por *SAmax*. Esse valor é um número inteiro maior do que zero.

(b) *Temperatura inicial*: temperatura na qual o SA vai iniciar a execução. Na Figura 4.10 essa temperatura é representada por T_0. Esse valor é um número real maior do que zero.

(c) *Temperatura de congelamento*: temperatura limite do SA, ou seja, quando a temperatura corrente for menor do que a temperatura de congelamento significa que o sistema congelou, terminando assim a execução do SA. Na Figura 4.10 essa temperatura é representada por *TC*.

(d) *Taxa de resfriamento*: taxa que será aplicada (multiplicada) à temperatura inicial enquanto o sistema não congelar. Na Figura 4.10 essa taxa é representada por α. Esse valor é um número real variando de zero a um;

(e) *Tempo máximo de processamento*: tempo pelo qual o SA será executado. Na Figura 4.10 esse tempo é representado por *TMP*.

(f) *Critério de parada*: indica qual critério o usuário quer utilizar para determinar o fim da execução do SA.

Clicando no botão "OK" da tela representada pela Figura 4.12, os valores para a execução do algoritmo do SA serão alterados para os valores apresentados na tela.

Clicando no botão "Restaurar padrão" da tela representada pela Figura 4.12, os campos serão preenchidos com os valores padrão.

Clicando no botão "Cancelar" da tela representada pela Figura 4.12, as alterações realizadas pelo usuário não terão efeito sobre a execução do SA, e a tela será fechada.

Programação da tripulação:

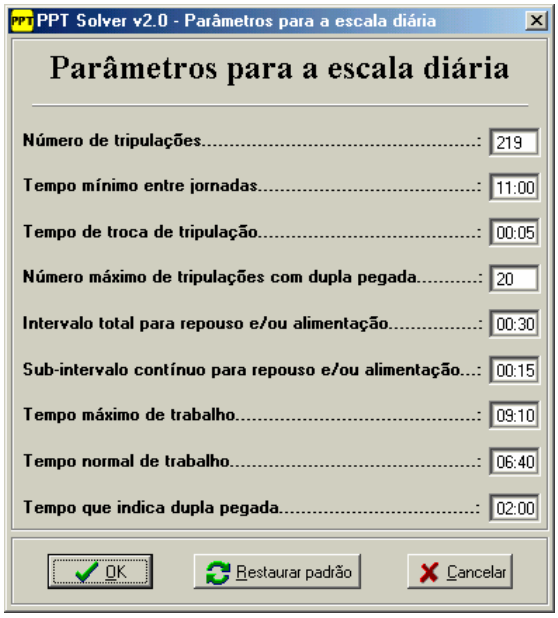


Figura 4.13 - Tela de parâmetros para a programação diária da tripulação.

(a) *Número de tripulações*: número de tripulações utilizadas na construção da escala. Esse valor é um número inteiro;

(b) *Tempo mínimo entre jornadas*: tempo mínimo necessário entre duas jornadas diárias. Esse valor é um número inteiro (hh:mm);

(c) *Tempo de troca de tripulação*: tempo necessário para realizar uma troca de tripulação. Esse valor é um número inteiro (hh:mm);

(d) *Número máximo de tripulações com dupla pegada*: número máximo de tripulações com dupla pegada. Esse valor é um número inteiro.

(e) *Intervalo total para repouso e/ou alimentação*: tempo mínimo que a tripulação deve ter para repouso e/ou alimentação. Esse valor é um número inteiro (hh:mm);

(f) *Sub*-intervalo contínuo para repouso e/ou alimentação: tempo contínuo mínimo que a tripulação deve ter para repouso e/ou alimentação. Esse valor é um número inteiro (hh:mm);

(g) *Tempo máximo de trabalho*: tempo máximo trabalhado diariamente. Esse valor é um número inteiro (hh:mm);

(h) *Tempo normal de trabalho*: tempo normal de trabalho em um dia. Esse valor é um número inteiro (hh:mm);

(i) *Tempo que indica dupla pegada*: intervalo de tempo que indica se uma jornada diária é do tipo simples ou dupla. Esse valor é um número inteiro (hh:mm);

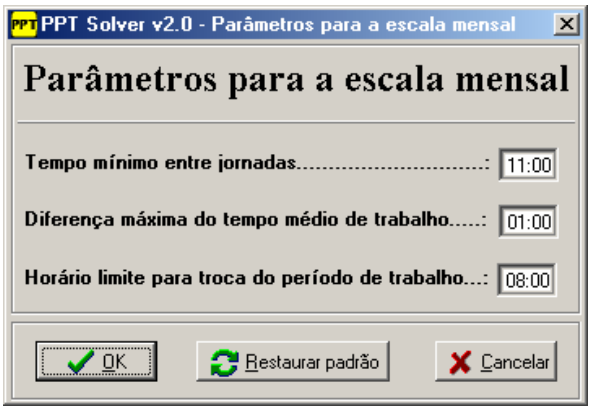


Figura 4.14 - Tela de parâmetros para a programação mensal da tripulação.

(a) *Tempo mínimo entre jornadas*: tempo mínimo necessário entre duas jornadas diárias. Esse valor é um número inteiro (hh:mm);

(b) *Diferença máxima do tempo médio de trabalho*: é a diferença máxima do tempo médio de trabalho das tripulações. Esse valor é um número inteiro (hh:mm);

(c) *Horário limite para troca do período de trabalho*: horário através do qual se determina em qual período de trabalho a jornada diária pertence. Esse valor é um número inteiro (hh:mm);

Clicando no botão “OK” da tela representada pela Figura 4.13 ou 4.14, os valores dos parâmetros serão alterados para os valores apresentados na tela.

Clicando no botão “Restaurar padrão” da tela representada pela Figura 4.13 ou 4.14, os campos serão preenchidos com os valores padrão.

Clicando no botão “Cancelar” da tela representada pela Figura 4.13 ou 4.14, as alterações realizadas pelo usuário não terão efeito sobre os parâmetros, e a tela será fechada.

Penalizações:

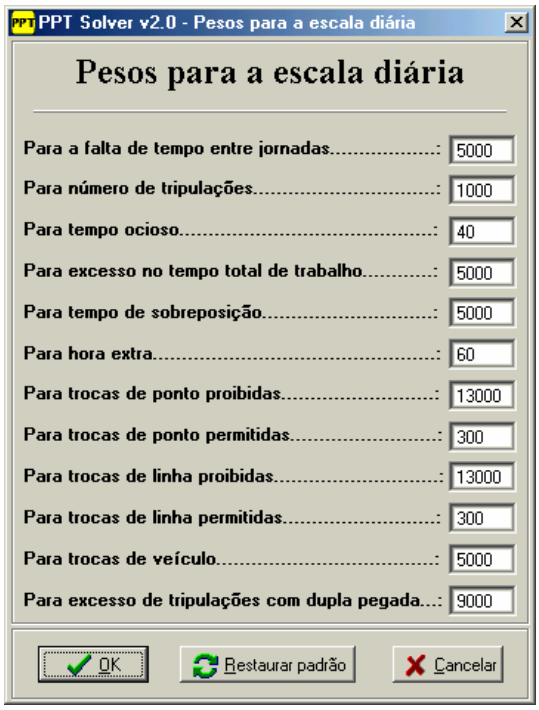


Figura 4.15 - Tela das penalizações para a programação diária.

(a) *Peso para a falta de tempo entre jornadas*: valor utilizado para penalizar a falta de tempo entre as jornadas das tripulações;

(b) *Peso para o número de tripulações*: valor utilizado para penalizar o número de tripulações;

(c) *Peso para o tempo ocioso*: valor utilizado para penalizar o tempo ocioso das tripulações;

(d) *Peso para o excesso de tempo total de trabalho*: valor utilizado para penalizar o excesso no tempo total de trabalho das tripulações;

(e) *Peso para o tempo de sobreposição*: valor utilizado para penalizar o tempo de sobreposição das tripulações;

(f) *Peso para as horas extras*: valor utilizado para penalizar as horas extras das tripulações;

(g) *Peso para o número de trocas de pontos proibidas*: valor utilizado para penalizar o número de trocas de ponto proibidas das tripulações;

(h) *Peso para o número de trocas de pontos permitidas*: valor utilizado para penalizar o número de trocas de ponto permitidas das tripulações;

(i) *Peso para o número de trocas de linhas proibidas*: valor utilizado para penalizar o número de trocas de linha proibidas das tripulações;

(j) *Peso para o número de trocas de linhas permitidas*: valor utilizado para penalizar o número de trocas de linha permitidas das tripulações;

(k) *Peso para o número de trocas de veículos*: valor utilizado para penalizar o número de trocas de veículos das tripulações;

(l) *Peso para o excesso de tripulações com dupla pegada*: valor utilizado para penalizar o excesso de tripulações com dupla pegada;

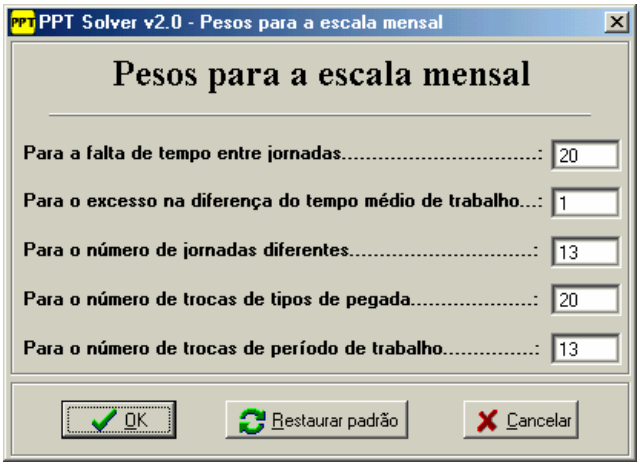


Figura 4.16 - ela das penalizações para a programação mensal.

(a) *Peso para a falta de tempo entre jornadas*: valor utilizado para penalizar a falta de tempo entre as jornadas diárias das tripulações;

(b) *Peso para o excesso na diferença do tempo médio de trabalho*: valor utilizado para penalizar o excesso na diferença do tempo médio de trabalho das tripulações;

(c) *Peso para o número de jornadas diferentes*: valor utilizado para penalizar o número de jornadas diferentes das tripulações;

(d) *Peso para o número de trocas de tipos de pegada*: valor utilizado para penalizar o número de trocas de tipos de pegada das tripulações;

(e) *Peso para o número de trocas do período de trabalho*: valor utilizado para penalizar o número de trocas do período de trabalho das tripulações.

Clicando no botão “OK” da tela representada pela Figura 4.15 ou 4.16, os valores para as penalizações serão alterados para os valores apresentados na tela.

Clicando no botão “Restaurar padrão” da tela representada pela Figura 4.15 ou 4.16, os campos serão preenchidos com os valores padrão.

Clicando no botão “Cancelar” da tela representada pela Figura 4.15 ou 4.16, as alterações realizadas pelo usuário não terão efeito sobre as penalizações, e a tela será fechada.

Obs: Caso o usuário queira reduzir algum valor do resultado (n° de tripulações com dupla pegada, por exemplo), basta-o aumentar o valor referente ao peso para o excesso de tripulações com dupla pegada.

4.4.3 Alteração dos valores padrão

O sistema permite ao usuário alterar os valores dos padrões do SA, da programação diária e mensal das tripulações e das penalizações (diária e mensal). Assim, ao clicar nos botões "Restaurar padrão" em uma das figuras citadas na seção anterior, os valores salvos como padrão, na tela representada pela Figura 4.17, serão carregados nos campos da respectiva tela. Além disso, na tela representada pela Figura 4.17, o usuário pode restaurar os valores para uma configuração default, cujos valores não podem ser alterados pelo usuário.

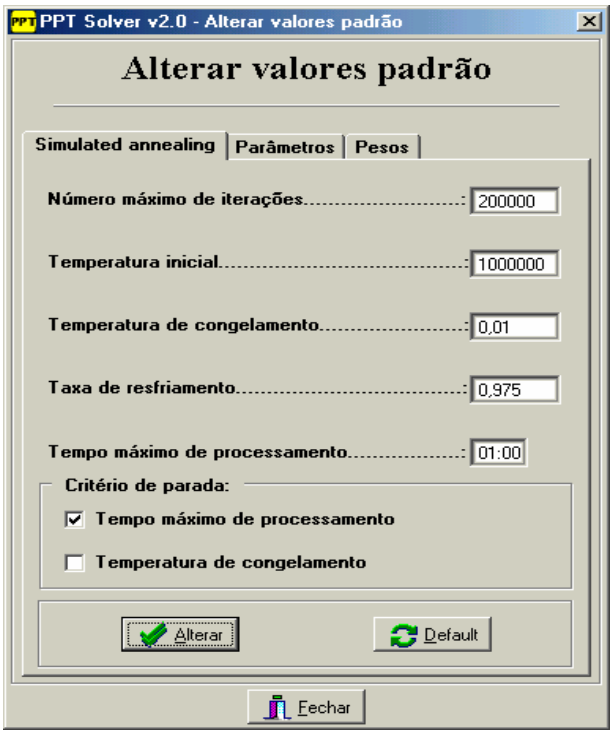
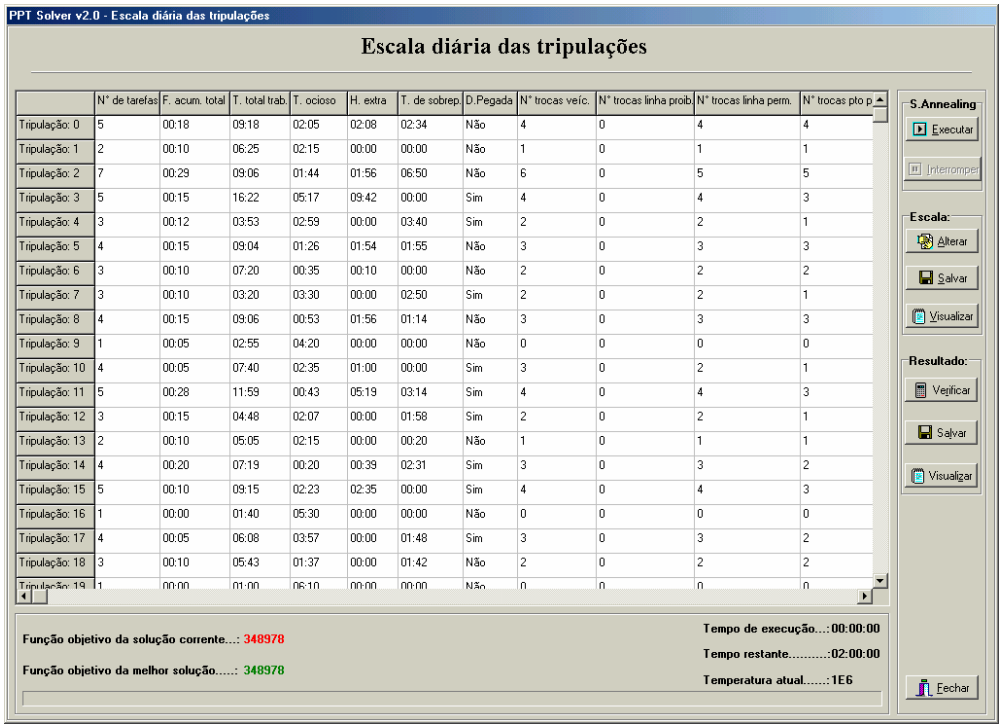


Figura 4.17 - Tela de alteração dos valores padrão.

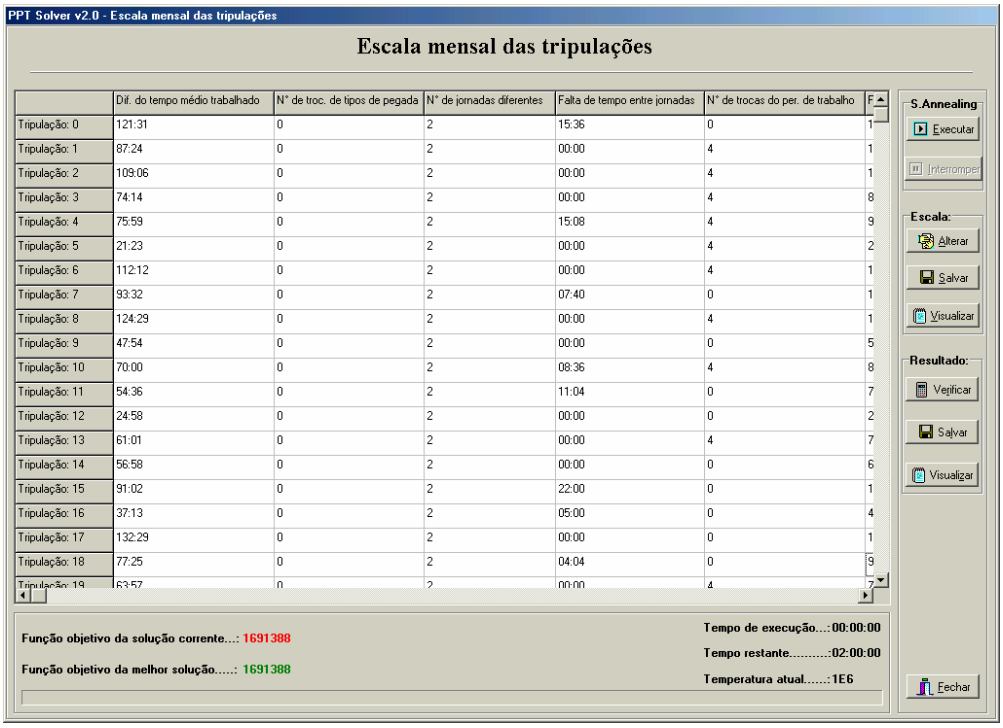
4.4.4 Iniciar, interromper e continuar a execução do SA

Na tela representada pelas Figuras 4.18 e 4.19, o usuário pode iniciar, interromper ou continuar a execução do SA. Além disso, ele pode ver as escalas (diária ou mensal) encontradas pelo SA, os resultados e outros.



	N° de tarefas	F. acum. total	T. total trab.	T. ocioso	H. extra	T. de sobrep.	D.Pegada	N° trocas veíc.	N° trocas linha proib.	N° trocas linha perm.	N° trocas pto p
Tripulação: 0	5	00:18	09:18	02:05	02:08	02:34	Não	4	0	4	4
Tripulação: 1	2	00:10	06:25	02:15	00:00	00:00	Não	1	0	1	1
Tripulação: 2	7	00:29	09:06	01:44	01:56	06:50	Não	6	0	5	5
Tripulação: 3	5	00:15	16:22	05:17	09:42	00:00	Sim	4	0	4	3
Tripulação: 4	3	00:12	03:53	02:59	00:00	03:40	Sim	2	0	2	1
Tripulação: 5	4	00:15	09:04	01:26	01:54	01:55	Não	3	0	3	3
Tripulação: 6	3	00:10	07:20	00:35	00:10	00:00	Não	2	0	2	2
Tripulação: 7	3	00:10	03:20	03:30	00:00	02:50	Sim	2	0	2	1
Tripulação: 8	4	00:15	09:06	00:53	01:56	01:14	Não	3	0	3	3
Tripulação: 9	1	00:05	02:55	04:20	00:00	00:00	Não	0	0	0	0
Tripulação: 10	4	00:05	07:40	02:35	01:00	00:00	Sim	3	0	2	1
Tripulação: 11	5	00:28	11:59	00:43	05:19	03:14	Sim	4	0	4	3
Tripulação: 12	3	00:15	04:48	02:07	00:00	01:58	Sim	2	0	2	1
Tripulação: 13	2	00:10	05:05	02:15	00:00	00:20	Não	1	0	1	1
Tripulação: 14	4	00:20	07:19	00:20	00:39	02:31	Sim	3	0	3	2
Tripulação: 15	5	00:10	09:15	02:23	02:35	00:00	Sim	4	0	4	3
Tripulação: 16	1	00:00	01:40	05:30	00:00	00:00	Não	0	0	0	0
Tripulação: 17	4	00:05	06:08	03:57	00:00	01:48	Sim	3	0	3	2
Tripulação: 18	3	00:10	05:43	01:37	00:00	01:42	Não	2	0	2	2
Tripulação: 19	1	00:00	01:00	06:10	00:00	00:00	Não	0	0	0	0

Figura 4.18 - Tela de execução do SA e visualização da escala diária.



	Dif. do tempo médio trabalhado	N° de troc. de tipos de pegada	N° de jornadas diferentes	Falta de tempo entre jornadas	N° de trocas do per. de trabalho	F
Tripulação: 0	121:31	0	2	15:36	0	1
Tripulação: 1	87:24	0	2	00:00	4	1
Tripulação: 2	109:06	0	2	00:00	4	1
Tripulação: 3	74:14	0	2	00:00	4	8
Tripulação: 4	75:59	0	2	15:08	4	9
Tripulação: 5	21:23	0	2	00:00	4	2
Tripulação: 6	112:12	0	2	00:00	4	1
Tripulação: 7	93:32	0	2	07:40	0	1
Tripulação: 8	124:29	0	2	00:00	4	1
Tripulação: 9	47:54	0	2	00:00	0	5
Tripulação: 10	70:00	0	2	08:36	4	8
Tripulação: 11	54:36	0	2	11:04	0	7
Tripulação: 12	24:58	0	2	00:00	0	2
Tripulação: 13	61:01	0	2	00:00	4	7
Tripulação: 14	56:58	0	2	00:00	0	6
Tripulação: 15	91:02	0	2	22:00	0	1
Tripulação: 16	37:13	0	2	05:00	0	4
Tripulação: 17	132:29	0	2	00:00	0	1
Tripulação: 18	77:25	0	2	04:04	0	9
Tripulação: 19	63:57	0	2	00:00	4	7

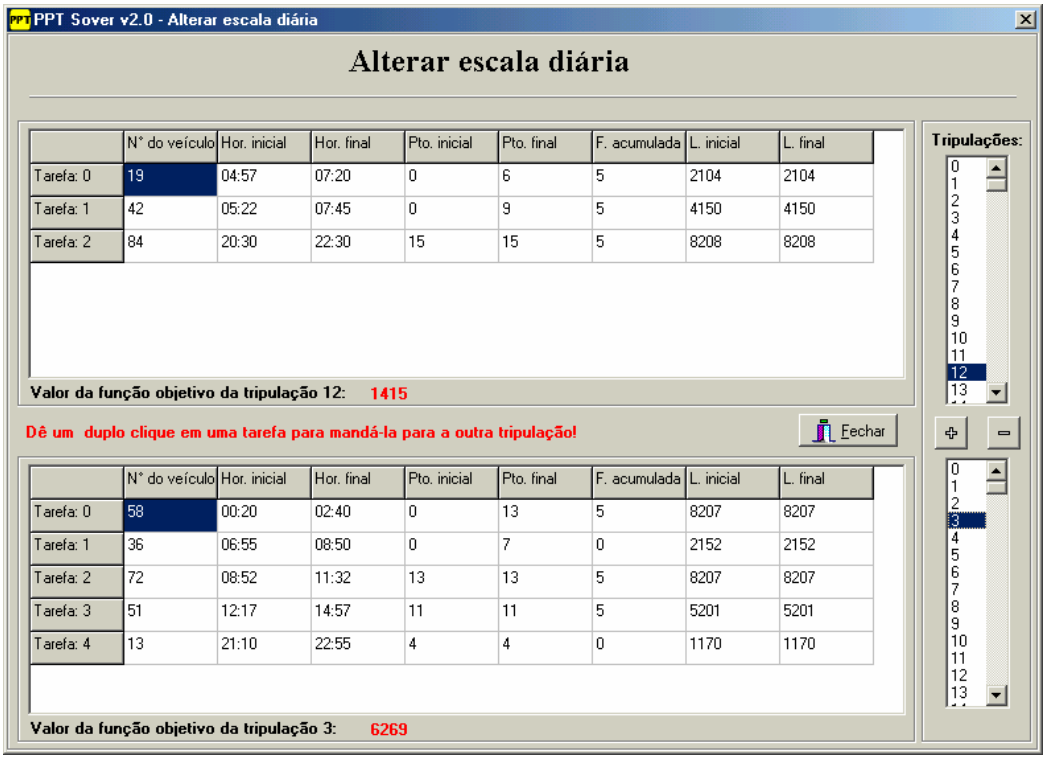
Figura 4.19 - Tela de execução do SA e visualização da escala mensal.

Para iniciar a execução do SA o usuário deve clicar no botão “Executar” na parte superior da tela (Figura 4.18 ou 4.19). Clicando no botão “Interromper”, abaixo do “Executar”, o usuário pára a execução do SA e visualiza a solução (escala diária ou mensal)

encontrada até o momento. Caso o usuário queira continuar a execução, basta-o clicar novamente no botão “Executar” e a execução continuará do “ponto” onde parou.

4.4.5 Alterar manualmente as escalas diária e mensal

O sistema permite ao usuário trocar tarefas e jornadas (manualmente) entre tripulações. Para isso, na Figura 4.20 ou 4.21, basta-o selecionar duas tripulações (nas listas à direita das tabelas), depois dar um duplo clique em alguma tarefa ou jornada de uma tabela e ela será enviada para a outra tabela, ou seja, outra tripulação.



	N° do veículo	Hor. inicial	Hor. final	Pto. inicial	Pto. final	F. acumulada	L. inicial	L. final
Tarefa: 0	19	04:57	07:20	0	6	5	2104	2104
Tarefa: 1	42	05:22	07:45	0	9	5	4150	4150
Tarefa: 2	84	20:30	22:30	15	15	5	8208	8208

	N° do veículo	Hor. inicial	Hor. final	Pto. inicial	Pto. final	F. acumulada	L. inicial	L. final
Tarefa: 0	58	00:20	02:40	0	13	5	8207	8207
Tarefa: 1	36	06:55	08:50	0	7	0	2152	2152
Tarefa: 2	72	08:52	11:32	13	13	5	8207	8207
Tarefa: 3	51	12:17	14:57	11	11	5	5201	5201
Tarefa: 4	13	21:10	22:55	4	4	0	1170	1170

Figura 4.20 - Tela de alteração manual de escalas diária.

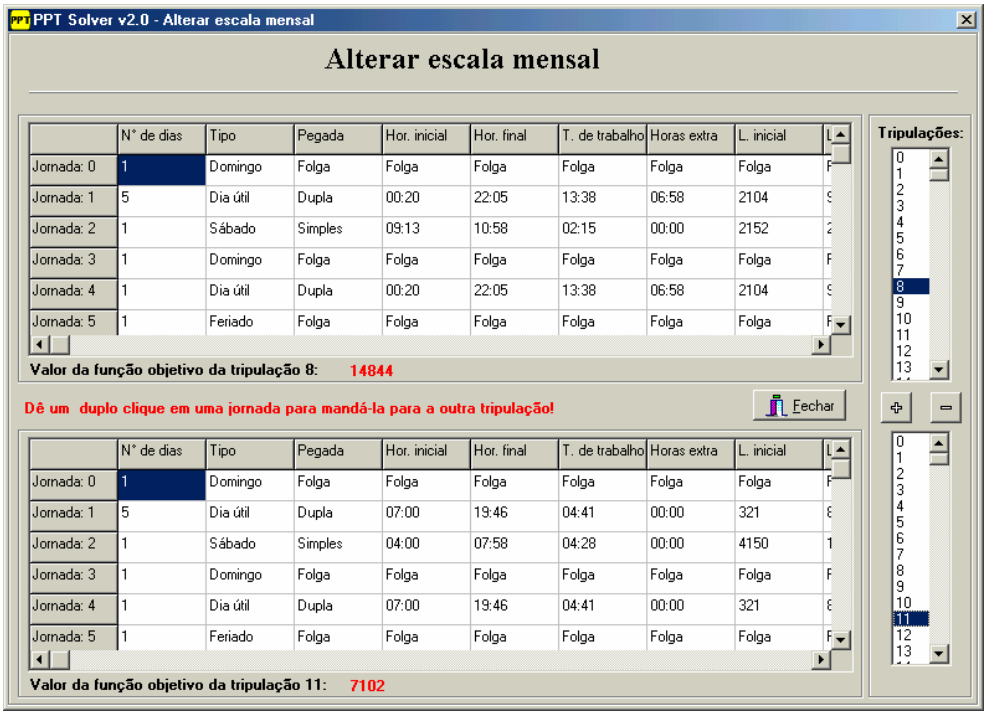


Figura 4.21 - Tela de alteração manual de escalas mensal.

4.4.6 Inserir e remover tripulações manualmente

Para inserir uma nova tripulação na escala, basta o usuário clicar no botão "+", do lado direito da tela representada pela Figura 4.20, assim uma nova tripulação será criada (sem tarefas). E, para remover uma tripulação, basta clicar no botão "-". Feito isso, a tripulação "vazia" (sem tarefas) mais ao fim da lista de tripulações será excluída da escala, e caso não haja tarefas vazias, uma mensagem será exibida e nenhuma tripulação será removida.

Este recurso não está disponível para a programação mensal (Figura 4.21).

4.4.7 Agrupar linhas

Na tela representada pela Figura 4.22, o usuário pode definir quais as linhas (contidas na programação de veículos) pertencem ao mesmo grupo.

Figura 4.22 - Tela para agrupar as linhas.

4.4.8 Selecionar um mês para a escala mensal

Na tela representada pela Figura 4.23, o usuário seleciona para qual mês será a escala mensal será construída. Aqui, ele também informa quais são os feriados do mês selecionado.

Figura 4.23 - Tela para seleção de um mês.

4.5 Telas do sistema

1) Tela de entrada do arquivo que contém a programação de veículos.

Nessa tela (representada pela Figura 4.24) o usuário seleciona o arquivo que contém a programação de veículos.

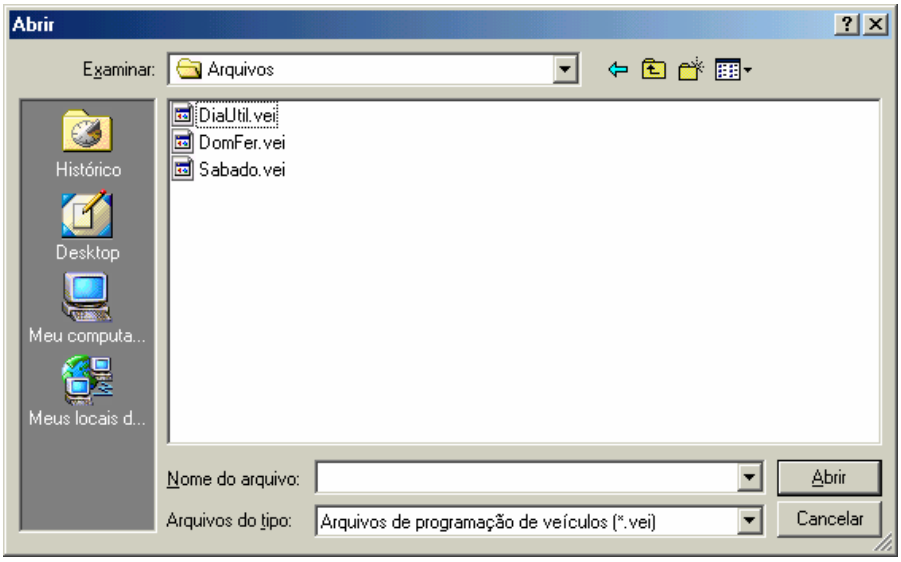


Figura 4.24 - Tela de entrada do arquivo da programação de veículos.

2) Tela de entrada dos arquivos que contém as escalas diárias para criação da mensal.

Nessa tela (representada pela Figura 4.25) o usuário seleciona os arquivos que contém as escalas diárias para dias úteis, sábados e domingos e feriados.

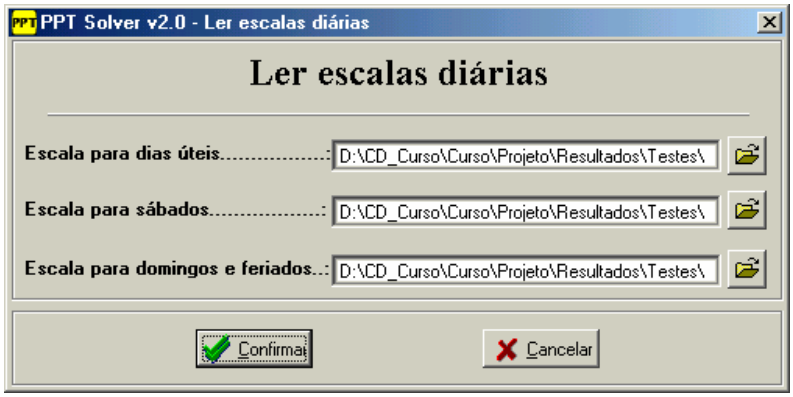


Figura 4.25 - Tela de entrada das escalas diárias para criação da mensal.

3) Tela de visualização das tarefas de uma tripulação.

Nessa tela (representada pela Figura 4.26) o usuário visualiza as tarefas de uma determinada tripulação.

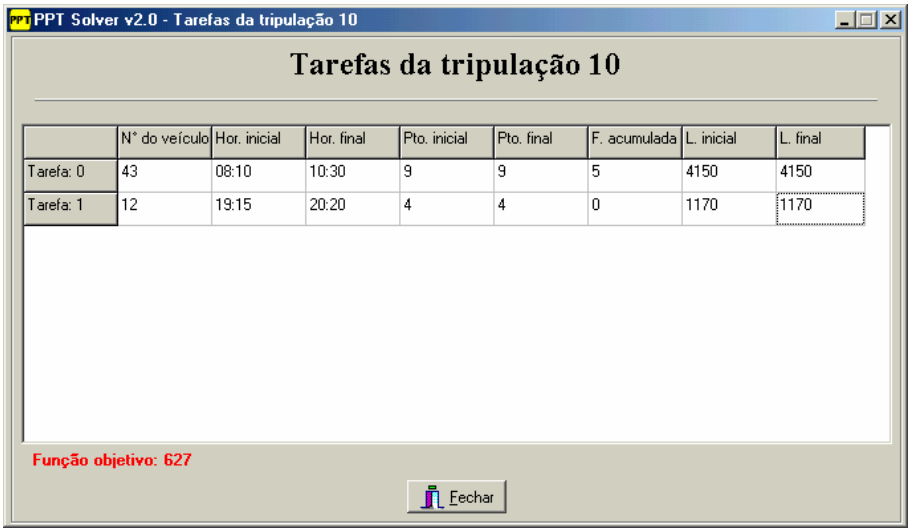


Figura 4.26 - Tela de visualização das tarefas de uma tripulação.

Para acessar essa tela (representada pela Figura 4.26) basta o usuário selecionar e clicar com o botão direito em uma tripulação na tabela da tela de exibição da escala diária (tela representada pela Figura 4.18).

4) Tela de visualização das jornadas de uma tripulação.

Nessa tela (representada pela Figura 4.27) o usuário visualiza as jornadas de uma determinada tripulação.

	N° de dias	Tipo	Pegada	Hor. inicial	Hor. final	T. de trabalho	Horas extra	L. inicial
Jornada: 0	3	Dia útil	Dupla	08:10	20:20	03:25	00:00	4150
Jornada: 1	1	Sábado	Dupla	05:11	24:40	07:16	00:36	2104
Jornada: 2	1	Domingo	Folga	Folga	Folga	Folga	Folga	Folga
Jornada: 3	5	Dia útil	Dupla	08:10	20:20	03:25	00:00	4150
Jornada: 4	1	Sábado	Dupla	05:11	24:40	07:16	00:36	2104
Jornada: 5	1	Domingo	Folga	Folga	Folga	Folga	Folga	Folga
Jornada: 6	5	Dia útil	Dupla	08:10	20:20	03:25	00:00	4150
Jornada: 7	1	Sábado	Dupla	05:11	24:40	07:16	00:36	2104
Jornada: 8	1	Domingo	Folga	Folga	Folga	Folga	Folga	Folga
Jornada: 9	5	Dia útil	Dupla	08:10	20:20	03:25	00:00	4150

Figura 4.27 - Tela de visualização das jornadas de uma tripulação.

Para acessar essa tela (representada pela Figura 4.27) basta o usuário selecionar e clicar com o botão direito em uma tripulação na tabela da tela de exibição da escala mensal (tela representada pela Figura 4.19).

5) Tela de verificação do resultado obtido para a escala diária.

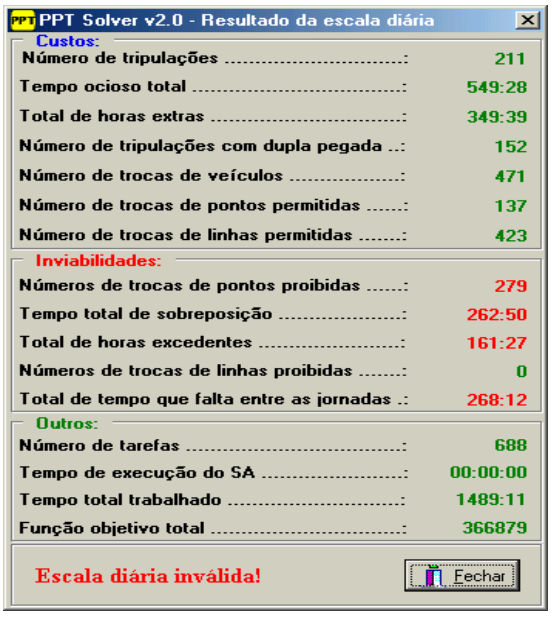


Figura 4.28 - Tela de verificação do resultado obtido para a escala diária.

Nessa tela (representada pela Figura 4.28) o usuário verifica a validade da escala diária encontrada. Caso o resultado apresente alguma inviabilidade, esta será apresentada em vermelho, caso contrário, em verde.

6) Tela de verificação do resultado obtido para a escala mensal.

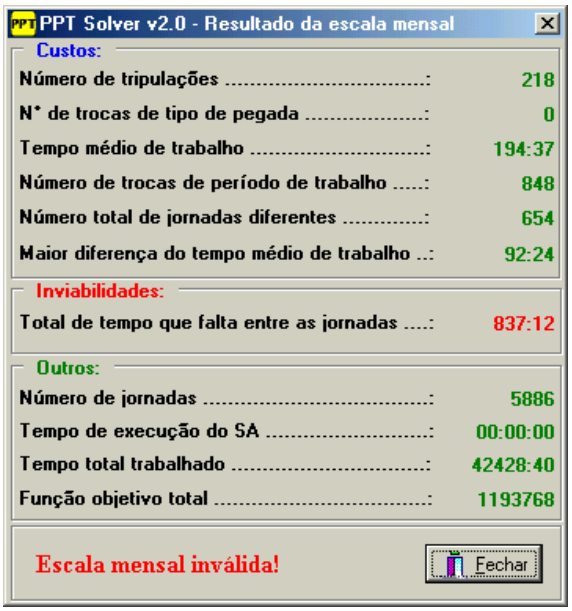


Figura 4.29 - Tela de verificação do resultado obtido para a escala mensal.

Nessa tela (representada pela Figura 4.29) o usuário verifica a validade da escala mensal encontrada. Caso o resultado apresente alguma inviabilidade, esta será apresentada em vermelho, caso contrário, em verde.

7) Tela de alteração das cores do sistema.

Nessa tela (representada pela Figura 4.30) o usuário pode alterar algumas cores do sistema.

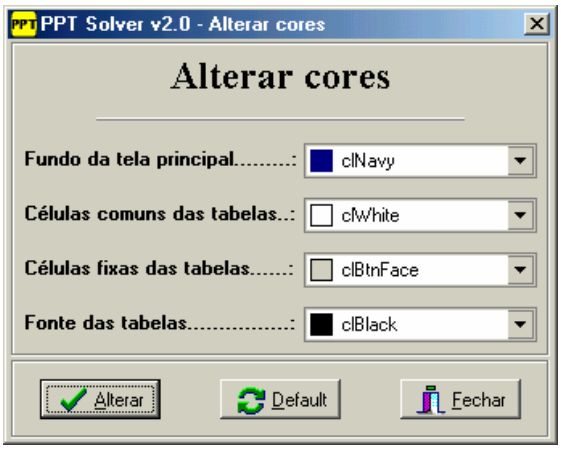


Figura 4.30 - Tela de alteração das cores do sistema.

8) Tela principal do sistema.

A partir dessa tela (representada pela Figura 4.31) o usuário seleciona as demais telas do sistema.

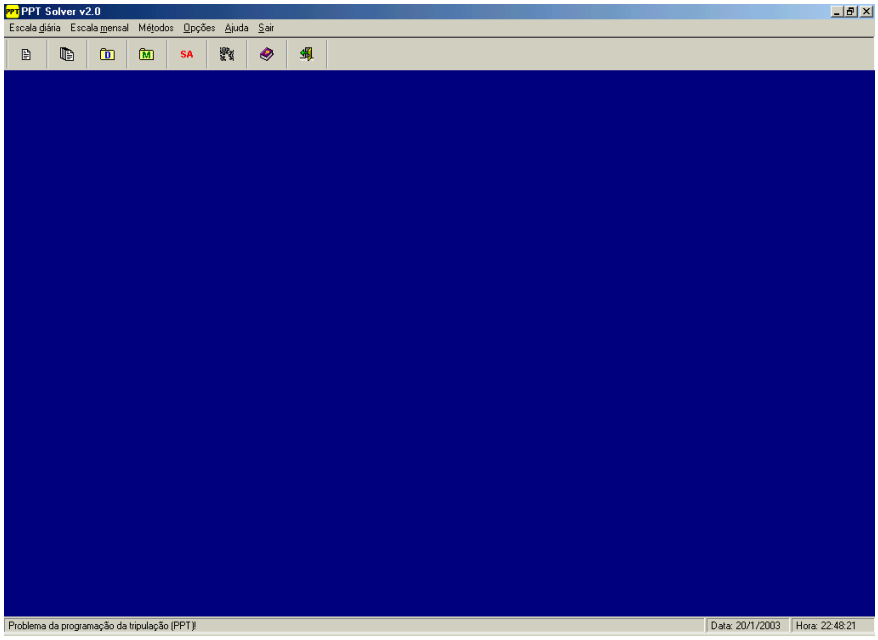


Figura 4.31 - Tela principal do sistema.

5 Resultados obtidos

Com o intuito de validar o sistema desenvolvido, esse foi utilizado para tentar resolver um caso real de uma empresa de transporte público que opera na cidade de Belo Horizonte.

Os dados relativos aos dias úteis foram obtidos a partir de dados reais da empresa considerada para análise. Já os demais dados relativos a sábados, domingos e feriados foram extraídos de uma programação de veículos hipotética, uma vez que não tínhamos essas informações disponíveis.

Estes testes foram realizados em um microcomputador com processador AMD Athlon 850 MHz e 128 MB de memória RAM. O tempo de CPU de cada teste foi de 60 minutos.

Testes realizados para a programação diária:

Inicialmente foram realizados testes visando calibrar os diversos parâmetros do modelo, bem como encontrar o melhor conjunto de valores para representar as penalizações. Feito isso, foram realizados cinco testes, cada qual partindo de uma semente diferente de números aleatórios. Os valores dos parâmetros para o *Simulated Annealing* relativos à programação diária, bem como as penalizações foram iguais em todos os testes. Estas informações estão representadas nas Tabela 5.1, 5.2 e 5.3, respectivamente. Já os resultados para os testes estão descritos na Tabela 5.4. Estes parâmetros e resultados são apenas para a programação diária para dias úteis.

Parâmetros utilizados pelo *Simulated Annealing*	
Número máximo de iterações	200000
Temperatura inicial	1000000
Taxa de resfriamento	0,975
Temperatura de congelamento	0,01
Tempo máximo de processamento	01:00
Critério de parada	
Temperatura de congelamento	false
Tempo máximo de processamento	true

Tabela 5.1 - Parâmetros do *S.A.* utilizados nos testes da programação diária.

Parâmetros para a programação diária	
Número de tripulações	219
Tempo mínimo entre jornadas	11:00
Tempo de troca de tripulação	00:05
Número máximo de tripulações com dupla pegada	20
Intervalo total para repouso e/ou alimentação	00:30
Sub-intervalo contínuo para repouso e/ou alimentação	00:15
Tempo máximo de trabalho	09:10
Tempo normal de trabalho	06:40
Tempo que indica dupla pegada	02:00

Tabela 5.2 - Parâmetros utilizados nos testes da programação diária.

Pesos	
Para falta de tempo entre jornadas	5000
Para o número de tripulações	1000
Para o tempo ocioso	40
Para o tempo total de trabalho	5000
Para as horas extras	60
Para o tempo de sobreposição	5000
Para o n° de trocas de ponto proibidas	13000
Para o n° de trocas de ponto permitidas	300
Para o n° de trocas de linha proibidas	13000
Para o n° de trocas de linha permitidas	300
Para o n° de trocas de veículos	5000
Para o n° máx. de tripulações com d. pegada	9000

Tabela 5.3 - Pesos utilizados nos testes da programação diária.

	Teste 1	Teste 2	Teste 3	Teste 4	Teste 5
Número de tripulações	218	219	218	218	218
Número de tarefas	688	688	688	688	688
Tempo total trabalhado	1583:44	1585:49	1583:28	1583:44	1583:44
Tempo que falta entre as jornadas	00:00	00:00	00:00	00:00	00:00
Tempo total de sobreposição	00:00	00:00	00:00	00:00	00:00
Horas excedentes	00:00	00:00	00:00	00:00	00:00
Hora extra total	95:41	97:39	94:29	95:41	95:41
Tempo ocioso total	129:53	139:30	129:01	129:53	129:53
Nº de tripulações com d. pegada	20	20	20	20	20
N° de trocas de veículos	28	26	27	28	28
N° de trocas de ponto proibidas	0	0	0	0	0
N° de trocas de ponto permitidas	17	13	15	17	17
N° de trocas de linha proibidas	0	0	0	0	0
N° de trocas de linha permitidas	12	10	12	12	12
Função objetivo total	1022880	1042240	1010880	1022880	1022880
Tempo de execução do SA	*01:00:01*	*01:00:03*	*01:00:03*	*01:00:01*	*01:00:02*

Tabela 5.4 - Resultados obtidos nos testes da programação diária.

Testes realizados para a programação mensal:

O procedimento realizado para a programação mensal é idêntico ao realizado para a diária. Os valores dos parâmetros para o *Simulated Annealing* relativos à programação mensal, bem como as penalizações, foram iguais em todos os testes. Estes dados estão representados nas Tabela 5.5, 5.6 e 5.7, respectivamente. Já os resultados para os testes estão descritos na Tabela 5.8.

Parâmetros utilizados pelo *Simulated Annealing*	
Número máximo de iterações	10000
Temperatura inicial	1000
Taxa de resfriamento	0.975
Temperatura de congelamento	0.01
Tempo máximo de processamento	01:00
Critério de parada	
Temperatura de congelamento	false
Tempo máximo de processamento	true

Tabela 5.5 - Parâmetros do S.A. utilizados nos testes da programação mensal.

Parâmetros para a programação diária	
Tempo mínimo entre jornadas	11:00
Maior diferença do tempo médio de trabalho	05:00
Horário limite para troca do período de trabalho	08:00

Tabela 5.6 - Parâmetros utilizados nos testes da programação mensal.

Pesos	
Para falta de tempo entre jornadas	20
Para o tempo médio trabalhado	1
Para o n° de jornadas diferentes	13
Para o n° de trocas de tipo de pegada	20
Para o n° de trocas de período de trabalho	13

Tabela 5.7 - Pesos utilizados nos testes da programação mensal.

	Teste 1	Teste 2	Teste 3	Teste 4	Teste 5
Número de tripulações	218	218	218	218	218
Nº de jornadas	5886	5886	5886	5886	5886
Tempo total trabalhado	42428:40	42428:40	42428:40	42428:40	42428:40
Tempo que falta entre as jornadas	00:00	00:00	00:00	00:00	00:00
Tempo médio de trabalho	194:37	194:37	194:37	194:37	194:37
Maior diferença do tempo médio de trabalho	04:58	05:00	05:11	05:12	05:08
N° total de jornadas diferentes	993	987	966	964	975
Nº total de trocas de tipo de pegada	46	52	50	52	46
Nº total de trocas do per. de trabalho	80	82	74	88	82
Função objetivo total	14869	14937	14541	14737	14669
Tempo de execução do SA	01:00:00	01:00:03	01:00:02	01:00:02	01:00:01

Tabela 5.8 - Resultados obtidos nos testes da programação mensal.

6 Conclusão

6.1 Análise dos resultados e conclusões

6.1.1 Programação diária

Na Tabela 6.1 são mostrados os valores da função objetivo obtidos nos testes. Já a Tabela 6.2 mostra a média dos valores das funções objetivo obtidos em cada teste, bem como o desvio apresentado em relação à melhor solução conhecida. Na Tabela 6.3 é feita uma comparação dos valores fornecidos pela empresa (para programação diária para dias úteis) com o resultado da melhor solução encontrada nos testes.

	Teste 1	**Teste 2**	**Teste 3**	**Teste 4**	**Teste 5**
F. Objetivo	1022880	1042240	1010880	1022880	1022880

Tabela 6.1 - Função objetivo dos testes realizados para a programação diária

Melhor função objetivo conhecida	1010880
Média das funções objetivo	1024352
Desvio	**1,3%**

Tabela 6.2 - Desempenho do método heurístico para a programação diária.

$$\textbf{Desvio} = \left(\frac{FOMedia - FOMelhor}{FOMelhor} \right) \times 100$$

	Empresa	**Melhor solução**
Horas extras (hh:mm)	**116:00**	**94:29**
Horas excedentes (hh:mm)	**12:06**	**00:00**
Ociosidade (hh:mm)	**188:22**	**129:01**
Duplas pegadas	**21**	**20**
N° de tripulações	**219**	**218**

Tabela 6.3 - Comparação dos resultados obtidos para a programação diária

Podemos verificar que a melhor solução encontrada pelo algoritmo apresenta uma diminuição significativa da ociosidade e de horas extras em relação à adotada na programação de dias úteis da empresa (vide Tabela 6.3). O algoritmo conseguiu, ainda, eliminar completamente o número de horas excedentes à jornada de trabalho.

Além disso, em todos os testes o algoritmo conseguiu atender aos requisitos essenciais, ou seja, obteve, em cada caso, soluções viáveis.

Pela análise da Tabela 6.2 verificamos que a metaheurística *Simulated Annealing* se mostrou robusta, pois partindo-se de diferentes soluções iniciais chega-se a soluções finais que diferem da melhor solução conhecida de apenas 1,3%.

Portanto, os resultados obtidos validam a utilização da dessa metaheurística para a resolução do Problema da Programação Diária da Tripulação.

6.1.2 Programação mensal

Na Tabela 6.4 são mostrados os valores das funções objetivo obtidas nos testes. Como os valores para a programação mensal utilizados na empresa foram hipotéticos, não foi possível uma validação "concreta" do módulo mensal do sistema. Ressalta-se, entretanto, que todas as inviabilidades foram eliminadas.

	Teste 1	**Teste 2**	**Teste 3**	**Teste 4**	**Teste 5**
F. Objetivo	14869	14937	14541	14737	14669

Tabela 6.4 - Função objetivo dos testes realizados para a programação mensal.

Melhor função objetivo conhecida	14541
Média das funções objetivo	14751
Desvio	**1,4%**

Tabela 6.5 - Desempenho do método heurístico para a programação mensal.

$$\textbf{Desvio} = \left(\frac{FOMedia - FOMelhor}{FOMelhor} \right) \times 100$$

Mais uma vez, em todos os testes, o algoritmo conseguiu atender aos requisitos essenciais, ou seja, obteve, em cada caso, soluções viáveis.

Pela análise da Tabela 6.5 verificamos que a metaheurística *Simulated Annealing* se mostrou robusta, pois partindo-se de diferentes soluções iniciais chega-se a soluções finais que diferem da melhor solução conhecida de apenas 1,4%.

Portanto, os resultados obtidos também validam a utilização dessa metaheurística para a resolução do Problema da Programação Mensal da Tripulação.

Em suma, apesar das dificuldades inerentes à abordagem do PPT, os resultados obtidos, tanto para a programação diária quanto para a mensal, mostram o cumprimento das leis trabalhistas e regras operacionais, além de mostrar uma significativa redução nos custos referentes à mão-de-obra operacional da programação diária. Tais resultados validam o sistema desenvolvido e mostram o potencial da metaheurística *Simulated Annealing*.

6.2 Sugestões para trabalhos futuros

- Implementar outros requisitos que não foram abordados tanto na Programação Diária, quanto na Mensal, como por exemplo, a fixação de uma tarefa a um determinado tripulante.
- Utilizar outros métodos na resolução do PPT visando à comparação com o *Simulated Annealing*.
- Desenvolver técnicas híbridas envolvendo a metaheurística *Simulated Annealing* para solucionar o PPT.

7 Referências Bibliográficas

BOUZADA, C. F. . **Análise das despesas administrativas no custo do transporte coletivo por ônibus no município de Belo Horizonte**. 2002. Dissertação (Mestrado). Escola de Governo, Fundação João Pinheiro, Belo Horizonte, 2002.

CLEMENT; WREN. Greed genetic algorithms, optimizing mutantis and bus driver scheduling. In: DADUNA, J. R.; BRANCO, I.; PAIXÃO, J. M. P. (ed.). **Computer-aided transit scheduling**. Berlin: Springer-Verlag, 1995.

DADUNA, J. R.; BRANCO, I.; PAIXÃO, J. M. P. (ed.). **Computer-aided transit scheduling**. Berlin: Springer-Verlag, 1995.

DESROCHERS, M.; ROUSSEAU, J. M. (ed.). **Computer-aided transit scheduling**. Berlin: Springer-Verlag, 1992.

ELIAS, S. E. G. **The use of digital computers in the economic scheduling for both man and machine in public transport**. Kansas,EUA: Technical Report 49, Kansas State University Bulletin, 1964.

GLOVER, F. Tabu search: part I. **ORSA Journal on Computing**, 1. p. 190-206, 1989.

GLOVER, F. Tabu search: part II. **ORSA Journal on Computing**, 2. p. 04-32, 1990.

GOLDBERG, D. E. **Genetic algorithms in search, optimization and machine learning**. Berkeley: Addison-Wesley, 1989.

KIRKPATRICK, S.; GELLAT, D. C.; VECCHI, M.P. Optimization by simulated annealing. **Science**, p. 671–680, 1983.

KWAN, A. S. K.; KWAN, R. K.; WREN, A. . Driver scheduling using genetic algorithms with embedded combinatorial traits. In: WILSON, N. H. M. (ed.). **Computer-aided transit Scheduling**. Berlin: Springer-Verlag, 1999. p. 81-102.

MANINGTON, P. D.; WREN, A. . Experiences with a bus scheduling algorithm which saves vehicles. In: WORKSHOP ON AUTOMATED TECHNIQUES FOR SCHEDULING OF VEHICLE OPERATORS FOR URBAN PUBLIC TRANSPORTATION SERVICES. Chicago, 1975.

SHEN, Y.; KWAN, R. S. K. . **Tabu Search for driver scheduling**. Berlin, 2000.

VOß, S.; DADUNA, J. (ed.). **Computer-aided scheduling of public transport**. Berlin: Springer-Verlag, 2001.

WILSON, N. H. M. (ed.). **Computer-aided transit. scheduling**. Berlin: Springer-Verlag, 1999.

WREN, A.; ROUSSEAU, J. M. . Bus driver scheduling - An overview. In DADUNA, J. R.; BRANCO, I.; PAIXÃO, J. M. P. (ed.). **Computer-aided transit scheduling**. Berlin: Springer-Verlag, 1995. p. 173-187.

WREN, A.; WREN, D. O. . **A genetic algorithm for public transport driver scheduling**. Computer and Operations Research 22, v.1, p. 101-110, 1995.

Anexo A. Código Fonte do Sistema

Devido ao tamanho do código fonte do sistema, foi colocada como anexo apenas a parte principal deste.

```
/*****************************************************************************
* Sistema...................: PPT_Solver
* Unit......................: UEscDiaria                          Tipo.......: .h
* Autor.....................: Geraldo Regis Mauri (KÇAPA)
* Última atualização........: 10 de JANEIRO de 2003
* Objetivo..................: Criar estruturas referentes à escala diária.
* Descrição.................: Esta unit contém a "declaração" de uma classe
*                             (TEscDiaria) referente à escala diária.
* Obs.......................: O arquivo .cpp dessa unit possui a implementação de
*                             tudo que foi declarado aqui.
* Dependências..............:
*     1) UParPpt
*     2) UPesos
*****************************************************************************/

#ifndef UEscDiariaH
#define UEscDiariaH

// -------------------------- ARQUIVOS "IMPORTADOS" ----------------------- \\
#include "classes.hpp"

#include "UParPpt.h"
#include "UPesos.h"
//---------------------------------------------------------------------------


/* ===========================================================================
- Classe.....: TEscDiaria
- Objetivo...: Possui os atributos e métodos referentes à escala diária.
=========================================================================== */
class TEscDiaria{
  private:
    void EfetuarCalcED(TParPptED *, TPesosED *);

  public:
    bool Valida;               // Indica se a escala é válida ou não.
    char Tipo;                 // Indica o tipo da escala (dia útil, sab, domFer).
    int NTripComTarefa;        // Número de tripulações com tarefas.
    int NTripDP;               // Número de tripulações com dupla pegada.
    long int FncObjTotal;      // Valor da função objetivo da escala.
    long int FOTotTripulacoes; // Valor da soma das funções objetivo das
tripulações.
    TList *LstTripulacoes;     // Lista de tripulações.

    TEscDiaria();
    void InicializaLstTripulacoes(int);
    void CriarJornadas(TList *);
    TEscDiaria *Clone();
    void Movimento(TList **, int, TList **);
    void EfetuarCalculos(TParPptED *, TPesosED *);
    void EfCalcDepoisDoMovimento(TParPptED *, TPesosED *, int, int);
    void GravarEscala(String);
    void CarregarEscala(String);
    void Free();
};
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
extern TEscDiaria *escdiaria;
//---------------------------------------------------------------------------
```

```
//---------------------------------------------------------------------------
#endif
// ---------------------------------- FIM ---------------------------------- \\




/*****************************************************************************
* Sistema..................: PPT_Solver
* Unit.....................: UEscDiaria                      Tipo.......: .cpp
* Autor....................: Geraldo Regis Mauri (KÇAPA)
* Última atualização.......: 10 de JANEIRO de 2003
* Objetivo.................: Criar estruturas referentes à escala diária.
* Descrição................: Esta unit contém a "implementação" de uma classe
*                            (TEscDiaria) referente à escala diária.
* Obs......................: O arquivo .h dessa unit possui a declaração de
*                            tudo que foi implementado aqui.
* Dependências.............:
*     1) UTripulacoes
*     2) UTar_Jorn
*     3) UUtil
*****************************************************************************/

#pragma hdrstop

#include "UEscDiaria.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------------------------------

// --------------------------- ARQUIVOS "IMPORTADOS" ----------------------- \\
#include "stdio.h"
#include "vcl.h"

#include "UTripulacoes.h"
#include "UTar_Jorn.h"

#include "UUtil.cpp"
//---------------------------------------------------------------------------

#define max(a, b)  (((a) > (b)) ? (a) : (b))

//---------------------------------------------------------------------------

TEscDiaria *escdiaria;

/* ============================================================================
- Método.....: TEscDiaria
- Objetivo...: Criar e inicializar o objeto TEscDiaria.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================ */
TEscDiaria::TEscDiaria()
{
 Valida           = false;
 String Tipo      = 'U';
 NTripComTarefa   = 0;
 NTripDP          = 0;
 FncObjTotal      = 0;
 FOTotTripulacoes = 0;
 LstTripulacoes = new TList();
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: InicializarLstTripulacoes
- Objetivo...: "Criar" NTrip tripulações e adicioná-las na lista.
```

```
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) NTrip -> Número de tripulações.
============================================================================== */
void TEscDiaria::InicializaLstTripulacoes(int NTrip)
{
 for (int i=0; i<NTrip; i++)
  {
   TTripED *T = new TTripED();
   this->LstTripulacoes->Add(T);
  }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CriarJornadas
- Objetivo...: Criar as jornadas e atribuí-las às tripulações.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) LstTar -> Lista de tarefas (criada a partir da prog. de veículos).
============================================================================== */
void TEscDiaria::CriarJornadas(TList *LstTar)
{
 int ListSize = LstTar->Count;
 TTarefa *T;
 TTripED *Trip;

 for (int i = 0; i < ListSize; i++)
  {
   T = (TTarefa *)LstTar->Extract(LstTar->Items[random(LstTar->Count)]);
   LstTar->Capacity = LstTar->Count;
   Trip = (TTripED *)this->LstTripulacoes->Items[random(this->LstTripulacoes-
>Count)];
   Trip->LstTarefas->Add(T);
  }

 for (int i = 0; i < this->LstTripulacoes->Count; i++)
  {
   Trip = (TTripED *)this->LstTripulacoes->Items[i];
   Trip->LstTarefas->Sort(OrdenaListaHIni);
  }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: Clone
- Objetivo...: Criar e retornar uma escala diária com os valores iguais aos da
               escala diária que chamou o método.
- Retorno....: Clone -> Objeto (TEscDiaria) com os mesmos valores da escala que
                        chamou o método.
- Parâmetros.: <-- Nenhum -->
============================================================================== */
TEscDiaria *TEscDiaria::Clone()
{
 TTripED *Trip, *TripC;
 TTarefa *Tar, *TarC;

 TEscDiaria *Clone = new TEscDiaria();

 Clone->InicializaLstTripulacoes(this->LstTripulacoes->Count);

 int i = 0;
 int j = 0;

 Clone->Valida           = this->Valida;
 Clone->FncObjTotal      = this->FncObjTotal;
 Clone->FOTotTripulacoes = this->FOTotTripulacoes;
 Clone->Tipo             = this->Tipo;
 Clone->NTripComTarefa   = this->NTripComTarefa;
 Clone->NTripDP          = this->NTripDP;
```

```
 while (i < this->LstTripulacoes->Count)
  {
   Trip = (TTripED *)this->LstTripulacoes->Items[i];
   TripC =(TTripED *)Clone->LstTripulacoes->Items[i];

   TripC->TmpTTrabalho       = Trip->TmpTTrabalho;
   TripC->TmpEJornadas       = Trip->TmpEJornadas;
   TripC->FncObjetivo        = Trip->FncObjetivo;
   TripC->FolAcuTotal        = Trip->FolAcuTotal;
   TripC->HExtra             = Trip->HExtra;
   TripC->TmpOcioso          = Trip->TmpOcioso;
   TripC->TmpSobreposicao    = Trip->TmpSobreposicao;
   TripC->NTarefas           = Trip->NTarefas;
   TripC->NDPegadas          = Trip->NDPegadas;
   TripC->NTPProibidas       = Trip->NTPProibidas;
   TripC->NTPPermitidas      = Trip->NTPPermitidas;
   TripC->NTLProibidas       = Trip->NTLProibidas;
   TripC->NTLPermitidas      = Trip->NTLPermitidas;
   TripC->NTVeiculos         = Trip->NTVeiculos;
   TripC->MaiorIntEntTarefas = Trip->MaiorIntEntTarefas;
   TripC->TVirtual           = Trip->TVirtual;

   while (j < Trip->LstTarefas->Count)
    {
     Tar = (TTarefa *)Trip->LstTarefas->Items[j];
     TarC = new TTarefa();
     TarC->HInicio = Tar->HInicio;
     TarC->HFim    = Tar->HFim;
     TarC->LinIni  = Tar->LinIni;
     TarC->LinFin  = Tar->LinFin;
     TarC->NumVeic = Tar->NumVeic;
     TarC->PIni    = Tar->PIni;
     TarC->PFim    = Tar->PFim;
     TarC->FolAcu  = Tar->FolAcu;
     TripC->LstTarefas->Add(TarC);
     j++;
    }
   j = 0;
   i++;
  }

 return(Clone);
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: Movimento
- Objetivo...: Remover uma tarefa de uma lista e adiciar em outra.
- Retorno....: <-- Nenhum -->
- Parâmetros.: (As lista são passadas por referência)
     1) LstOrigem  -> Lista de onde será removida a tarefa.
     2) PosRem     -> Posição (na lista de origem) da tarefa a ser removida.
     3) LstDestino -> Lista de onde será adicionada a tarefa.
============================================================================ */
void TEscDiaria::Movimento(TList **LstOrigem, int PosRem, TList **LstDestino)
{
 TList *LO = *LstOrigem;
 TList *LD = *LstDestino;
 TTarefa *Aux;

 Aux = (TTarefa *)LO->Extract(LO->Items[PosRem]);
 LO->Capacity = LO->Count;
 LD->Add(Aux);
 LD->Sort(OrdenaListaHIni);
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: EfetuarCalculos
```

```
- Objetivo...: Efetuar todos os cálculos para a escala diária.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) ParED -> Objeto que contém os parâmetros para a escala diária.
     2) PED   -> Objeto que contém os pesos para a escala diária.
================================================================================ */
void TEscDiaria::EfetuarCalculos(TParPptED *ParED, TPesosED *PED)
{
 TTripED *T;
 this->FOTotTripulacoes = 0;
 this->NTripComTarefa   = 0;
 this->NTripDP          = 0;

 for (int i=0; i<this->LstTripulacoes->Count; i++)
  {
   T = (TTripED *)LstTripulacoes->Items[i];
   T->EfetuarCalcTrip(ParED, PED);
   this->NTripDP = this->NTripDP + T->NDPegadas;
   if (T->NTarefas != 0)
    this->NTripComTarefa++;
   this->FOTotTripulacoes = this->FOTotTripulacoes + T->FncObjetivo;
  }

 this->EfetuarCalcED(ParED, PED);
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: EfetuarCalcED
- Objetivo...: Calcular o valor da função objetivo total da escala diária.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) ParED -> Objeto que contém os parâmetros para a escala diária.
     2) PED   -> Objeto que contém os pesos para a escala diária.
================================================================================ */
void TEscDiaria::EfetuarCalcED(TParPptED *ParED, TPesosED *PED)
{
 this->FncObjTotal = 0;

 int ETripDP = 0;  // Excesso de tripulações com dupla pegada.

 ETripDP = max(0, this->NTripDP - ParED->NMaxTripDP);

 this->FncObjTotal = this->FOTotTripulacoes + (PED->PNumTrip * this-
>NTripComTarefa) +
                     (PED->PNMaxTripDP * ETripDP);
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: EfCalDepoisDoMovimento
- Objetivo...: Calcular o valor da função objetivo total depois de um movimento.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) ParED -> Objeto que contém os parâmetros para a escala diária.
     2) PED   -> Objeto que contém os pesos para a escala diária.
     3) T1    -> Nº de uma das tripulações do movimento.
     4) T2    -> Nº da outra tripulação do movimento.
================================================================================ */
void TEscDiaria::EfCalcDepoisDoMovimento(TParPptED *ParED, TPesosED *PED, int T1,
int T2)
{
 TTripED *Trip1, *Trip2;

 int FOT1Antes, FOT1Depois, FOT2Antes, FOT2Depois;
 int NDPT1Antes, NDPT2Antes, NDPT1Depois, NDPT2Depois;
 int TemTarT1Antes, TemTarT2Antes, TemTarT1Depois, TemTarT2Depois;

 Trip1 = (TTripED *)this->LstTripulacoes->Items[T1];
 Trip2 = (TTripED *)this->LstTripulacoes->Items[T2];
```

```
 FOT1Antes = Trip1->FncObjetivo;
 FOT2Antes = Trip2->FncObjetivo;
 NDPT1Antes = Trip1->NDPegadas;
 NDPT2Antes = Trip2->NDPegadas;
 if (Trip1->NTarefas == 0)
  TemTarT1Antes = 0;
 else
  TemTarT1Antes = 1;
 if (Trip2->NTarefas == 0)
  TemTarT2Antes = 0;
 else
  TemTarT2Antes = 1;

 Trip1->EfetuarCalcTrip(ParED, PED);
 Trip2->EfetuarCalcTrip(ParED, PED);

 FOT1Depois = Trip1->FncObjetivo;
 FOT2Depois = Trip2->FncObjetivo;
 NDPT1Depois = Trip1->NDPegadas;
 NDPT2Depois = Trip2->NDPegadas;
 if (Trip1->NTarefas == 0)
  TemTarT1Depois = 0;
 else
  TemTarT1Depois = 1;
 if (Trip2->NTarefas == 0)
  TemTarT2Depois = 0;
 else
  TemTarT2Depois = 1;

 this->FOTotTripulacoes = this->FOTotTripulacoes - FOT1Antes - FOT2Antes +
FOT1Depois + FOT2Depois;
 this->NTripComTarefa   = this->NTripComTarefa - TemTarT1Antes - TemTarT2Antes +
TemTarT1Depois + TemTarT2Depois;
 this->NTripDP          = this->NTripDP - NDPT1Antes - NDPT2Antes + NDPT1Depois +
NDPT2Depois;

 this->EfetuarCalcED(ParED, PED);
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: GravarEscala
- Objetivo...: Gravar a escala diária obtida em nos arquivos "EsDU.txt" ou
               "EsDF.txt" ou "EsSab.txt" ou.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) Dir -> Diretório de destino do arquivo gravado.
============================================================================ */
void TEscDiaria::GravarEscala(String Dir)
{
 String Diretorio, Arquivo;
 String Tipo, separador3;

 if (this->Tipo == 'U')
  {
   Diretorio = ExtractFilePath(Application->ExeName) + "\\Resultados\\EsDU.txt";
   Arquivo = "EsDU.txt";
   Tipo = "*  ESCALA DIÁRIA - Dias úteis  *";
   separador3 = "*********************************";
  }
 else if (this->Tipo == 'D')
  {
   Diretorio = ExtractFilePath(Application->ExeName) + "\\Resultados\\EsDF.txt";
   Arquivo = "EsDF.txt";
   Tipo = "*  ESCALA DIÁRIA - Domingos e feriados  *";
   separador3 = "*****************************************";
  }
 else
  {
```

```
   Diretorio = ExtractFilePath(Application->ExeName) + "\\Resultados\\EsSab.txt";
   Arquivo = "EsSab.txt";
   Tipo = "*  ESCALA DIÁRIA - Sábados  *";
   separador3 = "****************************";
  }
 if (Dir == "")
  Dir = Diretorio;

 char *Path = new char[ Diretorio.Length() + 1 ];
 strcpy(Path, Diretorio.c_str());

 FILE *Arq;
 Arq = fopen(Path, "w");
 delete(Path);
 if(!Arq)
  {
   Beep();
   MessageDlg("O arquivo " + Arquivo + " não pode ser aberto!", mtError,
              TMsgDlgButtons() << mbOK, 0);
  }
 else
  {
   char separador1[108] =

"================================================================================
========================";
   char separador2[108] =
   "--------------------------------------------------------------------------------
----------------------------";

   TTripED *Trip;
   TTarefa *T;
   String Titulo,Pegadas;
   int i = 0;
   int j = 0;
   int NTrip = 0;
   String Aux, NV, HI, HF, PI, PF, FA, LI, LF, TTT, TO, HE;

   fprintf(Arq,"\n\t\t\t\t\t\t\t%s\n\t\t\t\t\t\t\t%s\n\t\t\t\t\t\t\t%s\n\n\n\n",
           separador3, Tipo, separador3);

   while(i < this->LstTripulacoes->Count)
    {
     Trip = (TTripED *)this->LstTripulacoes->Items[i];

     if (Trip->LstTarefas->Count != 0)
      {
       Titulo = "TRIPULAÇÃO " + IntToStr(NTrip);

fprintf(Arq,"%s\n\t\t\t\t\t\t\t%s\n%s\n\t\t\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t
\t%s\n%s\n",

separador1,Titulo,separador1,"N_Vei","H_Ini","H_Fim","P_Ini","P_Fim",
               "F_Acu","L_Ini","L_Fim",separador2);
       NTrip++;
       while(j < Trip->LstTarefas->Count)
        {
         T = (TTarefa *)Trip->LstTarefas->Items[j];

         Titulo = "Tarefa " + IntToStr(j);
         NV = IntToStr(T->NumVeic);
         PI = IntToStr(T->PIni);
         PF = IntToStr(T->PFim);
         FA = IntToStr(T->FolAcu);
         LI = getLinha(T->LinIni);
         LF = getLinha(T->LinFin);

         HI = MinuteToStr(T->HInicio);
         HF = MinuteToStr(T->HFim);
```

```
          fprintf(Arq,"%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\n%s\n",
                  Titulo,NV,HI,HF,PI,PF,FA,LI,LF,separador2);
          j++;
         }
        if (Trip->TVirtual != 0)
         {
          Titulo = "T.virtual!";
          NV = "TV!";
          PI = "TV!";
          PF = "TV!";
          FA = "TV!";
          LI = "TV!";
          LF = "TV!";

          HI = MinuteToStr(T->HFim);
          HF = MinuteToStr(Trip->TVirtual);

          fprintf(Arq,"%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\n%s\n",
                  Titulo,NV,HI,HF,PI,PF,FA,LI,LF,separador2);
         }
        j=0;
        fprintf(Arq,"\n");

        TTT = MinuteToStr(Trip->TmpTTrabalho);
        TO  = MinuteToStr(Trip->TmpOcioso);
        HE  = MinuteToStr(Trip->HExtra);

        if (Trip->NDPegadas == 0)
         Pegadas = "Jornada com pegada única.";
        else
         Pegadas = "Jornada com dupla pegada.";

        fprintf(Arq,"%s%s\n%s%s\n%s%s\n%s\n%s\n\n\n", "Tempo trabalhado.....: ",
                TTT, "Tempo ocioso.....: ", TO, "Hora extra.....: ",
                HE, Pegadas, "-------------------------------------");
       }

      i++;
     }
   }
 fprintf(Arq,"%s", "FIM");

 fclose(Arq);

 char *DDestino = new char[ Dir.Length() + 1 ];
 strcpy(DDestino, Dir.c_str());
 char *DOrigem = new char[ Diretorio.Length() + 1 ];
 strcpy(DOrigem, Diretorio.c_str());

 CopyFile(DOrigem, DDestino, false);
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CarregarEscala
- Objetivo...: Carregar uma escala diária a partir de um arquivo (.esd).
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) Diretorio -> Diretório do arquivo selecionado.
============================================================================ */
void TEscDiaria::CarregarEscala(String Diretorio)
{
 char *Path = new char[ Diretorio.Length() + 1 ];
 strcpy(Path, Diretorio.c_str());

 FILE *Arq;
 Arq = fopen(Path, "r");
 delete(Path);
 if(!Arq)
  {
```

```
   Beep();
   MessageDlg("O arquivo que contém a escala diária não pode ser aberto!", mtError,
              TMsgDlgButtons() << mbOK, 0);
  }
 else
  {
   char separador1[108] = "-----------------------------------------------------------
-------------------------------------------------";
   char separador2[39] = "---------------------------------------";
   char Linha[150];
   char Titulo[10];

   TTripED *TripED;
   TTarefa *T;

   int NTar;
   int HI, MI, HF, MF, LI, LF, NV, PI, PF, FA;
   char TV[3];
   int HorTT, MinTT, HorTO, MinTO, HorHE, MinHE;
   char TipoDePegada[24];

   do
    {
     fscanf(Arq, "%s", &Linha);
    }while(strncmp(Linha, "-", 2));
    fscanf(Arq, "%s", &Linha);

   if (!strncmp(Linha, "Dias", 5))
    this->Tipo = 'U';
   else if (!strncmp(Linha, "Domingos", 10))
    this->Tipo = 'D';
   else
    this->Tipo = 'S';

   fscanf(Arq,"%s",&Linha);
   while (strncmp(Linha, "FIM", 3))
    {
     TripED = new TTripED();

     do
      {
       fscanf(Arq,"%s",&Linha);
      }while(strncmp(Linha, separador1,109));

     fscanf(Arq, "%s", &Titulo);

     while (strncmp(Titulo, "Tempo", 5))
      {
       if (!strncmp(Titulo, "Tarefa", 6))
        {
         fscanf(Arq, "%d %d %d:%d %d:%d %d %d %d %d %s", &NTar, &NV, &HI, &MI,
&HF, &MF, &PI, &PF, &FA, &LI, &LF, &Linha);
         T = new TTarefa();
         T->PreencherAtributos( (HI*60) + MI, (HF*60) + MF, LI, LF, NV, PI, PF,
FA);
         TripED->LstTarefas->Add(T);
        }
       else
        {
         fscanf(Arq, "%s %d:%d %d:%d %s %s %s %s %s %s", &TV, &HI, &MI, &HF, &MF,
&TV, &TV, &TV, &TV, &TV, &Linha);
         TripED->TVirtual = (HF*60) + MF;
        }

       fscanf(Arq, "%s", &Titulo);
      }
     this->LstTripulacoes->Add(TripED);
     do
      {
       fscanf(Arq,"%s",&Linha);
```

```
      }while(strncmp(Linha, separador2, 40));
     fscanf(Arq,"%s",&Linha);
    }
  }

 fclose(Arq);
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: Free
- Objetivo...: Destuir o objeto TEscDiaria e suas tripulações.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================ */
void TEscDiaria::Free()
{
 TTripED *T;
 for (int i = 0; i < this->LstTripulacoes->Count; i++)
  {
   T = (TTripED *)this->LstTripulacoes->Items[i];
   T->Free();
  }
 delete(this->LstTripulacoes);
 delete(this);
}
//---------------------------------------------------------------------------

// ---------------------------------- FIM ---------------------------------- \\




/*****************************************************************************
* Sistema..................: PPT_Solver
* Unit.....................: UEscMensal                          Tipo.......: .h
* Autor....................: Geraldo Regis Mauri (KÇAPA)
* Última atualização.......: 10 de JANEIRO de 2003
* Objetivo.................: Criar estruturas referentes à escala mensal.
* Descrição................: Esta unit contém a "declaração" de uma classe
*                            (TEscMensal) referente à escala mensal.
* Obs......................: O arquivo .cpp dessa unit possui a implementação de
*                            tudo que foi declarado aqui.
* Dependências.............:
*     1) UEstruturaMes
*     2) UParPpt
*     3) UPesos
*****************************************************************************/

#ifndef UEscMensalH
#define UEscMensalH

// -------------------------- ARQUIVOS "IMPORTADOS" ----------------------- \\
#include "classes.hpp"

#include "UEstruturaMes.h"
#include "UParPpt.h"
#include "UPesos.h"
//---------------------------------------------------------------------------


/* ============================================================================
- Classe.....: TEscMensal
- Objetivo...: Possui os atributos e métodos referentes à escala mensal.
============================================================================ */
class TEscMensal{
  private:
    TList *getTarefas(TEstruturaMes *, char, int);
```

```
    void EfetuarCalcEM();

  public:
    bool Valida;               // Indica se a escala é válida ou não.
    int TmpMedioTrab;          // Tempo médio de trabalho das tripulações.
    long int FncObjTotal;      // Valor da função objetivo da escala.
    long int FOTotTripulacoes; // Valor da soma das funções objetivo das
tripulações.
    TList *LstTripulacoes;     // Lista de tripulações.

    TEscMensal();
    void InicializaLstTripulacoes(int);
    void DistribuirJornadas(TEstruturaMes *);
    TEscMensal *Clone();
    void Movimento(TList **, int, TList **);
    void EfetuarCalculos(TEstruturaMes *, TParPptEM *, TPesosEM *);
    void EfCalcDepoisDoMovimento(TEstruturaMes *, TParPptEM *, TPesosEM *, int,
int);
    void CalcTmpMedioTrab(TEstruturaMes *);
    void GravarEscala(TEstruturaMes *, String);
    void CarregarEscala(TEstruturaMes **, String);

    void GravarArqAux(TEstruturaMes *, String);

    void Free();
};
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
extern TEscMensal *escmensal;
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
#endif
// ---------------------------------- FIM ------------------------------- \\




/*****************************************************************************
* Sistema..................: PPT_Solver
* Unit.....................: UEscMensal                        Tipo.......: .cpp
* Autor....................: Geraldo Regis Mauri (KÇAPA)
* Última atualização.......: 10 de JANEIRO de 2003
* Objetivo.................: Criar estruturas referentes à escala mensal.
* Descrição................: Esta unit contém a "implementação" de uma classe
*                            (TEscMensal) referente à escala mensal.
* Obs......................: O arquivo .h dessa unit possui a declaração de
*                            tudo que foi implementado aqui.
* Dependências.............:
*     1) UTripulacoes
*     2) UTar_Jorn
*     3) UEscDiaria
*     4) UUtil
*****************************************************************************/

#pragma hdrstop

#include "UEscMensal.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------------------------------

// --------------------------- ARQUIVOS "IMPORTADOS" ----------------------- \\
#include "stdio.h"
#include "vcl.h"

#include "UTripulacoes.h"
```

```
#include "UTar_Jorn.h"
#include "UEscDiaria.h"

#include "UUtil.cpp"
//---------------------------------------------------------------------------

#define max(a, b)  (((a) > (b)) ? (a) : (b))

//---------------------------------------------------------------------------

TEscMensal *escmensal;

/* ============================================================================
- Método.....: TEscMensal
- Objetivo...: Criar e inicializar o objeto TEscMensal.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================ */
TEscMensal::TEscMensal()
{
 Valida           = false;
 TmpMedioTrab     = 0;
 FncObjTotal      = 0;
 FOTotTripulacoes = 0;
 LstTripulacoes = new TList();
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: InicializarLstTripulacoes
- Objetivo...: "Criar" NTrip tripulações e adicioná-las na lista.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) NTrip -> Número de tripulações.
============================================================================ */
void TEscMensal::InicializaLstTripulacoes(int NTrip)
{
 for (int i=0; i < NTrip; i++)
  {
   TTripEM *T = new TTripEM();
   this->LstTripulacoes->Add(T);
  }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: DistribuirJornadas
- Objetivo...: Distribuir as jornadas às tripulações.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) EM -> Objeto que contém a estrutura do mês selecionado.
============================================================================ */
void TEscMensal::DistribuirJornadas(TEstruturaMes *EM)
{
 TTripEM *T;
 TDias *D, *DAux;
 int Aux;

 for (int i = 0; i < EM->LstJornDU->Count; i++)
  {
   T = (TTripEM *)this->LstTripulacoes->Items[i];
   for (int j = 0; j < EM->LstDias->Count; j++)
    {
     DAux = (TDias *)EM->LstDias->Items[j];
     D = new TDias();

     if (DAux->Tipo == 'U')
      D->Numero = i;

     D->Tipo = DAux->Tipo;
```

```
     D->Qtd  = DAux->Qtd;
     T->LstDias->Add(D);
    }
  }

 for (int i = 0; i < EM->LstJornSab->Count; i++)
  {
   T = (TTripEM *)this->LstTripulacoes->Items[i];
   for (int j = 0; j < T->LstDias->Count; j++)
    {
     D = (TDias *)T->LstDias->Items[j];
     if (D->Tipo == 'S')
      D->Numero = i;
    }
  }

 Aux = 0;
 for (int i = EM->LstJornSab->Count; i < this->LstTripulacoes->Count; i++)
  {
   T = (TTripEM *)this->LstTripulacoes->Items[i];
   for (int j = 0; j < T->LstDias->Count; j++)
    {
     D = (TDias *)T->LstDias->Items[j];
     if ( (D->Tipo == 'D') || (D->Tipo == 'M') || (D->Tipo == 'F') || (D->Tipo ==
'B') )
      D->Numero = Aux;
    }
   Aux++;
  }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: Clone
- Objetivo...: Criar e retornar uma escala mensal com os valores iguais aos da
               escala mensal que chamou o método.
- Retorno....: Clone -> Objeto (TEscMensal) com os mesmos valores da escala que
                        chamou o método.
- Parâmetros.: <-- Nenhum -->
============================================================================ */
TEscMensal *TEscMensal::Clone()
{
 TTripEM *T, *TC;
 TDias *D, *DC;

 TEscMensal *Clone = new TEscMensal();

 Clone->InicializaLstTripulacoes(this->LstTripulacoes->Count);

 int i = 0;
 int j = 0;

 Clone->Valida           = this->Valida;
 Clone->FncObjTotal      = this->FncObjTotal;
 Clone->FOTotTripulacoes = this->FOTotTripulacoes;
 Clone->TmpMedioTrab     = this->TmpMedioTrab;

 while (i < this->LstTripulacoes->Count)
  {
   T = (TTripEM *)this->LstTripulacoes->Items[i];
   TC = (TTripEM *)Clone->LstTripulacoes->Items[i];

   TC->TmpTTrabalho   = T->TmpTTrabalho;
   TC->TmpEJornadas   = T->TmpEJornadas;
   TC->FncObjetivo    = T->FncObjetivo;
   TC->NumJornDif     = T->NumJornDif;
   TC->NumTrocaTipPeg = T->NumTrocaTipPeg;
   TC->NumTroPerTrab  = T->NumTroPerTrab;

   while (j < T->LstDias->Count)
```

```
    {
     D = (TDias *)T->LstDias->Items[j];
     DC = new TDias();
     DC->Numero = D->Numero;
     DC->Tipo   = D->Tipo;
     DC->Qtd    = D->Qtd;
     TC->LstDias->Add(DC);
     j++;
    }
   j = 0;
   i++;
  }

 return Clone;
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: Movimento
- Objetivo...: Tira um objeto (TDias) de uma lista e o coloca em outra
               (na mesma posição), e tira o objeto da outra lista (na mesma
posição)
               e o insere no lugar do primeiro.
- Retorno....: <-- Nenhum -->
- Parâmetros.: (As lista são passadas por referência)
     1) Lst1 -> Lista qualquer (de objetos TDias).
     2) Pos  -> Posição (nas listas 1 e 2) do "TDias" a ser trocado.
     3) Lst2 -> Lista qualquer (de objetos TDias).
============================================================================ */
void TEscMensal::Movimento(TList **Lst1, int Pos, TList **Lst2)
{
 TList *L1 = *Lst1;
 TList *L2 = *Lst2;
 TDias *Aux1, *Aux2;

 Aux1 = (TDias *)L1->Extract(L1->Items[Pos]);
 Aux2 = (TDias *)L2->Extract(L2->Items[Pos]);

 L1->Insert(Pos, Aux2);
 L2->Insert(Pos, Aux1);
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: EfetuarCalculos
- Objetivo...: Efetuar todos os cálculos para todas as tripulações da lista.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) EstMes -> Objeto que contém a estrutura do mês selecionado.
     2) ParEM  -> Objeto que contém os parâmetros para a escala mensal.
     3) PEM    -> Objeto que contém os pesos para a escala mensal.
============================================================================ */
void TEscMensal::EfetuarCalculos(TEstruturaMes *EstMes, TParPptEM *ParEM, TPesosEM
*PEM)
{
 TTripEM *T;
 this->FOTotTripulacoes = 0;
 this->CalcTmpMedioTrab(EstMes);

 for (int i = 0; i < this->LstTripulacoes->Count; i++)
  {
   T = (TTripEM *)LstTripulacoes->Items[i];
   T->EfetuarCalcTrip(this->TmpMedioTrab, EstMes, ParEM, PEM);
   this->FOTotTripulacoes = this->FOTotTripulacoes + T->FncObjetivo;
  }
 this->EfetuarCalcEM();
}
//---------------------------------------------------------------------------

/* ============================================================================
```

```
- Método.....: EfetuarCalcEM
- Objetivo...: Calcular o valor da função objetivo total da escala mensal.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================== */
void TEscMensal::EfetuarCalcEM()
{
 this->FncObjTotal = this->FOTotTripulacoes;
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: EfCalDepoisDoMovimento
- Objetivo...: Calcular o valor da função objetivo total depois de um movimento.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) EstEM -> Objeto que contém a estrutura do mês selecionado.
     2) ParEM -> Objeto que contém os parâmetros para a escala mensal.
     3) PEM   -> Objeto que contém os pesos para a escala mensal.
     4) T1    -> Nº de uma das tripulações do movimento.
     5) T2    -> Nº da outra tripulação do movimento.
============================================================================== */
void TEscMensal::EfCalcDepoisDoMovimento(TEstruturaMes *EstMes, TParPptEM *ParEM,
TPesosEM *PEM, int T1, int T2)
{
 TTripEM *Trip1, *Trip2;

 int FOT1Antes, FOT1Depois, FOT2Antes, FOT2Depois;

 Trip1 = (TTripEM *)this->LstTripulacoes->Items[T1];
 Trip2 = (TTripEM *)this->LstTripulacoes->Items[T2];

 FOT1Antes = Trip1->FncObjetivo;
 FOT2Antes = Trip2->FncObjetivo;

 this->CalcTmpMedioTrab(EstMes);

 Trip1->EfetuarCalcTrip(this->TmpMedioTrab, EstMes, ParEM, PEM);
 Trip2->EfetuarCalcTrip(this->TmpMedioTrab, EstMes, ParEM, PEM);

 FOT1Depois = Trip1->FncObjetivo;
 FOT2Depois = Trip2->FncObjetivo;

 this->FOTotTripulacoes = this->FOTotTripulacoes - FOT1Antes - FOT2Antes +
FOT1Depois + FOT2Depois;

 this->EfetuarCalcEM();
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcTmpMedioTrab
- Objetivo...: Calcular o tempo médio de trabalho das tripulações durante o mês.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) EM -> Objeto que contém a estrutura do mês selecionado.
============================================================================== */
void TEscMensal::CalcTmpMedioTrab(TEstruturaMes *EM)
{
 TTripEM *T;
 div_t Aux;
 this->TmpMedioTrab = 0;
 for (int i = 0; i < this->LstTripulacoes->Count; i++)
  {
   T = (TTripEM *)this->LstTripulacoes->Items[i];
   T->CalcTmpTTrabalho(EM);
   this->TmpMedioTrab = this->TmpMedioTrab + T->TmpTTrabalho;
  }
 Aux = div(this->TmpMedioTrab, this->LstTripulacoes->Count);
```

```
 this->TmpMedioTrab = Aux.quot;
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: GravarEscala
- Objetivo...: Gravar a escala mensal obtida em no arquivo EsM.txt.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) EM  -> Objeto que contém a estrutura do mês selecionado.
     2) Dir -> Diretório de destino do arquivo gravado.
============================================================================ */
void TEscMensal::GravarEscala(TEstruturaMes *EM, String Dir)
{
 String Diretorio;
 Diretorio = ExtractFilePath(Application->ExeName) + "\\Resultados\\EscM.txt";

 if (Dir == "")
  Dir = Diretorio;

 char *Path = new char[ Diretorio.Length() + 1 ];
 strcpy(Path, Diretorio.c_str());

 FILE *Arq;
 Arq = fopen(Path, "w");
 delete(Path);
 if(!Arq)
  {
   Beep();
   MessageDlg("O arquivo da escala mensal não pode ser aberto!", mtError,
              TMsgDlgButtons() << mbOK, 0);
  }
 else
  {
   char separador1[108] =

"============================================================================================
==========================";
   char separador2[108] =
   "--------------------------------------------------------------------------------------
---------------------------";

   TTripEM *TM;
   TTar_Trip *T;
   TDias *D;
   TList *LstAux;

   String Titulo;
   int i=0;
   int j=0;
   String Aux, NV, HI, HF, PI, PF, FA, LI, LF, TTT;
   int NTTP, NTPT;
   String Data, Tipo;

   fprintf(Arq,"\n\t\t\t\t\t\t\t%s\n\t\t\t\t\t\t\t%s\n\t\t\t\t\t\t\t%s\n\n",
           "******************","*  ESCALA MENSAL  *","******************");

   fprintf(Arq,"\n\t%s\n\t%s%s%d\n\t%s\n\n\n",
           "------------------------------"," MÊS: ", EM->Mes, "     ANO: ", EM-
>Ano, "------------------------------");

   while(i < this->LstTripulacoes->Count)
    {
     Titulo = "TRIPULAÇÃO " + IntToStr(i);

     fprintf(Arq,"%s\n\t\t\t\t\t\t\t%s\n%s\n\n\n",separador1,Titulo,separador1);

     TM = (TTripEM *)this->LstTripulacoes->Items[i];

     while(j < TM->LstDias->Count)
```

```
      {
       D = (TDias *)TM->LstDias->Items[j];

       Data = EM->getData(j);
       if (D->Tipo == 'U')
        Tipo = "Dia útil";
       else if (D->Tipo == 'S')
        Tipo = "Sábado";
       else if (D->Tipo == 'D')
        Tipo = "Domingo";
       else
        Tipo = "Feriado";

       fprintf(Arq,"%s%s\t\t\t%s%s\n%s\n"," Data: ", Data, "Tipo: ", Tipo,
separador2);

       if (D->Numero == -1)
        fprintf(Arq, "\t%s\n%s\n\n\n\n\n", "FOLGA",separador2);
       else
        {
         if (D->Tipo == 'U')
          LstAux = getTarefas(EM, 'U', D->Numero);
         else if (D->Tipo == 'S')
          LstAux = getTarefas(EM, 'S', D->Numero);
         else
          LstAux = getTarefas(EM, 'D', D->Numero);

         fprintf(Arq,"\t\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\n%s\n",

"N_Vei","H_Ini","H_Fim","P_Ini","P_Fim","F_Acu","L_Ini","L_Fim",separador2);

         for (int k = 0; k < LstAux->Count; k++)
          {
           T = (TTar_Trip *)LstAux->Items[k];

           if (T->NumVeic != -1)
            {
             Titulo = "Tarefa " + IntToStr(k);
             NV = IntToStr(T->NumVeic);
             PI = IntToStr(T->PIni);
             PF = IntToStr(T->PFim);
             FA = IntToStr(T->FolAcu);
             LI = T->LinIni;
             LF = T->LinFin;

             HI = MinuteToStr(T->HInicio);
             HF = MinuteToStr(T->HFim);
            }
           else
            {
             Titulo = "T.virtual!";
             NV = "TV!";
             PI = "TV!";
             PF = "TV!";
             FA = "TV!";
             LI = "TV!";
             LF = "TV!";

             HI = MinuteToStr(T->HInicio);
             HF = MinuteToStr(T->HFim);
            }


fprintf(Arq,"%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\t\t%s\n%s\n",Titulo,
                    NV,HI,HF,PI,PF,FA,LI,LF,separador2);
          }
         fprintf(Arq, "\n\n\n");
        }
       j++;
      }
```

```
     j = 0;
     TTT  = MinuteToStr(TM->TmpTTrabalho);
     NTTP = TM->NumTrocaTipPeg;
     NTPT = TM->NumTroPerTrab;

     fprintf(Arq,"%s%s\n%s%d\n%s%d\n%s\n\n\n\n\n",
     "Tempo total trabalhado.....................: ", TTT,
     "N° de trocas de tipo de pegada.............: ", NTTP,
     "N° de trocas do período de trabalho.........: ", NTPT,
     "----------------------------------------------------------------");
     i++;
    }
  }
 fprintf(Arq,"%s", "FIM");
 fclose(Arq);

 char *DDestino = new char[ Dir.Length() + 1 ];
 strcpy(DDestino, Dir.c_str());
 char *DOrigem = new char[ Diretorio.Length() + 1 ];
 strcpy(DOrigem, Diretorio.c_str());

 CopyFile(DOrigem, DDestino, false);
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: GravarArqAux
- Objetivo...: Gravar um arquivo para carregar a escala mensal.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) EstM -> Objeto que contém uma estrutura de um mês.
     2) Dir  -> Diretório a ser gravado o arquivo.
============================================================================ */
void TEscMensal::GravarArqAux(TEstruturaMes *EstM, String Dir)
{
 String Diretorio;
 Diretorio = ExtractFilePath(Application->ExeName) + "\\Resultados\\EscM.aux";

 if (Dir == "")
  Dir = Diretorio;

 char *Path = new char[ Diretorio.Length() + 1 ];
 strcpy(Path, Diretorio.c_str());

 FILE *Arq;
 Arq = fopen(Path, "w");
 delete(Path);
 if(!Arq)
  {
   Beep();
   MessageDlg("O arquivo auxiliar da escala mensal não pode ser aberto!", mtError,
              TMsgDlgButtons() << mbOK, 0);
  }
 else
  {
   TTripEM *TEM;
   TTar_Trip *T;
   TJornada *J;
   TDias *D;

   fprintf(Arq, "\n%s %d%s%d\n", "ESTRUTURA DO MÊS: ", EstM->NumMes, "/", EstM-
>Ano);
   for (int i = 0; i < EstM->LstDias->Count; i++)
    {
     D = (TDias *)EstM->LstDias->Items[i];
     fprintf(Arq, "%s %d %c %d\n", "D ", D->Numero, D->Tipo, D->Qtd);
    }

   fprintf(Arq, "\n\n%s\n\n", "TAREFAS");
```

```
   fprintf(Arq, "\n%s\n", "Dias úteis:");
   for (int i = 0; i < EstM->LstTar_TripDU->Count; i++)
    {
     T = (TTar_Trip *)EstM->LstTar_TripDU->Items[i];
     fprintf(Arq, "%s %d %d %d %d %d %d %d %s %s\n", "T ", T->NumTrip, T-
>NumTar, T->NumVeic, T->HInicio, T->HFim,
             T->PIni, T->PFim, T->FolAcu, T->LinIni, T->LinFin);
    }

   fprintf(Arq, "\n\n");
   fprintf(Arq, "\n%s\n", "Feriados e domingos:");
   for (int i = 0; i < EstM->LstTar_TripDF->Count; i++)
    {
     T = (TTar_Trip *)EstM->LstTar_TripDF->Items[i];
     fprintf(Arq, "%s %d %d %d %d %d %d %d %s %s\n", "T ", T->NumTrip, T-
>NumTar, T->NumVeic, T->HInicio, T->HFim,
             T->PIni, T->PFim, T->FolAcu, T->LinIni, T->LinFin);
    }

   fprintf(Arq, "\n\n");
   fprintf(Arq, "\n%s\n", "Sábados:");
   for (int i = 0; i < EstM->LstTar_TripSab->Count; i++)
    {
     T = (TTar_Trip *)EstM->LstTar_TripSab->Items[i];
     fprintf(Arq, "%s %d %d %d %d %d %d %d %d %s %s\n", "T ", T->NumTrip, T-
>NumTar, T->NumVeic, T->HInicio, T->HFim,
             T->PIni, T->PFim, T->FolAcu, T->LinIni, T->LinFin);
    }

   fprintf(Arq, "\n\n\n%s\n\n", "JORNADAS");

   fprintf(Arq, "\n%s\n", "Dias úteis:");
   for (int i = 0; i < EstM->LstJornDU->Count; i++)
    {
     J = (TJornada *)EstM->LstJornDU->Items[i];

     fprintf(Arq, "%s %c %d %d %d %d %d %d %d %s %s\n", "J ", J->Tipo, J->TmpTrab,
J->HExtra, J->VeiIni, J->VeiFin,
             J->Ocio, J->HInicio, J->HFim, J->LinIni, J->LinFin);
    }

   fprintf(Arq, "\n\n");
   fprintf(Arq, "\n%s\n", "Feriados e domingos:");
   for (int i = 0; i < EstM->LstJornDF->Count; i++)
    {
     J = (TJornada *)EstM->LstJornDF->Items[i];

     fprintf(Arq, "%s %c %d %d %d %d %d %d %d %s %s\n", "J ", J->Tipo, J->TmpTrab,
J->HExtra, J->VeiIni, J->VeiFin,
             J->Ocio, J->HInicio, J->HFim, J->LinIni, J->LinFin);
    }

   fprintf(Arq, "\n\n");
   fprintf(Arq, "\n%s\n", "Sábados:");
   for (int i = 0; i < EstM->LstJornSab->Count; i++)
    {
     J = (TJornada *)EstM->LstJornSab->Items[i];

     fprintf(Arq, "%s %c %d %d %d %d %d %d %d %s %s\n", "J ", J->Tipo, J->TmpTrab,
J->HExtra, J->VeiIni, J->VeiFin,
             J->Ocio, J->HInicio, J->HFim, J->LinIni, J->LinFin);
    }

   fprintf(Arq, "\n\n\n%s\n\n", "JORNADAS DAS TRIPULAÇÕES");

   for (int i = 0; i < this->LstTripulacoes->Count; i++)
    {
     TEM = (TTripEM *)this->LstTripulacoes->Items[i];
     fprintf(Arq, "%s %d\n", "Tripulação: ", i);
     for (int j = 0; j < TEM->LstDias->Count; j++)
```

```
      {
       D = (TDias *)TEM->LstDias->Items[j];
       fprintf(Arq, "%s %d %c %d\n", "D ", D->Numero, D->Tipo, D->Qtd);
      }
     fprintf(Arq, "\n\n");
    }
   fprintf(Arq, "%s", "FIM");
  }
 fclose(Arq);

 char *DDestino = new char[ Dir.Length() + 1 ];
 strcpy(DDestino, Dir.c_str());
 char *DOrigem = new char[ Diretorio.Length() + 1 ];
 strcpy(DOrigem, Diretorio.c_str());

 CopyFile(DOrigem, DDestino, false);
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CarregarEscala
- Objetivo...: Carregar uma escala mensal a partir de um arquivo (.esm).
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) EstM      -> Objeto que contém uma estrutura de um mês.
     2) Diretorio -> Diretório do arquivo selecionado.
============================================================================ */
void TEscMensal::CarregarEscala(TEstruturaMes **EstM, String Diretorio)
{
 TEstruturaMes *EM = *EstM;

 char *Path = new char[ Diretorio.Length() + 1 ];
 strcpy(Path, Diretorio.c_str());

 FILE *Arq;
 Arq = fopen(Path, "r");
 delete(Path);
 if(!Arq)
  {
   Beep();
   MessageDlg("O arquivo auxiliar da escala mensal não pode ser aberto!", mtError,
              TMsgDlgButtons() << mbOK, 0);
  }
 else
  {
   char Linha[150];

   do
    {
     fscanf(Arq, "%s", &Linha);
    }while(strncmp(Linha, "MÊS:", 5));
   fscanf(Arq, "%s", &Linha);

   String Aux = Linha;
   int Mes, Ano;

   if (Aux[2] == '/')
    {
     Mes = StrToInt(Aux.SubString(1, 1));
     Ano = StrToInt(Aux.SubString(3, 4));
    }
   else
    {
     Mes = StrToInt(Aux.SubString(1, 2));
     Ano = StrToInt(Aux.SubString(4, 4));
    }

   EM->Mes    = getMes(Mes);
   EM->NumMes = Mes;
   EM->Ano    = Ano;
```

```
   TDias *D;
   TTar_Trip *T;
   TJornada *J;
   TTripEM *Trip;
   char LinIni[10];
   char LinFin[10];
   char Tipo;

   fscanf(Arq, "%s", &Linha);
   while (strncmp(Linha, "TAREFAS", 8))
    {
     D = new TDias();
     fscanf(Arq, "%d %c %d", &D->Numero, &D->Tipo, &D->Qtd);
     EM->LstDias->Add(D);
     fscanf(Arq, "%s", &Linha);
    }

   fscanf(Arq, "%s%s%s", &Linha, &Linha, &Linha);

   while (!strncmp(Linha, "T", 2))
    {
     T = new TTar_Trip();
     fscanf(Arq, "%d %d %d %d %d %d %d %d %s %s", &T->NumTrip, &T->NumTar, &T-
>NumVeic, &T->HInicio,
            &T->HFim, &T->PIni, &T->PFim, &T->FolAcu, &LinIni, &LinFin);
     T->LinIni = LinIni;
     T->LinFin = LinFin;
     EM->LstTar_TripDU->Add(T);
     fscanf(Arq, "%s", &Linha);
    }

   fscanf(Arq, "%s%s%s", &Linha, &Linha, &Linha);

   while (!strncmp(Linha, "T", 2))
    {
     T = new TTar_Trip();
     fscanf(Arq, "%d %d %d %d %d %d %d %d %s %s", &T->NumTrip, &T->NumTar, &T-
>NumVeic, &T->HInicio,
            &T->HFim, &T->PIni, &T->PFim, &T->FolAcu, &LinIni, &LinFin);
     T->LinIni = LinIni;
     T->LinFin = LinFin;
     EM->LstTar_TripDF->Add(T);
     fscanf(Arq, "%s", &Linha);
    }

   fscanf(Arq, "%s", &Linha);

   while (!strncmp(Linha, "T", 2))
    {
     T = new TTar_Trip();
     fscanf(Arq, "%d %d %d %d %d %d %d %d %s %s", &T->NumTrip, &T->NumTar, &T-
>NumVeic, &T->HInicio,
            &T->HFim, &T->PIni, &T->PFim, &T->FolAcu, &LinIni, &LinFin);
     T->LinIni = LinIni;
     T->LinFin = LinFin;
     EM->LstTar_TripSab->Add(T);
     fscanf(Arq, "%s", &Linha);
    }

   fscanf(Arq, "%s%s%s", &Linha, &Linha, &Linha);

   while (!strncmp(Linha, "J", 2))
    {
     J = new TJornada();
     fscanf(Arq, "%s %d %d %d %d %d %d %d %s %s", &Tipo, &J->TmpTrab, &J->HExtra,
&J->VeiIni,
            &J->VeiFin, &J->Ocio, &J->HInicio, &J->HFim, &LinIni, &LinFin);
     J->Tipo   = Tipo;
     J->LinIni = LinIni;
```

```
     J->LinFin = LinFin;
     EM->LstJornDU->Add(J);
     fscanf(Arq, "%s", &Linha);
    }

   fscanf(Arq, "%s%s%s", &Linha, &Linha, &Linha);

   while (!strncmp(Linha, "J", 2))
    {
     J = new TJornada();
     fscanf(Arq, "%s %d %d %d %d %d %d %d %s %s", &Tipo, &J->TmpTrab, &J->HExtra,
&J->VeiIni,
            &J->VeiFin, &J->Ocio, &J->HInicio, &J->HFim, &LinIni, &LinFin);
     J->Tipo   = Tipo;
     J->LinIni = LinIni;
     J->LinFin = LinFin;
     EM->LstJornDF->Add(J);
     fscanf(Arq, "%s", &Linha);
    }

   fscanf(Arq, "%s", &Linha);

   while (!strncmp(Linha, "J", 2))
    {
     J = new TJornada();
     fscanf(Arq, "%s %d %d %d %d %d %d %d %s %s", &Tipo, &J->TmpTrab, &J->HExtra,
&J->VeiIni,
            &J->VeiFin, &J->Ocio, &J->HInicio, &J->HFim, &LinIni, &LinFin);
     J->Tipo   = Tipo;
     J->LinIni = LinIni;
     J->LinFin = LinFin;
     EM->LstJornSab->Add(J);
     fscanf(Arq, "%s", &Linha);
    }

   do
    {
     fscanf(Arq, "%s", &Linha);
    }while(strncmp(Linha, "Tripulação:", 11));
   do
    {
     fscanf(Arq, "%s", &Linha);
    }while(strncmp(Linha, "D", 2));

   while (strncmp(Linha, "FIM", 4))
    {
     Trip = new TTripEM();
     while (!strncmp(Linha, "D", 2))
      {
       D = new TDias();
       fscanf(Arq, "%d %c %d\n", &D->Numero, &D->Tipo, &D->Qtd);
       Trip->LstDias->Add(D);
       fscanf(Arq, "%s", &Linha);
      }
     this->LstTripulacoes->Add(Trip);
     fscanf(Arq, "%s%s", &Linha, &Linha);
    }
  }
 fclose(Arq);
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: getTarefas
- Objetivo...: Agrupar todas as tarefas de uma tripulação.
- Retorno....: Uma lista com as tarefas.
- Parâmetros.:
     1) EM    -> Objeto que contém a estrutura do mês selecionado.
     2) id    -> Indica se a lista é para dias úteis, sábados ou domingos e
feriados.
```

```
     3) NTrip -> Nº da tripulação.
============================================================================== */
TList *TEscMensal::getTarefas(TEstruturaMes *EM, char id, int NTrip)
{
 TList *L;
 if (id == 'U')
  L = EM->LstTar_TripDU;
 else if (id == 'D')
  L = EM->LstTar_TripDF;
 else
  L = EM->LstTar_TripSab;

 TList *LstAux = new TList();

 TTar_Trip *T;

 for (int i = 0; i < L->Count; i++)
  {
   T = (TTar_Trip *)L->Items[i];
   if (T->NumTrip == NTrip)
    LstAux->Add(T);
  }
 L->Capacity = L->Count;
 return LstAux;
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: Free
- Objetivo...: Destuir o objeto TEscMensal e suas tripulações.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================== */
void TEscMensal::Free()
{
 TTripEM *Trip;
 for (int i = 0; i < this->LstTripulacoes->Count; i++)
  {
   Trip = (TTripEM *)this->LstTripulacoes->Items[i];
   Trip->Free();
  }
 delete(this->LstTripulacoes);
 delete(this);
}
//---------------------------------------------------------------------------

// ---------------------------------- FIM ---------------------------------- \\




/*****************************************************************************
* Sistema..................: PPT_Solver
* Unit.....................: UTar_Jorn                          Tipo.......: .h
* Autor....................: Geraldo Regis Mauri (KÇAPA)
* Última atualização.......: 10 de JANEIRO de 2003
* Objetivo.................: Representar as tarefas e as jornadas.
* Descrição................: Esta unit contém a "declaração" de uma classe
*                            (TTar_Jorn) referente aos atributos comuns entre
*                            tarefas e jornadas, uma outra (TTarefa) referente
*                            às tarefas, outra (TJornada) referente às jornadas
*                            e outra referente a jornadas de uma tripulação.
* Obs......................: O arquivo .cpp dessa unit possui a implementação de
*                            tudo que foi declarado aqui.
* Dependências.............: <--- NENHUMA --->
*****************************************************************************/

#ifndef UTar_JornH
#define UTar_JornH
```

```
// -------------------------- ARQUIVOS "IMPORTADOS" ----------------------- \\
#include "system.hpp"
//---------------------------------------------------------------------------

/* ===========================================================================
- Classe.....: TTar_Jorn
- Objetivo...: Possui os atributos e métodos comuns entre tarefas e jornadas.
=========================================================================== */
class TTar_Jorn{
  public:
    int HInicio;   // Horário de início.
    int HFim;      // Horário de término.
    String LinIni; // Linha inicial.
    String LinFin; // Linha Final.

    void PreencherAtributos(int, int, String, String);
    void Free();
};
//---------------------------------------------------------------------------

/* ===========================================================================
- Classe.....: TTarefa
- Objetivo...: Possui os atributos e métodos referentes às tarefas.
=========================================================================== */
class TTarefa : public TTar_Jorn{
  public:
    int NumVeic; // Número do veículo.
    int PIni;    // Ponto inicial.
    int PFim;    // Ponto final.
    int FolAcu;  // Folga acumulada.

    void PreencherAtributos(int, int, String, String, int, int, int, int);
};
//---------------------------------------------------------------------------

/* ===========================================================================
- Classe.....: TJornada
- Objetivo...: Possui os atributos e métodos referentes às jornadas.
=========================================================================== */
class TJornada : public TTar_Jorn{
  public:
    char Tipo;   // Tipo da jornada (dia útil, sábado ou domingo e feriado).
    int TmpTrab; // Tempo de trabalho (minutos).
    int HExtra;  // Hora extra (minutos).
    int VeiIni;  // Veículo inicial da jornada.
    int VeiFin;  // Veículo final da jornada.
    int Ocio;    // Ociosidade total durante a jornada.

    void PreencherAtributos(int, int, String, String, char, int, int, int, int,
int);
};
//---------------------------------------------------------------------------

/* ===========================================================================
- Classe.....: TTar_Trip
- Objetivo...: Possui os atributos e métodos referentes às tarefas de uma
               tripulação.
=========================================================================== */
class TTar_Trip : public TTarefa{
  public:
    int NumTrip; // Número da tripulação.
    int NumTar;  // Número da tarefa.

    void PreencherAtributos(int, int, String, String, int, int, int, int, int,
int);
};
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
```

```
#endif
// ---------------------------------- FIM ---------------------------------- \\



/*****************************************************************************
* Sistema..................: PPT_Solver
* Unit.....................: UTar_Jorn                          Tipo.......: .cpp
* Autor....................: Geraldo Regis Mauri (KÇAPA)
* Última atualização.......: 10 de JANEIRO de 2003
* Objetivo.................: Representar as tarefas e as jornadas.
* Descrição................: Esta unit contém a "implementação" de uma classe
*                            (TTar_Jorn) referente aos atributos comuns entre
*                            tarefas e jornadas, uma outra (TTarefa) referente
*                            às tarefas, outra (TJornada) referente às jornadas
*                            e outra referente a jornadas de uma tripulação.
* Obs......................: O arquivo .h dessa unit possui a declaração de
*                            tudo que foi implementado aqui.
* Dependências.............: <--- NENHUMA --->
*****************************************************************************/

#pragma hdrstop

#include "UTar_Jorn.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------------------------------


                        // ----- CLASSE TTar_Jorn ----- \\

/* ============================================================================
- Método.....: PreencherAtributos
- Objetivo...: Preencher os valores do objeto com os valores recebidos.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
   1) HI -> Horário de início.
   2) HF -> Horário de término.
   3) LI -> Linha inicial.
   4) LF -> Linha final.
============================================================================ */
void TTar_Jorn::PreencherAtributos(int HI, int HF, String LI, String LF)
{
 HInicio = HI;
 HFim    = HF;
 LinIni  = LI;
 LinFin  = LF;
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: Free
- Objetivo...: Destuir o objeto TTar_Jorn (ou filho).
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================ */
void TTar_Jorn::Free()
{
 delete(this);
}
//---------------------------------------------------------------------------


                        // ----- CLASSE TTarefa ----- \\

/* ============================================================================
- Método.....: PreencherAtributos
```

```
- Objetivo...: Preencher os valores do objeto com os valores recebidos.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
   1) HI -> Horário de início.
   2) HF -> Horário de término.
   3) LI -> Linha inicial.
   4) LF -> Linha final.
   5) NV -> Número do veículo.
   6) PI -> Ponto inicial.
   7) PF -> Ponto final.
   8) FA -> Folga acumulada.
============================================================================== */
void TTarefa::PreencherAtributos(int HI, int HF, String LI, String LF, int NV,
                                 int PI, int PF, int FA)
{
 TTar_Jorn::PreencherAtributos(HI, HF, LI, LF);
 NumVeic = NV;
 PIni    = PI;
 PFim    = PF;
 FolAcu  = FA;
}
//---------------------------------------------------------------------------


                        // ----- CLASSE TJornada ----- \\

/* ============================================================================
- Método.....: PreencherAtributos
- Objetivo...: Preencher os valores do objeto com os valores recebidos.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
   1) HI -> Horário de início.
   2) HF -> Horário de término.
   3) LI -> Linha inicial.
   4) LF -> Linha final.
   5) T  -> Tipo de jornada.
   6) TT -> Tempo trabalhado.
   7) HE -> Hora extra.
   8) VI -> Veículo inicial.
   9) VF -> Veículo final.
  10) O  -> Ociosidade.
============================================================================== */
void TJornada::PreencherAtributos(int HI, int HF, String LI, String LF, char T,
                                  int TT, int HE, int VI, int VF, int O)
{
 TTar_Jorn::PreencherAtributos(HI, HF, LI, LF);
 Tipo    = T;
 TmpTrab = TT;
 HExtra  = HE;
 VeiIni  = VI;
 VeiFin  = VF;
 Ocio    = O;
}
//---------------------------------------------------------------------------


                        // ----- CLASSE TTar_Trip ----- \\

/* ============================================================================
- Método.....: PreencherAtributos
- Objetivo...: Preencher os valores do objeto com os valores recebidos.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
   1) HI   -> Horário de início.
   2) HF   -> Horário de término.
   3) LI   -> Linha inicial.
   4) LF   -> Linha final.
   5) NV   -> Número do veículo.
   6) PI   -> Ponto inicial.
   7) PF   -> Ponto final.
```

```
   8) FA   -> Folga acumulada.
   9) NT   -> Número da tripulação.
  10) NTar -> Número da tarefa.
============================================================================== */
void TTar_Trip::PreencherAtributos(int HI, int HF, String LI, String LF, int NV,
                                   int PI, int PF, int FA, int NT, int NTar)
{
 TTarefa::PreencherAtributos(HI, HF, LI, LF, NV, PI, PF, FA);
 NumTrip = NT;
 NumTar  = NTar;
}
//---------------------------------------------------------------------------

// ---------------------------------- FIM ---------------------------------- \\




/*****************************************************************************
* Sistema..................: PPT_Solver
* Unit.....................: UTripulacoes                      Tipo.......: .h
* Autor....................: Geraldo Regis Mauri (KÇAPA)
* Última atualização.......: 10 de JANEIRO de 2003
* Objetivo.................: Criar estruturas referentes às tripulações.
* Descrição................: Esta unit contém a "declaração" de uma classe
*                            (TTripulacao) referente aos atributos comuns para
*                            as tripulações de ambas as escalas, de uma outra
*                            (TTripED) referente aos atributos das tripulações
*                            para a escala diária e de uma outra (TTripEM)
*                            referente aos atributos das tripulações para a
*                            escala mensal.
* Obs......................: O arquivo .cpp dessa unit possui a implementação de
*                            tudo que foi declarado aqui.
* Dependências.............:
*     1) UEstruturaMes
*     2) UParPpt
*     3) UPesos
*****************************************************************************/

#ifndef UTripulacoesH
#define UTripulacoesH

// --------------------------- ARQUIVOS "IMPORTADOS" ------------------------ \\
#include "classes.hpp"

#include "UEstruturaMes.h"
#include "UParPpt.h"
#include "UPesos.h"
//---------------------------------------------------------------------------


/* ============================================================================
- Classe.....: TTripulacao
- Objetivo...: Possui os atributos e métodos referentes às tripulações.
============================================================================== */
class TTripulacao{
  public:
    int TmpTTrabalho;     // Tempo total de trabalho (minutos).
    int TmpEJornadas;     // Tempo entre jornadas (minutos).
    long int FncObjetivo; // Função objetivo.

    TTripulacao();
};
//---------------------------------------------------------------------------


/* ============================================================================
- Classe.....: TTripED
- Objetivo...: Possui os atributos e métodos referentes às tripulações para a
```

```
               escala diária.
============================================================================== */
class TTripED : public TTripulacao{
  private:
    void CalcTmpTTrabalho();
    void CalcTmpEJornadas();

    void CalcFolAcuTotal();
    void CalcHExtra(TParPptED *);
    void CalcTmpOcioso(TParPptED *);
    void CalcTmpSobreposicao(TParPptED *);
    void CalcNTarefas();
    void CalcNTPProibidas(TParPptED *);
    void CalcNTPPermitidas(TParPptED *);
    void CalcNTVeiculos();
    void CalcDPegada(TParPptED *);
    void InserirTarVirtual(TParPptED *);
    void CalcNTLinhas();

  public:
    int FolAcuTotal;        // Folga acumulada total (minutos).
    int HExtra;             // Hora extra (minutos).
    int TmpOcioso;          // Tempo ocioso (minutos).
    int TmpSobreposicao;    // Tempo de sobreposição (minutos).
    int NTarefas;           // Número de tarefas.
    int NDPegadas;          // Número de duplas pegadas.
    int NTPProibidas;       // Número de trocas de ponto proibidas.
    int NTPPermitidas;      // Número de trocas de ponto permitidas.
    int NTLProibidas;       // Número de trocas de linha proibidas.
    int NTLPermitidas;      // Número de trocas de linha permitidas.
    int NTVeiculos;         // Número de trocas de veículos.
    int MaiorIntEntTarefas; // Maior intervalo entre tarefas (minutos).
    int TVirtual;           // Horário de término da tarefa virtual (caso exista).
    TList *LstTarefas;      // Lista de tarefas.

    TTripED();
    void CalcFncObjetivo(TParPptED *, TPesosED *);
    void EfetuarCalcTrip(TParPptED *, TPesosED *);
    void Free();
};
//---------------------------------------------------------------------------


/* ============================================================================
- Classe.....: TTripEM
- Objetivo...: Possui os atributos e métodos referentes às tripulações para a
               escala mensal.
============================================================================== */
class TTripEM : public TTripulacao{
  private:
    void CalcTmpEJornadas(TEstruturaMes *, TParPptEM *);
    void CalcNumJornDif();
    void CalcNumTrocaTipPeg(TEstruturaMes *);
    void CalcNumTroPerTrab(TEstruturaMes *, TParPptEM *);

  public:
    int NumJornDif;     // Número de jornadas diferentes.
    int NumTrocaTipPeg; // Número de trocas de tipo de pegada.
    int NumTroPerTrab;  // Número de troca de período de trabalho.
    TList *LstDias;     // Lista de dias.

    TTripEM();
    void CalcTmpTTrabalho(TEstruturaMes *);
    void CalcFncObjetivo(int, TParPptEM *, TPesosEM *);
    void EfetuarCalcTrip(int, TEstruturaMes *, TParPptEM *, TPesosEM *);
    void Free();
};
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
```

```
#endif
// ---------------------------------- FIM ---------------------------------- \\



/*****************************************************************************
* Sistema..................: PPT_Solver
* Unit.....................: UTripulacoes                    Tipo.......: .cpp
* Autor....................: Geraldo Regis Mauri (KÇAPA)
* Última atualização.......: 10 de JANEIRO de 2003
* Objetivo.................: Criar estruturas referentes às tripulações.
* Descrição................: Esta unit contém a "implementação" de uma classe
*                            (TTripulacao) referente aos atributos comuns para
*                            as tripulações de ambas as escalas, de uma outra
*                            (TTripED) referente aos atributos das tripulações
*                            para a escala diária e de uma outra (TTripEM)
*                            referente aos atributos das tripulações para a
*                            escala mensal.
* Obs......................: O arquivo .h dessa unit possui a declaração de
*                            tudo que foi implementado aqui.
* Dependências.............:
*     1) UTar_Jorn
*     2) UUtil
*****************************************************************************/

#pragma hdrstop

#include "UTripulacoes.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------------------------------

// -------------------------- ARQUIVOS "IMPORTADOS" ----------------------- \\
#include "UTar_Jorn.h"

#include "UUtil.cpp"
//---------------------------------------------------------------------------

#define max(a, b)  (((a) > (b)) ? (a) : (b))

//---------------------------------------------------------------------------


                        // ----- CLASSE TTripulacao ----- \\

/* ============================================================================
- Método.....: TTripulacao
- Objetivo...: Criar e inicializar o objeto TTripulacao.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================ */
TTripulacao::TTripulacao()
{
 TmpTTrabalho = 0;
 TmpEJornadas = 0;
 FncObjetivo  = 0;
}
//---------------------------------------------------------------------------


                        // ----- CLASSE TTripED ----- \\

/* ============================================================================
- Método.....: TTripED
- Objetivo...: Criar e inicializar o objeto TTripED.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
```

```
============================================================================== */
TTripED::TTripED()
{
 FolAcuTotal        = 0;
 HExtra             = 0;
 TmpOcioso          = 0;
 TmpSobreposicao    = 0;
 NTarefas           = 0;
 NDPegadas          = 0;
 NTPProibidas       = 0;
 NTPPermitidas      = 0;
 NTLProibidas       = 0;
 NTLPermitidas      = 0;
 NTVeiculos         = 0;
 MaiorIntEntTarefas = 0;
 TVirtual           = 0;
 LstTarefas = new TList();
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcTmpTTrabalho
- Objetivo...: Calcular o valor do tempo total de trabalho da tripulação.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================== */
void TTripED::CalcTmpTTrabalho()
{
 TTarefa *T1,*T2;
 this->TmpTTrabalho = 0;

 if (this->LstTarefas->Count != 0)
  {
   T1 = (TTarefa *)this->LstTarefas->First();
   T2 = (TTarefa *)this->LstTarefas->Last();
   this->TmpTTrabalho = (T2->HFim - T1->HInicio) - this->MaiorIntEntTarefas;
  }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcTmpEJornadas
- Objetivo...: Calcular o valor do tempo entre jornadas da tripulação.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================== */
void TTripED::CalcTmpEJornadas()
{
 TTarefa *T1,*T2;
 this->TmpEJornadas = 0;

 if (this->LstTarefas->Count != 0)
  {
   T1 = (TTarefa *)this->LstTarefas->First();
   T2 = (TTarefa *)this->LstTarefas->Last();

   if ((1440 - T2->HFim) + (T1->HInicio) > 0)
    this->TmpEJornadas = (1440 - T2->HFim) + (T1->HInicio);
  }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcFolAcuTotal
- Objetivo...: Calcular o valor da folga acumulada total da tripulação.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================== */
void TTripED::CalcFolAcuTotal()
{
```

```
 TTarefa *T;
 this->FolAcuTotal = 0;

 for (int i=0; i < this->LstTarefas->Count; i++)
  {
   T = (TTarefa *)this->LstTarefas->Items[i];
   this->FolAcuTotal = this->FolAcuTotal + T->FolAcu;
  }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcHExtra
- Objetivo...: Calcular o valor da hora extra da tripulação.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) PED -> Objeto que contém os parâmetros para a escala diária.
============================================================================ */
void TTripED::CalcHExtra(TParPptED *PED)
{
 this->HExtra = 0;

 if (this->NDPegadas == 0)  //Não é Dupla Pegada
  {
   if (this->TmpTTrabalho > (PED->TmpNormalTrab + PED->IntTotRepAli))
    this->HExtra = this->TmpTTrabalho - (PED->TmpNormalTrab + PED->IntTotRepAli);
  }
 else
  if (this->TmpTTrabalho > PED->TmpNormalTrab)
   this->HExtra = this->TmpTTrabalho - PED->TmpNormalTrab;
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcTmpOcioso
- Objetivo...: Calcular o valor do tempo ocioso da tripulação.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) PED -> Objeto que contém os parâmetros para a escala diária.
============================================================================ */
void TTripED::CalcTmpOcioso(TParPptED *PED)
{
 TTarefa *T1,*T2;
 this->TmpOcioso = 0;

 for (int i=0; i < this->LstTarefas->Count; i++)
  {
   if (this->LstTarefas->Items[i] != this->LstTarefas->Last())
    {
     T1 = (TTarefa *)this->LstTarefas->Items[i];
     T2 = (TTarefa *)this->LstTarefas->Items[i+1];

     if ((T2->HInicio - T1->HFim) > 0)
     this->TmpOcioso = this->TmpOcioso + (T2->HInicio - T1->HFim);
    }
  }

 if (this->NDPegadas == 0)  //Não é Dupla Pegada
  this->TmpOcioso = this->TmpOcioso + this->FolAcuTotal +
                    max (0, (PED->TmpNormalTrab + PED->IntTotRepAli) - this-
>TmpTTrabalho);
 else
  this->TmpOcioso = this->TmpOcioso + this->FolAcuTotal - MaiorIntEntTarefas +
                    max (0, PED->TmpNormalTrab - this->TmpTTrabalho);
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcTmpSobreposicao
- Objetivo...: Calcular o valor do tempo de sobreposição da tripulação.
```

```
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) PED -> Objeto que contém os parâmetros para a escala diária.
============================================================================== */
void TTripED::CalcTmpSobreposicao(TParPptED *PED)
{
 TTarefa *T1,*T2;
 int Aux;
 this->TmpSobreposicao = 0;

 for (int i=0; i < this->LstTarefas->Count; i++)
  {
   if (this->LstTarefas->Items[i] != this->LstTarefas->Last())
    {
     T1 = (TTarefa *)this->LstTarefas->Items[i];
     T2 = (TTarefa *)this->LstTarefas->Items[i+1];

     if (T2->HInicio < T1->HFim)
      {
       this->TmpSobreposicao = this->TmpSobreposicao + (T1->HFim - T2->HInicio);

       if (T1->NumVeic != T2->NumVeic)
        this->TmpSobreposicao = this->TmpSobreposicao + PED->TmpTrocaTrip;
      }
     else
      if (T1->NumVeic != T2->NumVeic)
       this->TmpSobreposicao = this->TmpSobreposicao + max(0, PED->TmpTrocaTrip -
(T2->HInicio - T1->HFim));
    }
  }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcNTarefas
- Objetivo...: Calcular o número de tarefas da tripulação.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================== */
void TTripED::CalcNTarefas()
{
 this->NTarefas = this->LstTarefas->Count;
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcNTPProibidas
- Objetivo...: Calcular o número de trocas de ponto proibidas da tripulação.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) PED -> Objeto que contém os parâmetros para a escala diária.
============================================================================== */
void TTripED::CalcNTPProibidas(TParPptED *PED)
{
 bool AchouMInt = false;
 TTarefa *T1,*T2;
 this->NTPProibidas = 0;

 if (this->LstTarefas->Count == 2)
  {
   T1 = (TTarefa *)this->LstTarefas->Items[0];
   T2 = (TTarefa *)this->LstTarefas->Items[1];
   if ( this->MaiorIntEntTarefas <= PED->TmpDPegada && T1->PFim != T2->PIni )
    this->NTPProibidas++;
  }
 else
  for (int i=0; i < this->LstTarefas->Count; i++)
   {
    if (this->LstTarefas->Items[i] != this->LstTarefas->Last())
     {
```

```
      T1 = (TTarefa *)this->LstTarefas->Items[i];
      T2 = (TTarefa *)this->LstTarefas->Items[i+1];

      if (T1->PFim != T2->PIni)
       {
        if ((T2->HInicio - T1->HFim) != this->MaiorIntEntTarefas)
         this->NTPProibidas++;
        else
         if ( (this->MaiorIntEntTarefas > PED->TmpDPegada) && (!AchouMInt) )
          AchouMInt = true;
         else
          this->NTPProibidas++;
       }
     }
   }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcNTPPermitidas
- Objetivo...: Calcular o número de trocas de ponto permitidas da tripulação.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) PED -> Objeto que contém os parâmetros para a escala diária.
============================================================================ */
void TTripED::CalcNTPPermitidas(TParPptED *PED)
{
 bool AchouMInt = false;
 TTarefa *T1,*T2;
 this->NTPPermitidas = 0;

 if (this->LstTarefas->Count == 2)
  {
   T1 = (TTarefa *)this->LstTarefas->Items[0];
   T2 = (TTarefa *)this->LstTarefas->Items[1];
   if ((this->MaiorIntEntTarefas > PED->TmpDPegada) && (T1->PFim != T2->PIni))
    this->NTPPermitidas++;
  }
 else
  for (int i=0; i < this->LstTarefas->Count; i++)
   {
    if (this->LstTarefas->Items[i] != this->LstTarefas->Last())
     {
      T1 = (TTarefa *)this->LstTarefas->Items[i];
      T2 = (TTarefa *)this->LstTarefas->Items[i+1];

      if (((T2->HInicio - T1->HFim) == this->MaiorIntEntTarefas) && (this-
>MaiorIntEntTarefas > PED->TmpDPegada))
       {
        if ((!AchouMInt) && (T1->PFim != T2->PIni))
         {
          this->NTPPermitidas++;
          AchouMInt = true;
         }
       }
     }
   }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcNTVeiculos
- Objetivo...: Calcular o número de trocas de veículos da tripulação.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================ */
void TTripED::CalcNTVeiculos()
{
 TTarefa *T1,*T2;
 this->NTVeiculos = 0;
```

```
 for (int i=0; i < this->LstTarefas->Count; i++)
  {
   if (this->LstTarefas->Items[i] != this->LstTarefas->Last())
    {
     T1 = (TTarefa *)this->LstTarefas->Items[i];
     T2 = (TTarefa *)this->LstTarefas->Items[i+1];

     if (T2->NumVeic != T1->NumVeic)
      this->NTVeiculos++;
    }
  }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcDPegada
- Objetivo...: Calcular se existe dupla pegada e o tempo desta
(MaiorIntEntTarefas).
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) PED -> Objeto que contém os parâmetros para a escala diária.
============================================================================ */
void TTripED::CalcDPegada(TParPptED *PED)
{
 TTarefa *T1,*T2;
 this->NDPegadas = 0;
 this->MaiorIntEntTarefas = 0;

 for (int i=0; i < this->LstTarefas->Count; i++)
  {
   if (this->LstTarefas->Items[i] != this->LstTarefas->Last())
    {
     T1 = (TTarefa *)this->LstTarefas->Items[i];
     T2 = (TTarefa *)this->LstTarefas->Items[i+1];
     if (((T2->HInicio - T1->HFim) > this->MaiorIntEntTarefas) && ((T2->HInicio -
T1->HFim) > PED->TmpDPegada))
      {
       NDPegadas = 1;
       this->MaiorIntEntTarefas = (T2->HInicio - T1->HFim);
      }
    }
  }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: InserirTarVirtual
- Objetivo...: Inserir a tarefa virtual (caso seja necessário) para a tripulação.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) PED -> Objeto que contém os parâmetros para a escala diária.
============================================================================ */
void TTripED::InserirTarVirtual(TParPptED *PED)
{
 TTarefa *T1, *T2, *TF;
 bool TemIntMin = false;

 this->TVirtual = 0;

 if (this->NDPegadas == 0)
  {
   if (this->TmpTTrabalho > PED->TmpNormalTrab)
    {
     if (this->LstTarefas->Count != 0)
      {
       for (int i=0; i < this->LstTarefas->Count; i++)
        {
         if (this->LstTarefas->Items[i] != this->LstTarefas->Last())
          {
```

```
            T1 = (TTarefa *)this->LstTarefas->Items[i];
            T2 = (TTarefa *)this->LstTarefas->Items[i+1];

            if ((T2->HInicio - T1->HFim) >= PED->SubIntContRA)
             TemIntMin = true;
           }
         }// for

        TF = (TTarefa *)this->LstTarefas->Last();

        if (!TemIntMin)
         {
          if (this->TmpOcioso >= PED->IntTotRepAli)
           {
            this->TVirtual = TF->HFim + PED->SubIntContRA;
            this->TmpOcioso = this->TmpOcioso - PED->IntTotRepAli;
            this->TmpTTrabalho = this->TmpTTrabalho + (TVirtual - TF->HFim);
           }
          else
           {
            this->TmpOcioso = this->TmpOcioso + PED->SubIntContRA;
            this->TVirtual = TF->HFim + PED->SubIntContRA + max(0, PED->IntTotRepAli
- this->TmpOcioso);
            this->TmpTTrabalho = this->TmpTTrabalho + (TVirtual - TF->HFim);
            this->TmpOcioso = max(0, this->TmpOcioso - PED->IntTotRepAli);
           }
         }
        else // if (TemIntMin)
         {
          if (this->TmpOcioso >= PED->IntTotRepAli)
           {
            this->TmpOcioso = this->TmpOcioso - PED->IntTotRepAli;
           }
          else
           {
            this->TVirtual = TF->HFim + PED->IntTotRepAli - this->TmpOcioso;
            this->TmpOcioso = 0;
            this->TmpTTrabalho = this->TmpTTrabalho + (TVirtual - TF->HFim);
           }
         }// else
       }// if (this->LstTarefas->Count != 0)
     }// if (this->TmpTTrabalho > PED->TmpNormalTrab)
    else
     {
      this->TmpTTrabalho = this->TmpTTrabalho + PED->IntTotRepAli;
      this->TmpOcioso = this->TmpOcioso - PED->IntTotRepAli;
     }
   }// if (this->NDPegadas == 0)
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcNTLinhas
- Objetivo...: Calcular o número de trocas de linhas permitidas e proibidas.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================ */
void TTripED::CalcNTLinhas()
{
 TTarefa *T1,*T2;
 this->NTLPermitidas = 0;
 this->NTLProibidas = 0;

 for (int i=0; i < this->LstTarefas->Count; i++)
  {
   if (this->LstTarefas->Items[i] != this->LstTarefas->Last())
    {
     T1 = (TTarefa *)this->LstTarefas->Items[i];
     T2 = (TTarefa *)this->LstTarefas->Items[i+1];
```

```
     if (T2->LinIni != T1->LinFin)
      {
       if (CompararGrupo(T1->LinFin, T2->LinIni))
        this->NTLPermitidas++;
       else
        this->NTLProibidas++;
      }
    }
  }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcFncObjetivo
- Objetivo...: Calcular o valor da função objetivo da tripulação.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) ParED -> Objeto que contém os parâmetros para a escala diária.
     2) PED   -> Objeto que contém os pesos para a escala diária.
============================================================================ */
void TTripED::CalcFncObjetivo(TParPptED *ParED, TPesosED *PED)
{
 int FaltaTmpEJor = 0;

 if (NTarefas != 0)
  FaltaTmpEJor = max(0, ParED->TmpMinEntJorn - this->TmpEJornadas);

 FncObjetivo = (PED->PTOcioso * this->TmpOcioso +
                PED->PHExtra * this->HExtra +
                PED->PTTTrabalho * max(0, TmpTTrabalho - ParED->TmpMaxTrab) +
                PED->PTSobreposicao * this->TmpSobreposicao +
                PED->PTPProibidas * this->NTPProibidas +
                PED->PTPPermitidas * this->NTPPermitidas +
                PED->PTVeiculos * this->NTVeiculos +
                PED->PTEJornadas * FaltaTmpEJor +
                PED->PTLProibidas * this->NTLProibidas +
                PED->PTLPermitidas * this->NTLPermitidas);
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: EfetuarCalcTrip
- Objetivo...: Efetuar todos os cálculos da tripulação.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) ParED -> Objeto que contém os parâmetros para a escala diária.
     2) PED   -> Objeto que contém os pesos para a escala diária.
============================================================================ */
void TTripED::EfetuarCalcTrip(TParPptED *ParED, TPesosED *PED)
{
 CalcNTarefas();
 FolAcuTotal = 0;
 HExtra = 0;
 TmpTTrabalho = 0;
 TmpOcioso = 0;
 TmpSobreposicao = 0;
 TmpEJornadas = 0;
 NDPegadas = 0;
 NTPProibidas = 0;
 NTPPermitidas = 0;
 NTVeiculos = 0;
 NTLProibidas = 0;
 NTLPermitidas = 0;
 MaiorIntEntTarefas = 0;
 FncObjetivo = 0;
 TVirtual = 0;

 if (this->NTarefas != 0)
  {
```

```
   CalcDPegada(ParED); // Também calcula o tempo da dupla pegada
(MaiorIntEntTarefas)
   CalcFolAcuTotal();
   CalcTmpSobreposicao(ParED);
   CalcTmpEJornadas();
   CalcNTVeiculos();
   CalcTmpTTrabalho();       // Parcial...
   CalcTmpOcioso(ParED);     // Parcial...
   InserirTarVirtual(ParED); // Contempla o Tempo Ocioso e o Tempo Total de
Trabalho
   CalcHExtra(ParED);
   CalcNTPProibidas(ParED);
   CalcNTPPermitidas(ParED);
   CalcNTLinhas();
   CalcFncObjetivo(ParED, PED);
  }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: Free
- Objetivo...: Destuir o objeto TTripED.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================ */
void TTripED::Free()
{
 TTarefa *T;
 for (int i = 0; i < this->LstTarefas->Count; i++)
  {
   T = (TTarefa *)this->LstTarefas->Items[i];
   T->Free();
  }
 delete(this->LstTarefas);
 delete(this);
}
//---------------------------------------------------------------------------


                        // ----- CLASSE TTripEM ----- \\

/* ============================================================================
- Método.....: TTripEM
- Objetivo...: Criar e inicializar o objeto TTripEM.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================ */
TTripEM::TTripEM()
{
 NumJornDif     = 0;
 NumTrocaTipPeg = 0;
 NumTroPerTrab  = 0;
 LstDias = new TList();
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcTmpTTrabalho
- Objetivo...: Calcular o tempo de trabalho da tripulação.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) EM -> Objeto que contém as listas de jornadas.
============================================================================ */
void TTripEM::CalcTmpTTrabalho(TEstruturaMes *EM)
{
 TDias *D;
 TJornada *J;
 this->TmpTTrabalho = 0;

 for (int i = 0; i < this->LstDias->Count; i++)
```

```
  {
   D = (TDias *)this->LstDias->Items[i];
   if (D->Numero != -1)
    {
     if (D->Tipo == 'U')
      J = (TJornada *)EM->LstJornDU->Items[D->Numero];
     else if (D->Tipo == 'S')
      J = (TJornada *)EM->LstJornSab->Items[D->Numero];
     else
      J = (TJornada *)EM->LstJornDF->Items[D->Numero];

     this->TmpTTrabalho = this->TmpTTrabalho + (D->Qtd * J->TmpTrab);
    }
  }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcTmpEntreJorn
- Objetivo...: Calcular o tempo entre as jornadas.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) EM -> Objeto que contém as listas de jornadas.
     2) PEM -> Objeto que contém os parâmetros para a escala mensal.
============================================================================ */
void TTripEM::CalcTmpEJornadas(TEstruturaMes *EM, TParPptEM *PEM)
{
 TDias *D1, *D2;
 TJornada *J1, *J2;
 this->TmpEJornadas = 0;

 for (int i = 0; i < this->LstDias->Count; i++)
  {
   if (this->LstDias->Items[i] != this->LstDias->Last())
    {
     D1 = (TDias *)this->LstDias->Items[i];
     D2 = (TDias *)this->LstDias->Items[i+1];
     if ( (D1->Numero != -1) && (D2->Numero != -1) )
      {
       if (D1->Tipo == 'U')
        J1 = (TJornada *)EM->LstJornDU->Items[D1->Numero];
       else if (D1->Tipo == 'S')
        J1 = (TJornada *)EM->LstJornSab->Items[D1->Numero];
       else
        J1 = (TJornada *)EM->LstJornDF->Items[D1->Numero];

       if (D2->Tipo == 'U')
        J2 = (TJornada *)EM->LstJornDU->Items[D2->Numero];
       else if (D2->Tipo == 'S')
        J2 = (TJornada *)EM->LstJornSab->Items[D2->Numero];
       else
        J2 = (TJornada *)EM->LstJornDF->Items[D2->Numero];

       if ((1440 - J1->HFim) + (J2->HInicio) > 0)
         this->TmpEJornadas = this->TmpEJornadas + max(0, PEM->TmpMinEntJorn -
((1440 - J1->HFim) + J2->HInicio));
       else
         this->TmpEJornadas = this->TmpEJornadas + PEM->TmpMinEntJorn;
      }
    }
  }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcNumJornDif
- Objetivo...: Calcular o número de jornadas diferentes.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================ */
```

```
void TTripEM::CalcNumJornDif()
{
 TDias *D;
 this->NumJornDif = 0;
 bool Achou;
 int Aux = 0;
 String Num = "";

 TList *LstAux = new TList();
 LstAux->Add((void *)-1);

 for (int i = 0; i < this->LstDias->Count; i++)
  {
   D = (TDias *)this->LstDias->Items[i];
   Achou = false;
   if (D->Numero != -1)
    {
     if (D->Tipo == 'U')
      Num = "-1" + IntToStr(D->Numero);
     else if (D->Tipo == 'S')
      Num = "-2" + IntToStr(D->Numero);
     else
      Num = "-3" + IntToStr(D->Numero);

     for (int j = 1; j < LstAux->Count; j++)
      {
       Aux = (int)LstAux->Items[j];
       if (Num == Aux)
        Achou = true;
      }
     if (!Achou)
      LstAux->Add((void *)StrToInt(Num));
    }
  }

 this->NumJornDif = LstAux->Count;
 delete(LstAux);
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcNumTrocaTipPeg
- Objetivo...: Calcular o número de trocas de tipo de pegada.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) EM -> Objeto que contém as listas de jornadas para dias úteis.
============================================================================ */
void TTripEM::CalcNumTrocaTipPeg(TEstruturaMes *EM)
{
 TDias *D;
 TJornada *J;
 char Tipo;
 this->NumTrocaTipPeg = 0;

 int j = 0;
 do
  {
   D = (TDias *)this->LstDias->Items[j];
   j++;
  }while (D->Tipo != 'U');
 J = (TJornada *)EM->LstJornDU->Items[D->Numero];
 Tipo = J->Tipo;

 for (int i = 0; i < this->LstDias->Count; i++)
  {
   D = (TDias *)this->LstDias->Items[i];
   if (D->Tipo == 'U')
    {
     J = (TJornada *)EM->LstJornDU->Items[D->Numero];
     if (J->Tipo != Tipo)
```

```
      {
       Tipo = J->Tipo;
       this->NumTrocaTipPeg++;
      }
    }
  }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcNumTroPerTrab
- Objetivo...: Calcular o número de trocas de período de trabalho.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) EM  -> Objeto que contém as listas de jornadas.
     2) PEM -> Objeto que contém os parâmetros para a escala mensal.
============================================================================ */
void TTripEM::CalcNumTroPerTrab(TEstruturaMes *EM, TParPptEM *PEM)
{
 TDias *D1, *D2;
 TJornada *J1, *J2;
 int Aux;
 this->NumTroPerTrab = 0;

 for (int i = 0; i < this->LstDias->Count; i++)
  {
   if (this->LstDias->Items[i] != this->LstDias->Last())
    {
     D1 = (TDias *)this->LstDias->Items[i];
     if (D1->Numero != -1)
      {
       Aux = i;
       do
        {
         Aux++;
         D2 = (TDias *)this->LstDias->Items[Aux];
        }while (D2->Numero == -1);

       if (D1->Tipo == 'U')
        J1 = (TJornada *)EM->LstJornDU->Items[D1->Numero];
       else if (D1->Tipo == 'S')
        J1 = (TJornada *)EM->LstJornSab->Items[D1->Numero];
       else
        J1 = (TJornada *)EM->LstJornDF->Items[D1->Numero];

       if (D2->Tipo == 'U')
        J2 = (TJornada *)EM->LstJornDU->Items[D2->Numero];
       else if (D2->Tipo == 'S')
        J2 = (TJornada *)EM->LstJornSab->Items[D2->Numero];
       else
        J2 = (TJornada *)EM->LstJornDF->Items[D2->Numero];

       if ( (J1->HInicio < PEM->HorLTPTrab) && (J2->HInicio >= PEM->HorLTPTrab) ||
            (J1->HInicio >= PEM->HorLTPTrab) && (J2->HInicio < PEM->HorLTPTrab) )
         this->NumTroPerTrab++;
      }
    }
  }
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: CalcFncObjetivo
- Objetivo...: Calcular o valor da função objetivo da tripulação.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) TMT   -> Tempo médio de trabalho na escala mensal.
     2) ParEM -> Objeto que contém os parâmetros para a escala mensal.
     3) PEM   -> Objeto que contém os pesos para a escala mensal.
============================================================================ */
```

```
void TTripEM::CalcFncObjetivo(int TMT, TParPptEM *ParEM, TPesosEM *PEM)
{
 int DifTMT = TMT - this->TmpTTrabalho;
 if (DifTMT < 0)
  DifTMT = DifTMT * (-1);
 int DTMT = max(0, DifTMT - ParEM->DifTMT);

 this->FncObjetivo = (PEM->PNJorD * this->NumJornDif +
                      PEM->PNTTPeg * this->NumTrocaTipPeg +
                      PEM->PTEJornadas * this->TmpEJornadas +
                      PEM->PTMedioTrab * DTMT +
                      PEM->PNTPTrab * this->NumTroPerTrab);
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: EfetuarCalcTrip
- Objetivo...: Efetuar todos os cálculos da tripulação.
- Retorno....: <-- Nenhum -->
- Parâmetros.:
     1) TMT   -> Tempo médio de trabalho na escala mensal.
     1) EM    -> Objeto que contém as listas de jornadas.
     2) ParEM -> Objeto que contém os parâmetros para a escala mensal.
     3) PEM   -> Objeto que contém os pesos para a escala mensal.
============================================================================ */
void TTripEM::EfetuarCalcTrip(int TMT, TEstruturaMes *EM, TParPptEM *ParEM,
TPesosEM *PEM)
{
 CalcTmpEJornadas(EM, ParEM);
 CalcNumJornDif();
 CalcNumTrocaTipPeg(EM);
 CalcNumTroPerTrab(EM, ParEM);
 CalcFncObjetivo(TMT, ParEM, PEM);
}
//---------------------------------------------------------------------------

/* ============================================================================
- Método.....: Free
- Objetivo...: Destuir o objeto TTripEM.
- Retorno....: <-- Nenhum -->
- Parâmetros.: <-- Nenhum -->
============================================================================ */
void TTripEM::Free()
{
 for (int i = 0; i < this->LstDias->Count; i++)
  delete(this->LstDias->Items[i]);
 delete(this->LstDias);
 delete(this);
}
//---------------------------------------------------------------------------

// ---------------------------------- FIM ---------------------------------- \\
```

Anexo B. Saídas do Sistema

Visando mostrar as saídas do sistema, foram colocadas como anexo uma parte da escala diária e uma parte da escala mensal, ambas obtidas nos melhores testes realizados.

Obs: Não foram colocadas as escalas por completo devido aos seus tamanhos.

Escala diária:

```
                        ********************************
                        *  ESCALA DIÁRIA - Dias úteis  *
                        ********************************


================================================================================
                                TRIPULAÇÃO 0
================================================================================
            N_Vei H_Ini       H_Fim       P_Ini P_Fim F_Acu L_Ini L_Fim
--------------------------------------------------------------------------------
Tarefa 0    4     05:20       05:55       0     2     0     201   201
--------------------------------------------------------------------------------
Tarefa 1    4     06:00       06:25       2     2     0     201   201
--------------------------------------------------------------------------------
Tarefa 2    4     06:30       06:55       2     2     0     201   201
--------------------------------------------------------------------------------
Tarefa 3    4     07:00       07:25       2     2     0     201   201
--------------------------------------------------------------------------------
Tarefa 4    4     07:30       07:55       2     2     0     201   201
--------------------------------------------------------------------------------
Tarefa 5    4     08:00       08:25       2     2     0     201   201
--------------------------------------------------------------------------------
Tarefa 6    4     08:30       08:55       2     2     0     201   201
--------------------------------------------------------------------------------
Tarefa 7    4     09:00       09:25       2     2     0     201   201
--------------------------------------------------------------------------------
Tarefa 8    4     09:30       09:55       2     2     0     201   201
--------------------------------------------------------------------------------
Tarefa 9    4     10:00       10:25       2     2     0     201   201
--------------------------------------------------------------------------------
Tarefa 10   4     10:30       10:55       2     2     0     201   201
--------------------------------------------------------------------------------
Tarefa 11   4     11:00       11:25       2     2     0     201   201
--------------------------------------------------------------------------------
Tarefa 12   4     11:30       11:55       2     2     0     201   201
--------------------------------------------------------------------------------
Tarefa 13   4     12:00       12:25       2     2     0     201   201
--------------------------------------------------------------------------------
T.virtual!  TV!   12:25       12:40       TV!   TV!   TV!   TV!   TV!
--------------------------------------------------------------------------------

Tempo trabalhado........: 07:20
Tempo ocioso............: 00:40
Hora extra..............: 00:10
Jornada com pegada única.
--------------------------------------
```

```
================================================================================
                                TRIPULAÇÃO 1
================================================================================
           N_Vei H_Ini       H_Fim       P_Ini P_Fim F_Acu L_Ini L_Fim
--------------------------------------------------------------------------------
Tarefa 0    21    05:11       07:34       0     6     5     2104  2104
--------------------------------------------------------------------------------
Tarefa 1    21    07:44       10:15       6     6     5     2104  2104
--------------------------------------------------------------------------------
Tarefa 2    21    10:36       13:16       6     6     5     2104  2104
--------------------------------------------------------------------------------

Tempo trabalhado........: 08:05
Tempo ocioso............: 00:16
Hora extra..............: 00:55
Jornada com pegada única.
---------------------------------------
```

Continua...

Escala mensal:

```
                          ******************
                          *  ESCALA MENSAL  *
                          ******************


      ------------------------------
       MÊS: Janeiro     ANO: 2003
      ------------------------------


================================================================================
                                TRIPULAÇÃO 0
================================================================================


 Data: De  1  até  3/1/2003              Tipo: Dia útil
--------------------------------------------------------------------------------
           N_Vei H_Ini       H_Fim       P_Ini P_Fim F_Acu L_Ini L_Fim
--------------------------------------------------------------------------------
Tarefa 0    72    05:50       08:20       0     13    5     8207  8207
--------------------------------------------------------------------------------
Tarefa 1    59    09:05       11:40       13    13    5     8207  8207
--------------------------------------------------------------------------------
Tarefa 2    59    11:54       14:29       13    13    5     8207  8207
--------------------------------------------------------------------------------


 Data: 4/1/2003              Tipo: Sábado
--------------------------------------------------------------------------------
           N_Vei H_Ini       H_Fim       P_Ini P_Fim F_Acu L_Ini L_Fim
--------------------------------------------------------------------------------
Tarefa 0    47    03:50       08:29       0     11    10    5201  5201
--------------------------------------------------------------------------------
Tarefa 1    47    08:35       11:15       11    11    5     5201  5201
--------------------------------------------------------------------------------
T.virtual!  TV!   11:15       11:30       TV!   TV!   TV!   TV!   TV!
--------------------------------------------------------------------------------
```

Data: 5/1/2003 Tipo: Domingo

FOLGA

Data: De 6 até 10/1/2003 Tipo: Dia útil

	N_Vei	H_Ini	H_Fim	P_Ini	P_Fim	F_Acu	L_Ini	L_Fim
Tarefa 0	72	05:50	08:20	0	13	5	8207	8207
Tarefa 1	59	09:05	11:40	13	13	5	8207	8207
Tarefa 2	59	11:54	14:29	13	13	5	8207	8207

Data: 11/1/2003 Tipo: Sábado

	N_Vei	H_Ini	H_Fim	P_Ini	P_Fim	F_Acu	L_Ini	L_Fim
Tarefa 0	47	03:50	08:29	0	11	10	5201	5201
Tarefa 1	47	08:35	11:15	11	11	5	5201	5201
T.virtual!	TV!	11:15	11:30	TV!	TV!	TV!	TV!	TV!

Data: 12/1/2003 Tipo: Domingo

FOLGA

Data: De 13 até 17/1/2003 Tipo: Dia útil

	N_Vei	H_Ini	H_Fim	P_Ini	P_Fim	F_Acu	L_Ini	L_Fim
Tarefa 0	72	05:50	08:20	0	13	5	8207	8207
Tarefa 1	59	09:05	11:40	13	13	5	8207	8207
Tarefa 2	59	11:54	14:29	13	13	5	8207	8207

Data: 18/1/2003 Tipo: Sábado

	N_Vei	H_Ini	H_Fim	P_Ini	P_Fim	F_Acu	L_Ini	L_Fim
Tarefa 0	47	03:50	08:29	0	11	10	5201	5201
Tarefa 1	47	08:35	11:15	11	11	5	5201	5201
T.virtual!	TV!	11:15	11:30	TV!	TV!	TV!	TV!	TV!

```
 Data: 19/1/2003             Tipo: Domingo
-------------------------------------------------------------------------
      FOLGA
-------------------------------------------------------------------------

 Data: De  20  até  24/1/2003            Tipo: Dia útil
-------------------------------------------------------------------------
           N_Vei H_Ini      H_Fim       P_Ini P_Fim F_Acu L_Ini L_Fim
-------------------------------------------------------------------------
Tarefa 0    73    06:06      08:43       0     0     5     8207  8207
-------------------------------------------------------------------------
Tarefa 1    88    11:05      13:30       11    11    5     9206  9206
-------------------------------------------------------------------------

 Data: 25/1/2003             Tipo: Sábado
-------------------------------------------------------------------------
           N_Vei H_Ini      H_Fim       P_Ini P_Fim F_Acu L_Ini L_Fim
-------------------------------------------------------------------------
Tarefa 0    47    03:50      08:29       0     11    10    5201  5201
-------------------------------------------------------------------------
Tarefa 1    47    08:35      11:15       11    11    5     5201  5201
-------------------------------------------------------------------------
T.virtual!  TV!   11:15      11:30       TV!   TV!   TV!   TV!   TV!
-------------------------------------------------------------------------

 Data: 26/1/2003             Tipo: Domingo
-------------------------------------------------------------------------
      FOLGA
-------------------------------------------------------------------------

 Data: De  27  até  31/1/2003            Tipo: Dia útil
-------------------------------------------------------------------------
           N_Vei H_Ini      H_Fim       P_Ini P_Fim F_Acu L_Ini L_Fim
-------------------------------------------------------------------------
Tarefa 0    73    06:06      08:43       0     0     5     8207  8207
-------------------------------------------------------------------------
Tarefa 1    88    11:05      13:30       11    11    5     9206  9206
-------------------------------------------------------------------------

Tempo total trabalhado.....................: 193:27
Nº de trocas de tipo de pegada.............: 1
Nº de trocas do período de trabalho.........: 0
-----------------------------------------------------------------



=========================================================================
                              TRIPULAÇÃO 1
=========================================================================

 Data: De  1  até  3/1/2003              Tipo: Dia útil
-------------------------------------------------------------------------
           N_Vei H_Ini      H_Fim       P_Ini P_Fim F_Acu L_Ini L_Fim
-------------------------------------------------------------------------
Tarefa 0    97    05:22      07:42       0     11    5     9206  9206
```

Tarefa 1	97	08:00	10:25	11	11	5	9206	9206
Tarefa 2	97	10:35	13:00	11	11	5	9206	9206

Data: 4/1/2003 Tipo: Sábado

	N_Vei	H_Ini	H_Fim	P_Ini	P_Fim	F_Acu	L_Ini	L_Fim
Tarefa 0	38	04:20	06:30	0	9	5	4150	4150
Tarefa 1	38	06:30	08:47	9	9	5	4150	4150
Tarefa 2	38	08:50	11:05	9	9	5	4150	4150
T.virtual!	TV!	11:05	11:20	TV!	TV!	TV!	TV!	TV!

Data: 5/1/2003 Tipo: Domingo

FOLGA

Data: De 6 até 10/1/2003 Tipo: Dia útil

	N_Vei	H_Ini	H_Fim	P_Ini	P_Fim	F_Acu	L_Ini	L_Fim
Tarefa 0	97	05:22	07:42	0	11	5	9206	9206
Tarefa 1	97	08:00	10:25	11	11	5	9206	9206
Tarefa 2	97	10:35	13:00	11	11	5	9206	9206

Data: 11/1/2003 Tipo: Sábado

	N_Vei	H_Ini	H_Fim	P_Ini	P_Fim	F_Acu	L_Ini	L_Fim
Tarefa 0	38	04:20	06:30	0	9	5	4150	4150
Tarefa 1	38	06:30	08:47	9	9	5	4150	4150
Tarefa 2	38	08:50	11:05	9	9	5	4150	4150
T.virtual!	TV!	11:05	11:20	TV!	TV!	TV!	TV!	TV!

Data: 12/1/2003 Tipo: Domingo

FOLGA

Continua...