DEPARTAMENTO DE COMPUTAÇÃO

D E C O M

Resolução do Problema de Programação de Jogos do Campeonato Brasileiro de Futebol

U F O P

UNIVERSIDADE FEDERAL DE OURO PRETO

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

Resolução do Problema de Programação de Jogos do Campeonato Brasileiro de Futebol

Fabrício Lacerda Biajoli

Orientador: Prof. Marcone Jamilson Freitas Souza

OURO PRETO, MG – BRASIL
DEZEMBRO DE 2003

Fabrício Lacerda Biajoli

Resolução do Problema de Programação de Jogos do Campeonato Brasileiro de Futebol

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. Carlos Frederico M. C. Cavalcante
Doutor pela UFMG – Universidade Federal de Minas Gerais
Departamento de Ciência da Computação, UFOP

MSc. Roberto Carlos Vieira Pontes
Mestre pelo IME – Instituto Militar de Engenharia

OURO PRETO, MG – BRASIL
DEZEMBRO DE 2003

Resumo

Este trabalho apresenta contribuições junto à solução de dois problemas de escalonamento de jogos: programação de jogos para competições que apresentam turno único e programação de jogos para competições que apresentam turno e returno. Com isso procura-se construir uma tabela de jogos entre times que participam de uma competição esportiva realizada em vários locais e ao longo de um determinado conjunto de rodadas, minimizando os deslocamentos dos mesmos no decorrer do campeonato bem como a diferença entre a distância percorrida pelo time que mais se deslocou e a distância percorrida pelo time que menos se deslocou, satisfazendo a um determinado conjunto de restrições.

A estratégia de solução proposta para este problema está baseada no emprego de duas técnicas metaheurísticas de busca local associadas: *Simulated Annealing* e Busca Tabu.

Relata-se uma experiência com a utilização das técnicas mencionadas acima, aplicadas na montagem da tabela de jogos da primeira divisão do Campeonato Brasileiro de Futebol de 2002 e 2003.

Sendo assim, foi implementado um sistema onde as metaheurísticas *Simulated Annealing* e Busca Tabu são aplicadas. Este foi implementado utilizando-se a linguagem de programação C++, com a ferramenta C++ Builder 5.0.

Para melhor tratamento do problema, tentou-se fazer com que o sistema apresentasse também uma boa solução para a interface com os usuários, fazendo assim com que o mesmo seja, além de eficiente, funcional e prático em termos de usabilidade.

Palavras-Chave: Otimização Combinatória, Programação de Jogos, Metaheurísticas, *Simulated Annealing* e Busca Tabu.

Agradecimentos

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

À minha família, pelo apoio e incentivo durante a realização do curso de Ciência da Computação.

Ao Prof. Dr. Marcone Jamilson Freitas Souza, pela orientação e confiança na minha capacidade para a realização deste trabalho, e também pelo companheirismo e amizade.

A todos os professores do DECOM, pelos ensinamentos transmitidos.

Ao Prof. Dr. Luiz Ricardo Pinto pela oportunidade de realizar um trabalho de Iniciação Científica, o qual me despertou o interesse pela área de Pesquisa Operacional.

A CBF – Confederação Brasileira de Futebol – representada pelo Prof. Jorge Tanus Nagem, pelo fornecimento dos dados necessários para a realização deste trabalho.

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

Ao Sr. Roberto Carlos Vieira Pontes, pelo interesse, apoio e valiosa colaboração na correção no texto dessa monografia, contribuindo para a melhoria do trabalho.

A todos os meus amigos que mesmo de longe sempre me deram força.

À República K-Zona e a todos os Zoneiros, pelos aprendizados e pelo companheirismo nesses quatro anos.

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

Sumário

Lista de Figuras

Lista de Tabelas

1. Introdução

1.1. O Problema de Programação de Jogos

Para muitas competições esportivas (tais como futebol, futsal, voleibol, basquetebol etc), onde os jogos são disputados dois a dois e realizados em vários locais ao longo de um determinado período de tempo, há a necessidade de se fazer um escalonamento desses jogos. Esse problema, conhecido na literatura como Problema de Programação de Torneios Esportivos (*Sports Timetabling Problem*), consiste em fazer com que todo time jogue no mínimo uma vez com todos os outros (condição que varia entre as competições) e que todos os times joguem em todas as rodadas com oponentes diferentes (seja em sua sede ou fora dela).

Problemas dessa natureza contêm em geral muitas restrições que devem ser satisfeitas e diferentes objetivos a cumprir, como a minimização dos deslocamentos dos times durante o campeonato, realização de apenas uma partida por time e por dia, realização de determinados jogos em estádios e em datas pré-estabelecidas, número mínimo de partidas consecutivas realizadas na sede do time e fora dela etc.

A geração de escalonamentos satisfatórios, respeitando essas condições e objetivos, é um problema muito difícil de se resolver. A dificuldade de solução desse problema é devida ao grande número de possibilidades a serem analisadas. De acordo com Concílio & Zuben (2002), para uma competição envolvendo n times confrontando-se entre si em turnos completos, o número de combinações possíveis é dado por:

$$(n-1)!(n-3)!(n-5)!...(n-(n-1))!\times 2^{(n-1)\times\frac{n}{2}}$$

Para exemplificar a magnitude do espaço de soluções, para uma competição com 20 participantes há $2{,}9062 \times 10^{130}$ combinações possíveis.

Estudos têm sido realizados para tentar resolver esses problemas com uma variedade de abordagens: programação inteira, linear, programação por restrições, etc. Porém este problema é normalmente abordado por técnicas heurísticas de solução, entre as quais: *Simulated Annealing*, Busca Tabu e Algoritmos Genéticos.

Trata-se, portanto, de um problema de grande importância e interesse prático, uma vez que a tabela afeta não apenas os resultados dos jogos, mas também todo rendimento financeiro da competição. Conforme descrito anteriormente, conseguir um resultado satisfatório é uma tarefa muito difícil, mesmo quando realizada por um especialista ou pela utilização de ferramentas convencionais.

1.2. Objetivos do trabalho

Neste trabalho é proposta a aplicação de dois métodos metaheurísticos para resolução do problema de programação de jogos do campeonato brasileiro: *Simulated Annealing* e Busca Tabu. Com isso pretende-se desenvolver um algoritmo eficiente, capaz de montar uma tabela para tal competição, minimizando os gastos com deslocamento dos clubes no decorrer do campeonato, minimizando a diferença entre a maior e a menor distância percorrida pelos times e respeitando as restrições impostas pelo comitê organizador da competição. Como dados de entrada e de comparação serão utilizados os times e a tabela do campeonato brasileiro de futebol de 2002 para o problema de programação de jogos com turno único, e os times e a tabela do campeonato brasileiro de futebol de 2003 para o problema de programação de jogos com turno e returno.

1.3. Estrutura do Trabalho

Este trabalho está estruturado em seções. Ele se apresenta subdividido em 7 seções, incluindo esta introdução.

Na Seção 2 é apresentada parte da bibliografia consultada (há referências bibliográficas que contribuíram de alguma forma ao trabalho, mas que não são citadas no texto), mostrando como o problema de programação de jogos foi tratado por outros autores. Apresenta-se também um resumo sobre métodos heurísticos e metaheurísticos e a definição dos métodos que serão utilizados neste trabalho: *Simulated Annealing* e Busca Tabu.

A Seção 3 define os problemas abordados, ou seja, o problema de programação de jogos com turno único e o problema de programação de jogos com turno e returno, cada um com suas respectivas particularidades.

A Seção 4 apresenta a metodologia: representação de uma solução, tipos de movimentos realizados nos dois problemas, as estruturas de vizinhança, geração de uma solução inicial, função de avaliação, os métodos *Simulated Annealing* e Busca Tabu aplicados aos problemas e o funcionamento básico do algoritmo.

Na Seção 5 é apresentado o sistema desenvolvido para tentar resolver o Problema de Programação de Jogos.

Na Seção 6 são apresentados os resultados obtidos pelos algoritmos e uma breve discussão sobre eles.

Na seção 7 é apresentada uma conclusão sobre o trabalho desenvolvido, com análise de desempenho do algoritmo para os tipos de campeonato abordados. São feitas também algumas sugestões para trabalhos futuros.

Em “Referências Bibliográficas” apresenta-se a revisão citada no texto da monografia e também algumas bibliografias consultadas que não são citadas diretamente no texto.

2. Revisão Bibliográfica

2.1. Introdução

Segundo Miyashiro et al. (2002), construir uma tabela para uma competição esportiva é uma importante tarefa para os organizadores, pois a tabela afeta não apenas os resultados dos jogos, mas também o rendimento da competição. Visto que a construção manual de uma tabela que atenda a todos os requisitos impostos e que possa ser implantada é muito difícil, a demanda por meios de escalonamentos automatizados vem crescendo.

Esses problemas de escalonamento contêm em geral muitas restrições conflitantes entre si para satisfazer e diferentes objetivos para se otimizar, como a minimização da distância total percorrida pelos times (Bean, J. C. & Birge, J. R. 1980), a realização de apenas um jogo por dia, estádios indisponíveis em datas particulares, número mínimo de dias entre jogos em casa e seus correspondentes fora de casa etc (Hamiez, J. P. & Hao, J. K.).

De acordo com Concílio & Zuben (2000) o problema geração de turnos completos (rodadas cheias) em torneios esportivos apresenta uma explosão combinatória de candidatos à solução, de modo que uma busca exaustiva pela solução ótima representaria um procedimento computacional intratável.

O tratamento de problemas de otimização com este grau de complexidade é conseguido caso se abra mão da solução ótima, em prol da obtenção de uma solução de boa qualidade. Como muitas vezes não se têm condições de saber o quão distante da solução ótima se encontra a solução obtida, a qualidade desta é medida em relação às soluções candidatas anteriormente produzidas pelo processo iterativo, a partir de uma condição inicial (Concílio & Zuben 2000).

Em vista desse fato, este problema é normalmente abordado por técnicas heurísticas de solução. Concílio & Zuben (2000) abordam o problema da montagem da tabela de jogos do Campeonato Paulista de Futebol de 1997 utilizando programação genética. Nemhauser & Trick (1998) tratam o problema da programação de jogos de um torneio de basquete envolvendo 9 times de diferentes localidades dos Estados Unidos utilizando uma combinação de uma técnica de programação inteira e uma técnica enumerativa. Trick (2000) apresenta um método de duas fases para resolver o problema. Na primeira fase, os times são alocados ignorando-se os requisitos de se jogar fora ou dentro de suas sedes. Somente na segunda fase esses requisitos são observados. Costa (1995) desenvolve um método híbrido combinando as técnicas Algoritmos Genéticos e Busca Tabu para resolver um problema de escalonamento de jogos da Liga Americana de Hóquei. Miyashiro et al. (2002) desenvolvem um algoritmo de tempo polinomial para verificar se existe uma solução viável para a realização de um torneio em um certo número de rodadas.

2.2. Métodos de Busca Local

Métodos de busca local são técnicas baseadas na noção de vizinhança e aplicadas em problemas de otimização. De forma mais objetiva, seja S o espaço de pesquisa de um dado problema de otimização e f a função objetivo a minimizar. Uma função N, dependente da estrutura do problema tratado, associa a cada solução viável $s \in S$, sua vizinhança $N(S) \subseteq S$. Cada solução s' de $N(S)$ é chamada de vizinho de s. Denomina-se *movimento* a modificação m que transforma uma solução s em uma nova solução, s', que esteja em sua vizinhança. Tal transformação é representada por $s' \leftarrow s \oplus m$.

Podemos dividir a técnica de busca local em dois tipos diferentes: heurísticas

convencionais e metaheurísticas. As heurísticas convencionais são métodos de busca local que, ao encontrarem um ótimo local, terminam sua execução e aceitam este resultado como solução para o problema. As demais são normalmente desenvolvidas para resolver um problema específico. Pode-se citar os seguintes exemplos de métodos heurísticos convencionais: Método de Descida, Método Randômico de Descida e Método Randômico Não Ascendente – RNA.

Os Métodos Metaheurísticos são métodos de busca local capazes de escapar de ótimos locais. Tal abordagem será mais detalhada nas próximas seções.

De forma generalizada, uma técnica de busca local, começando de uma solução inicial s_0 (a qual pode ser gerada de forma aleatória ou obtida através do emprego de alguma outra técnica), navega pelo espaço de pesquisa, passando, iterativamente, de uma solução para outra que seja sua vizinha.

2.3. Metaheurísticas

As metaheurísticas foram criadas, principalmente, para solucionar problemas de otimização combinatória considerados NP-difíceis, ou seja, de difícil resolução exata, com exceção de problemas de pequenas dimensões. São procedimentos destinados a encontrar uma solução de boa qualidade, eventualmente a ótima, consistindo na aplicação, em cada passo, de uma heurística subordinada, a qual tem que ser modelada para cada problema específico, Ribeiro (1996). As metaheurísticas utilizam buscas locais para obtenção das soluções, e contrariamente às heurísticas convencionais podem escapar de ótimos locais.

As metaheurísticas, assim como as heurísticas convencionais, diferenciam entre si, basicamente, pelas seguintes características:

- Critério de escolha de uma solução inicial;
- Definição da vizinhança N(s) de uma solução s;
- Critério de seleção de uma solução vizinha dentro de N(s);
- Critério de término;

Pode-se citar várias metaheurísticas existentes na literatura, tais como, Busca Tabu, *Simulated Annealing, Greed Randomized Adaptative Search Procedure* (GRASP), *Variable Neighborhood Search* (VNS), Algoritmos Genéticos e outros.

A seguir serão descritas as metaheurísticas que foram utilizadas neste trabalho.

2.4. Simulated Annealing

O *Simulated Annealing* (SA) é um método de busca local que aceita movimentos 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, operação conhecida como recozimento.

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, variação do valor da função objetivo, isto é, $\Delta = f(s') - f(s)$. Se $\Delta < 0$, o método aceita a solução s', que passa a ser a nova solução corrente s. Caso contrário, ou seja, $\Delta \geq 0$, a solução vizinha candidata s' 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 que pioram o valor da função objetivo (soluções de piora).

Inicialmente, a temperatura T assume 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 de se escolher soluções de piora, conseqüentemente, aumenta-se a chance de se escapar de ótimos 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 ótimo 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 SA(f(.), N(.), α, SAmax, T0, s)
1.  s* ← s;              {Melhor solução obtida até o momento}
2.  Iter ← 0;            {Número de iterações na temperatura T}
3.  T ← T0;              {Temperatura corrente};
4.  enquanto ( T > 0) faça
5.      enquanto (Iter < SAmax) faça
6.          Iter ← Iter + 1;
7.          Gere um vizinho qualquer s` ∈ N(s);
8.          Δ = f(s`) – f(s);
9.          se (Δ < 0 ) então
10.             s ← s`;
11.             se ( f(s`) < f(s*) ) então s* ← s`;
12.         senão
13.             Tome x ∈ [0, 1];
14.             se ( x < e^(-Δ/T)) então s ← s`;
15.         fim-se
16.     fim-enquanto
17.         T ← α * T;
18.         Iter ← 0;
19. fim-enquanto
20. s ← s*;
21. Retorne s;
Fim SA
```

Figura 2.1 – Algoritmo *Simulated Annealing*

2.5. Busca Tabu

São apresentados, a seguir, os princípios básicos da Busca Tabu (BT), técnica originada nos trabalhos de Fred Glover (1986) e Pierre Hansen (1986). Para um estudo mais detalhado desta técnica referencia-se: Glover (1986, 1989, 1990), F. Glover & M. Laguna (1993, 1997), D. De Werra (1989), A. Hertz & D. De Werra (1990).

A Busca Tabu é um procedimento adaptativo que utiliza uma estrutura de memória para guiar um método de descida a continuar a exploração do espaço de soluções mesmo na ausência de movimentos de melhora, evitando que haja a formação de ciclos, isto é, o retorno a um ótimo local previamente visitado. Mais especificamente, começando com uma solução inicial s_0, um algoritmo BT explora, a cada iteração, um subconjunto V da vizinhança $N(s)$ da solução corrente s. O membro s_0 de V com menor valor nessa região segundo a função $f(.)$ torna-se a nova solução corrente mesmo que s_0 seja pior que s, isto é, que $f(s_0) > f(s)$. O critério de escolha do melhor vizinho é utilizado para escapar de um mínimo local. Esta estratégia, entretanto, pode fazer com que o algoritmo entre num estado de ciclagem, isto é, que retorne a uma solução já gerada anteriormente. De forma a evitar que isto ocorra, existe uma lista tabu T, a qual é uma lista de movimentos proibidos. A lista tabu clássica contém os movimentos reversos aos últimos |T| movimentos realizados (onde |T| é um parâmetro do método) e funciona como uma fila de tamanho fixo (FIFO – First In First Out), isto é, quando um novo movimento é adicionado à lista, o mais antigo sai. Assim, na exploração do subconjunto V da vizinhança $N(s)$ da solução corrente s, ficam excluídos da busca os vizinhos s_0 que são obtidos de s por movimentos m que constam na lista tabu.

Se por um lado, a lista tabu reduz o risco de ciclagem (uma vez que ela garante o não retorno, por |T| iterações, a uma solução já visitada anteriormente); por outro, ela também pode proibir movimentos para soluções que ainda não foram visitadas (D. De Werra 1989). Assim, existe também uma função de aspiração, que é um mecanismo que retira, sob certas circunstâncias, o status tabu de um movimento. Mais precisamente, para cada possível valor v da função objetivo existe um nível de aspiração $A(v)$: uma solução s_0 em V pode ser gerada se $f(s_0) \leq A(f(s))$, mesmo que o movimento m esteja na lista tabu. A função de aspiração A é tal que, para cada valor v da função objetivo, retorna outro valor $A(v)$, que representa o valor que o algoritmo aspira ao chegar de v. Considerando uma função objetivo de valores inteiros, um exemplo simples de aplicação desta idéia é considerar $A(f(s)) = f(s^*) - 1$ onde s^* é a melhor solução encontrada até então. Neste caso, aceita-se um movimento tabu somente se ele conduzir a um vizinho melhor que s^*.

Duas regras são normalmente utilizadas de forma a interromper o procedimento. Pela primeira, pára-se quando é atingido um certo número máximo de iterações sem melhora no valor da melhor solução. Pela segunda, quando o valor da melhor solução chega a um limite inferior conhecido (ou próximo dele). Esse segundo critério evita a execução desnecessária do algoritmo quando uma solução ótima é encontrada ou quando uma solução é julgada suficientemente boa.

Os parâmetros principais de controle do método de Busca Tabu são a cardinalidade |T| da lista tabu, a função de aspiração A, a cardinalidade do conjunto V de soluções vizinhas testadas em cada iteração e *BTmax*, o número máximo de iterações sem melhora no valor da melhor solução.

Apresenta-se, pela Figura 2.2, o pseudocódigo de um algoritmo de Busca Tabu básico. Neste procedimento *fmin* é o valor mínimo conhecido da função f, informação essa que em alguns casos está disponível.

É comum em métodos de Busca Tabu incluir estratégias de intensificação, as quais têm por objetivo concentrar a pesquisa em determinadas regiões consideradas promissoras.

Uma estratégia típica é retornar a soluções já visitadas para explorar sua vizinhança de forma mais efetiva. Outra estratégia consiste em incorporar atributos das melhores soluções já encontradas durante o progresso da pesquisa e estimular componentes dessas soluções a tornar parte da solução corrente. Nesse caso, são consideradas livres no procedimento de busca local apenas as componentes não associadas às boas soluções, permanecendo as demais componentes fixas. Um critério de término, tal como um número fixo de iterações, é utilizado para encerrar o período de intensificação.

Métodos baseados em Busca Tabu incluem, também, estratégias de diversificação. O objetivo dessas estratégias, que tipicamente utilizam uma memória de longo prazo, é redirecionar a pesquisa para regiões ainda não suficientemente exploradas do espaço de soluções. Estas estratégias procuram, ao contrário das estratégias de intensificação, gerar soluções que têm atributos significativamente diferentes daqueles encontrados nas melhores soluções obtidas. A diversificação, em geral, é utilizada somente em determinadas situações, como, por exemplo, quando dada uma solução *s*, não existem movimentos *m* de melhora para ela, indicando que o algoritmo já exauriu a análise naquela região. Para escapar dessa região, a idéia é estabelecer uma penalidade $w(s,m)$ para uso desses movimentos. Um número fixo de iterações sem melhora no valor da solução ótima corrente é, em geral, utilizado para acionar essas estratégias.

Métodos de Busca Tabu incluem, também, listas tabu dinâmicas (F. Dammeyer & S. Vob 1993, A. Schaefer 1996), muitas das quais atualizadas de acordo com o progresso da pesquisa (R. Battiti & G. Tecchiolli 1994, R. Battiti 1996, M. P. Bastos & C.C. Ribeiro 1999). A grande vantagem de se usar uma lista tabu de tamanho dinâmico é que se minimiza a possibilidade de ocorrência de ciclagem.

```
Procedimento BT(f(.), N(.), A(.), |V|, f_min, |T|, BTmax, s)
1.  s* ← s;                      {Melhor solução obtida até o momento}
2.  Iter ← 0;                    {Contador do número de iterações}
3.  MelhorIter ← 0;              {Iteração mais recente que forneceu s*}
4.  T ← ∅;                       {Lista Tabu}
5.  Inicialize a função de aspiração A;
6.  enquanto ( f(s) > f_min  e Iter – MelhorIter < BTmax ) faça
7.      Iter ← iter + 1;
8.      Seja s` ← s ⊕ m o melhor elemento de V ⊂ N(s) tal que
                    o movimento m não seja tabu ( m ∉ T) ou
                    s` atenda a condição de aspiração ( f(s`) < A(f(s)) );
9.      T ← T – {movimento mais antigo} + {movimento que gerou s`};
10.     Atualize a função de aspiração A;
11.     s ← s`;
12.     se ( f(s) < f(s*) ) então
13.         s* ← s;
14.         MelhorIter ← Iter;
15.     fim-se
16. fim-enquanto
17. s ← s*;
18. Retorne s;
Fim BT
```

Figura 2.2 – Algoritmo de Busca Tabu

3. Os Problemas de Programação de Jogos Abordados

3.1. Introdução

Realizar escalonamento de jogos para competições esportivas é uma tarefa que exige bastante tempo e estudo dos profissionais que lidam com este tipo de problema. Isso se deve pelo fato de que cada problema apresenta restrições a serem atendidas e objetivos diferentes a serem cumpridos.

Os problemas abordados neste trabalho apresentam uma série de restrições que devem ser obedecidas, sendo essas definidas como *essenciais*, ou seja, caso não forem atendidas, a tabela gerada é considerada inviável e não poderá ser praticada na competição. Eles possuem também algumas restrições consideradas *não essenciais*, ou seja, aquelas que devem ser satisfeitas sempre que for possível.

O principal objetivo é realizar a montagem de dois tipos de tabela referentes ao Campeonato Brasileiro de Futebol, respeitando as restrições definidas pela Confederação Brasileira de Futebol (CBF). O primeiro tipo de tabela é a que os times se enfrentam entre si em turno único. O segundo tipo é a que os times se enfrentam em turno e returno.

Além da montagem de uma tabela viável, deve-se minimizar os deslocamentos dos times durante o campeonato e a diferença entre a maior e a menor distância percorrida pelos times.

3.2. Descrição do Problema com Turno Único

Competições esportivas disputadas dois a dois e de turno único são aquelas onde cada time joga com cada um dos outros uma única vez.

O problema correspondente, que será abordado, é o do Campeonato Brasileiro de 2002. Neste problema há 26 times que devem se enfrentar em um número de rodadas que seja maior que 25 (mínimo possível de rodadas) e menor que 29 (número de rodadas da tabela oficial de 2002), satisfazendo uma série de requisitos impostos pela Confederação Brasileira de Futebol (CBF), entre os quais:

1) Um time não pode jogar mais de uma vez na mesma rodada;
2) O campeonato deve ser realizado em turno único, isto é, cada time só pode jogar uma única vez com cada um dos outros times durante o campeonato, seja em sua sede ou fora dela;
3) Nas duas primeiras rodadas que cada time participar, um jogo deve ser realizado em sua sede e outro fora de sua sede. Por exemplo, se na primeira rodada o confronto de um time for dentro de casa, então na segunda rodada o confronto deste time deve ser fora de casa;
4) As duas últimas rodadas que cada time participar devem ter a mesma configuração de seus dois primeiros confrontos. Exemplificando, se os dois primeiros confrontos de um time foram jogar fora de casa e depois em casa, respectivamente, então os dois últimos confrontos desse time devem ser, respectivamente, jogar fora de casa e depois em casa;
5) O número de jogos realizados fora de casa deve ser igual ao número de jogos em casa. Quando o número de jogos for ímpar, o time com melhor classificação no ranking da CBF, joga uma partida a mais em casa;
6) Não pode haver jogos entre clubes do mesmo estado na última rodada;
7) Evitar que um time jogue mais de duas vezes consecutivas em casa;

8) Evitar que um time jogue mais de duas vezes consecutivas fora de casa.

Os seis primeiros requisitos são considerados *essenciais* e devem ser atendidos para que a tabela possa ser praticada na competição. Os dois últimos requisitos (7 e 8) são considerados *não essenciais*, ou seja, devem ser atendidos sempre que for possível.

3.3. Descrição do Problema com Turno e Returno

Quando os times de uma competição jogam duas vezes com cada um dos outros, sendo que um dos jogos é realizado em sua sede e o outro é fora dela e em turnos diferentes, dizemos que a competição é realizada em turno e returno. O Campeonato Brasileiro de Futebol de 2003 foi organizado nesse formato.

O problema de programação de jogos com turno e returno é parecido com o problema de programação de jogos com turno único, porém apresenta algumas particularidades. No problema abordado 24 times devem se enfrentar duas vezes, sendo um jogo em casa e outro fora de casa, ambos realizados em turnos diferentes. Para a resolução desse problema são necessárias 46 rodadas (duas vezes o número mínimo de rodadas possíveis). Os seguintes requisitos impostos pela CBF devem ser satisfeitos:

1) Um time não pode jogar mais de uma vez na mesma rodada;
2) O campeonato deve ser realizado em turno e returno, isto é, cada time deve jogar duas vezes com cada um dos outros times durante o campeonato, sendo um jogo dentro de sua sede e o outro fora dela e os jogos em turnos diferentes;
3) Nas duas primeiras rodadas, do turno e returno, que cada time participar, um jogo deve ser realizado em sua sede e outro fora. Por exemplo, se na primeira rodada do turno o confronto de um time for dentro de casa, então na segunda rodada o confronto deste time deve ser fora de casa;
4) As duas últimas rodadas do turno, que cada time participar, devem ter a mesma configuração de seus dois primeiros confrontos. Exemplificando, se os dois primeiros confrontos de um time (turno) foram jogar fora de casa e depois em casa, respectivamente, então os dois últimos confrontos desse time (turno) devem ser, respectivamente, jogar fora de casa e depois em casa. A mesma regra vale para a fase de returno;
5) Não pode haver jogos entre clubes do mesmo estado na última rodada;
6) O número de jogos realizados fora de casa deve ser igual ao número de jogos em casa dentro de cada turno. Quando o número de jogos for ímpar, a diferença entre o número de jogos dentro de fora de casa deve ser no máximo um;
7) Evitar que um time jogue mais de duas vezes consecutivas em casa;
8) Evitar que um time jogue mais de duas vezes consecutivas fora de casa;
9) As rodadas do returno devem ter a mesma configuração das rodadas do turno, porém com os mandos de campo invertidos e com exceção para as duas primeiras e as duas últimas rodadas;
10) A duas primeiras e as duas últimas rodadas do returno devem seguir as seguintes regras:
    a. A primeira e segunda rodada do returno devem ter a mesma configuração da segunda e primeira rodada do turno, respectivamente, porém com os mandos de campo trocados;
    b. A penúltima e última rodada do returno devem ter a mesma configuração da última e penúltima rodada do turno, respectivamente, porém com os mandos de campo trocados;

Os oito primeiros requisitos devem ser atendidos para que a tabela possa ser praticada na competição. Os requisitos 7 e 8 não são essenciais, e devem ser atendidos sempre que for possível. Note que o número de jogos disputados em casa será sempre igual ao número de jogos disputados fora de casa, porém é necessário manter esse equilíbrio dentro de cada turno. Os requisitos 9 e 10 não estão diretamente ligados à montagem da tabela, no diz respeito à viabilidade ou não da mesma, mas apenas na sua apresentação e no cálculo da distância percorrida pelos times.

4. Metodologia

4.1. Representação

Uma tabela, ou solução $s = (s_{ij})$ do problema, é representada por uma matriz com tantas linhas quantas forem às rodadas da competição e com tantas colunas quantos forem os confrontos por rodada. Como o campeonato de 2002 contou com a participação de 26 times que se confrontaram em 29 rodadas, então uma solução s para este problema é uma matriz com 29 linhas e 13 colunas. Na realidade bastavam 25 rodadas para a realização de todos os jogos, lembrando que, caso a tabela do campeonato tivesse 25 rodadas, todos os times deveriam ter jogado em todas as rodadas. No entanto, optou-se por considerar um máximo de 29 rodadas para efeito de comparação de resultados, não necessitando todas elas estarem preenchidas. Nesta matriz cada célula s_{ij} representa um possível jogo. Um jogo do tipo A x B (onde A tem mando de campo) ou B x A (onde B tem mando de campo) é representado unicamente por "(sinal) A x B", sendo o mando de campo definido pelo sinal. Quando positivo, o mando de campo é do time A e, quando negativo, o mando é do time B. Ou seja, o jogo A x B é representado por (+A x B) e o jogo B x A, por (-A x B). A Figura 4.1 ilustra um fragmento de uma escala.

Rodada/Jogo	1	...	13
1	+ A x B		- C x D
...			
29	-B x D		+ A x C

Figura 4.1 – Representação de uma escala de jogos para o problema de turno único

Deve ser observado que, como há um máximo de 13 jogos a serem realizados em cada uma das 29 rodadas possíveis, tem-se 377 células s_{ij} disponíveis para representar os 325 jogos possíveis. Dado este fato temos dois possíveis casos. O primeiro deles seria uma tabela contendo 25 rodadas preenchidas, o que significaria que todos os times jogam em todas as rodadas. O outro caso seria uma tabela com o número de rodadas maior que 26 e menor que 29. Em outras palavras, poderá ocorrer o caso em que nem todas as rodadas serão completas, isto é, haverá rodadas em que alguns times não participarão.

O campeonato de 2003 foi realizado em dois turnos. Isso quer dizer que, cada um dos times joga duas vezes com cada um dos outros, sendo um jogo realizado no primeiro turno e outro no segundo turno. Esse campeonato contou com a participação de 24 times que se confrontaram em 46 rodadas. Então uma solução s para este problema é uma matriz com 46 linhas e 12 colunas. É importante observar que neste campeonato todos os times jogaram em todas as rodadas, obtendo-se como resultado uma tabela completa, ou seja, uma tabela com o número mínimo de rodadas possível e todas as rodadas completas (12 jogos realizados em cada rodada). Assim como na representação apresentada para o problema do campeonato de 2002, cada célula da matriz representa um possível jogo, definidos da mesma forma que no problema anterior. A figura 4.2 ilustra um fragmento de uma solução para o problema.

	Rodada/Jogo	1	...	12
Turno	1	+ A x B		- C x D
	2	- D x E		+ A x C
	3	- A x D		- B x C
	...			
	22	+ A x E		+ B x D
	23	- B x E		- A x F
Returno	24	+ D x E		- A x C
	25	- A x B		+ C x D
	26	+ A x D		+ B x C
	...			
	45	+ B x E		+ A x F
	46	- A x E		- B x D

Figura 4.2 – Representação de uma escala de jogos para o problema de dois turnos

Nesta representação há um máximo de 12 jogos a serem realizados em cada uma das 46 rodadas, tendo 552 células s_{ij} disponíveis para representar os 552 jogos possíveis, ou seja, se confirma o fato de que todo time joga em toda rodada.

4.2. Tipos de movimentos

4.2.1. Para o Problema com Turno Único

Foram definidos dois tipos de movimentos para se obter a vizinhança *N(s)* de uma dada escala *s*. O primeiro caracteriza-se pela mudança do mando de campo, ou seja, pela simples troca de sinal do valor que representa o primeiro time de um dado jogo. Para exemplificar seja – A x B um jogo escolhido aleatoriamente. A realização do primeiro movimento sobre este jogo resultaria na seguinte configuração: + A x B. Observe que o jogo continua o mesmo, porém o mando de campo que era do time B passou a ser do time A.

O segundo movimento é caracterizado pela troca de jogos entre as rodadas, ou seja, são escolhidos aleatoriamente dois jogos e estes trocados de rodada. Por exemplo, sejam os jogos – A x B e + C x D realizados nas rodadas 1 e 2, respectivamente. A aplicação do segundo movimento nesses dois jogos resultaria no jogo + C x D sendo realizado na primeira rodada e o jogo – A x B realizado na segunda rodada. Observe que os mandos de campo não foram alterados, porém os jogos das rodadas foram trocados, resultando em uma configuração diferente.

4.2.2. Para o Problema com Turno e Returno

Para o problema com dois turnos (turno e returno) os movimentos definidos são os mesmos que foram definidos para o problema com turno único. A diferença é que estes movimentos são aplicados apenas nas rodadas do primeiro turno, ou seja, nas 23 primeiras rodadas. Isso se deve pelo fato de que nesse campeonato os jogos do returno, ou segundo turno, são realizados de tal forma que a tabela do returno seja o espelho da tabela do turno. Então se um jogo – A x B é realizado na décima rodada do turno, este mesmo jogo será realizado na décima rodada do returno, porém com o mando de campo trocado, resultando na seguinte configuração: + A x B.

4.3. Estruturas de vizinhança

4.3.1. Para o Problema com Turno Único

Uma escala, ou solução s', é dita vizinha de *s* se for obtida desta a partir de um movimento ou de mudança de mando de campo ou de troca de jogos entre rodadas. A partir desse conceito temos que a estrutura de vizinhança para esses problemas pode ser obtida de duas formas. Uma delas seria a aplicação do movimento de troca de mando de campo. A figura 4.3 ilustra um vizinho s' obtido a partir desse tipo de movimento. Como pode ser observado na escala *s*, no jogo 1 da primeira rodada o mando de campo é do time A, enquanto que na escala *s'* o mando de campo passou para o time B.

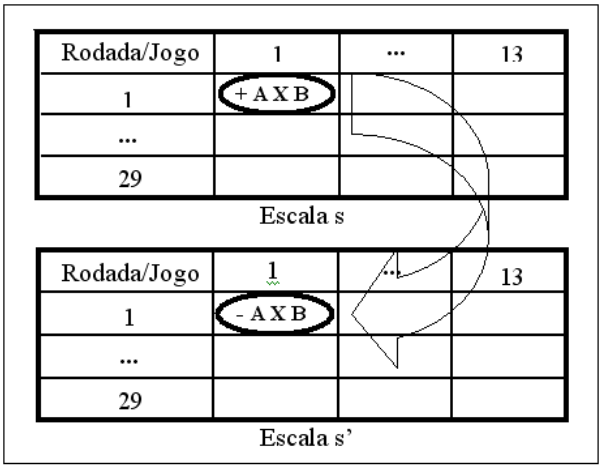


Figura 4.3 – Vizinho s' obtido pelo movimento "troca de mando de jogo"

Um vizinho s' obtido pela aplicação do movimento de troca de jogos entre rodadas pode ser visto na figura 4.4. Como pode ser observado, o confronto entre os times A e B, realizado na primeira rodada da escala *s*, passa a ser realizado na ultima da rodada escala *s'*. Situação inversa ocorre com o confronto entre os times C e D.

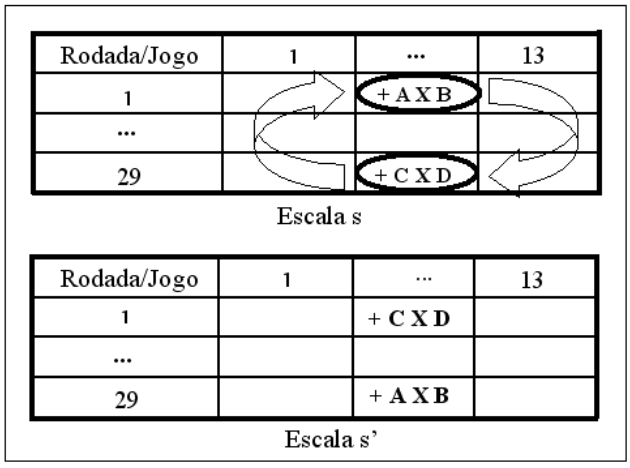


Figura 4.4 – Vizinho s' obtido pelo movimento "troca de jogos entre rodadas"

4.3.2. Para o Problema com Turno e Returno

Para o problema com turno e returno aplicam-se os mesmos movimentos aplicados no problema com turno único, sendo eles: troca de mando de campo de um jogo e troca de jogos entre rodadas. É importante ressaltar que, para esse tipo de problema, os movimentos são aplicados no primeiro turno (turno), ou seja, nas primeiras 23 rodadas. Conforme já mencionado antes, isso se deve pelo fato de que a configuração do segundo turno (ou returno) é o espelho do primeiro turno, isto é, um jogo realizado na quinta rodada do turno será realizado na quinta rodada do returno, porém com mando de campo trocado.

A figura 4.5 ilustra um vizinho s' obtido pela aplicação do movimento de troca de mando de campo. Como pode ser observado na escala *s*, no jogo 1 da terceira rodada o mando de campo é do time D, enquanto que na escala *s'* o mando de campo passou para o time A.

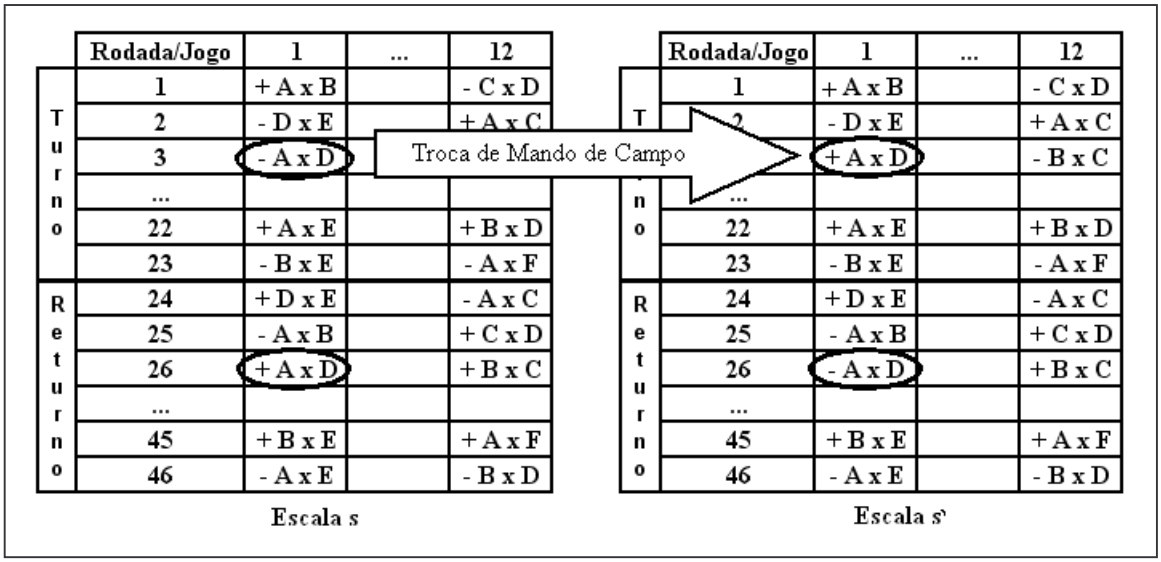


Figura 4.5 – Vizinho s' obtido pelo movimento "troca de mando de jogo"

A figura 4.6 ilustra um vizinho s' obtido pela aplicação do movimento de troca de jogos entre rodadas. Observe que o confronto + A x B, realizado na primeira rodada da escala *s*, passa a ser realizado na terceira rodada da escala *s'*. Situação inversa ocorre com o confronto – A x D. É interessante observar que, a alteração desses dois jogos no primeiro turno resulta na alteração de outros dois jogos, sendo esses os seus respectivos "espelhos" no returno (ver rodadas 25 e 26 da escala s').

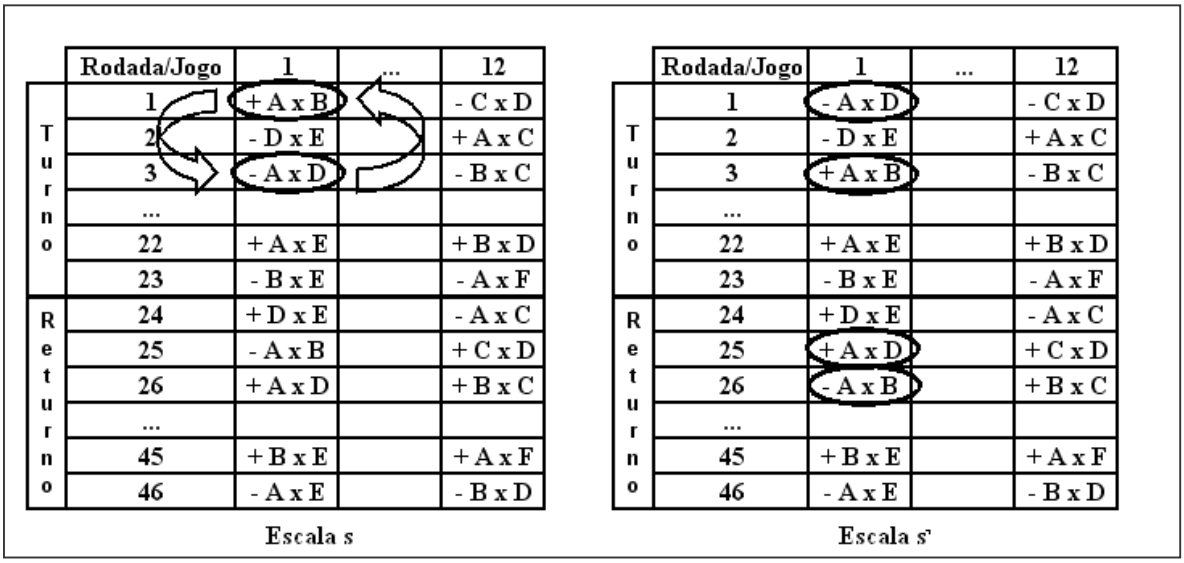


Figura 4.6 – Vizinho s' obtido pelo movimento "troca de jogos entre rodadas"

4.4. Geração de uma solução inicial

A solução inicial é obtida de forma aleatória, conforme a seguir se descreve. Inicialmente deve já estar construída uma lista L de todos os possíveis confrontos, sorteando o mando de campo. A seguir, para cada jogo de cada rodada escolhe-se aleatoriamente um confronto da lista L, alocando-o à rodada. A lista L é, então, atualizada para que os confrontos sorteados não sejam escolhidos novamente. Este procedimento prossegue enquanto a lista L for não vazia. Como parâmetros este algoritmo necessita da lista L de confrontos, da matriz Tabela que representa uma solução e da variável rMax, que é o número máximo de rodadas a serem preenchidas. Para o problema com turno único o rMax tem valor 29 e para o problema com turno e returno o rMax tem valor 23 (valor referente às rodadas do turno, pois as rodadas do returno são o seu espelho). A figura 4.7 apresenta o algoritmo desenvolvido.

```
Procedimento Constroi_Solucao_Inicial(L[.], Tabela[.][.], rMax)
1.  r ← 1;          {Representa a rodada a ser preenchida}
2.  j ← 1;          {Representa o jogo a ser preenchido}
3.  enquanto ( L <> ∅ ) faça
4.      jogo ← confronto sorteado da lista L de confrontos;
5.      Tabela[r][j] ← jogo;
6.      Atualizar lista L;
7.      if ( r < rMax) então  r ← r + 1;
8.      senão
9.              j ← j + 1;
10.             r ← 1;
11.     fim-se
12. fim-enquanto
13. Retorne  Tabela[.][.];
Fim Constroi_Solucao_Inicial;
```
Figura 4.7 – Algoritmo de geração de solução inicial

É importante salientar que em virtude da representação adotada e da forma de construção de uma solução inicial, o requisito que trata do número de vezes que um time pode jogar com cada um dos outros oponentes é sempre verificado, isto é, cada time confronta cada um dos outros apenas uma única vez.

4.5. Função de Avaliação

O problema de programação de jogos não possui uma função de avaliação tão natural quanto às de muitos outros problemas de otimização. As escalas de competições feitas manualmente em geral exprimem o resultado de uma negociação entre os clubes participantes, a entidade organizadora e as redes de televisão, iteradas por várias tentativas de conciliar interesses legítimos, mas muitas vezes mutuamente excludentes. É, portanto, uma tarefa complexa construir uma função de avaliação que exprima de forma fiel à interação entre os inúmeros aspectos a considerar. A solução adotada é inspirada no trabalho de Costa (1995), o qual avalia uma escala *s* através da seguinte função:

$$f(s) = \sum_{i \in C} \alpha_i f_i(s) x_i$$

onde:

- Cada componente $f_i(s)$ desta função computa o grau de violação da *i*-ésima restrição do conjunto *C* ;
- α_i é um peso que representa a importância relativa de se atender a essa restrição e;
- x_i é um valor binário que indica se a *i*-ésima restrição será computada durante a resolução. Ela assumirá valor 1 se a *i*-ésima restrição tiver que ser computada e assumirá valor 0 se a *i*-ésima restrição não tiver que ser computada.

A variável binária x_i foi necessária na implementação da estratégia de solução, onde se realiza a execução do algoritmo *Simulated Annealing* em duas etapas, sendo os objetivos de cada etapa diferentes. Essa estratégia de solução será explicada na seção 4.8.

Os valores dos pesos adotados foram escolhidos de forma a se produzir soluções que fossem comparáveis com aquelas produzidas manualmente.

Sendo assim, a função de avaliação considerada neste trabalho, a qual deve ser minimizada, é baseada em penalidade e é composta por 9 partes distintas, descritas na seção 3.2 e 3.3.

4.6. *Simulated Annealing* Aplicado aos Problemas

Para tratar o problema de programação de jogos propôs-se uma modificação no algoritmo básico de *Simulated Annealing* no que diz respeito à geração de vizinhos. Na versão básica do algoritmo *Simulated Annealing* apresentado na seção 2.4 os vizinhos são obtidos a partir da realização de um determinado tipo de movimento (linha 7 da figura 2.1). Para os problemas abordados foram definidos dois tipos de movimentos: troca de mando de campo e troca de jogos entre rodadas. Baseado nessa estratégia de solução foi feita a seguinte modificação no método: quando o algoritmo vai gerar um vizinho *s'* de uma solução corrente *s*, é escolhido aleatoriamente um tipo de movimento, sendo o vizinho obtido a partir da aplicação deste movimento. Além da modificação na geração de vizinhos, foi acrescentada ao método a característica de reaquecimento. Essa característica realiza o aumento da temperatura do *Simulated Annealing*, voltando a aceitar movimentos de piora e aumentando as chances de se escapar de ótimos locais. O reaquecimento é realizado quando a temperatura corrente T for menor que a temperatura de reaquecimento TR e as inviabilidades essenciais ainda não estiverem todas atendidas. Para essa temperatura de reaquecimento (TR) foi

definido um valor bem próximo da temperatura de congelamento, para que o reaquecimento sempre ocorra quando o algoritmo estiver próximo do fim de sua execução a ainda existirem inviabilidades essenciais não atendidas. A figura 4.8 apresenta o algoritmo após as modificações.

```
Procedimento SA(f(.), N(.), α, SAmax, T0, s)
1. s* ← s;                {Melhor solução obtida até o momento}
2. Iter ← 0;              {Número de iterações na temperatura T}
3. T ← T0;                {Temperatura corrente};
4. enquanto ( T > temperatura de congelamento) faça
5.       enquanto (Iter < SAmax) faça
6.            Iter ← Iter + 1;
7.            Escolha aleatoriamente um dos movimentos definidos;
8.            Gere um vizinho qualquer s` ∈ N(s), a partir do movimento escolhido;
9.            Δ = f(s`) – f(s);
10.           se (Δ < 0 ) então
11.                s ← s`;
12.                se ( f(s`) < f(s*) ) então s* ← s`;
13.           senão
14.                Tome x ∈ [0, 1];
15.                se ( x < e^(-Δ/T)) então s ← s`;
16.           fim-se
17.      fim-enquanto
18.           T ← α * T;
19.           Iter ← 0;
20.           se ( T < TR ) então
21.                se ( inviabilidades essenciais não atendidas ) então
22.                     T ← 0.20 * (0.10 * T0);
23.                     SAmax ← 0.20 * SAmax;
24.                senão T ← TC – 1;  {Congela o sistema para terminar a execução}
25.                fim-se
26.           fim-se
25. fim-enquanto
26. s ← s*;
27. Retorne s;
Fim SA
```

Figura 4.8 – Algoritmo *Simulated Annealing* aplicado aos problemas

4.7. Busca Tabu Aplicado aos Problemas

São propostas três modificações em relação ao algoritmo básico de Busca Tabu (vide seção 2.5) para aplicá-lo aos problemas. Como são utilizados dois tipos diferentes de movimentos, são criadas duas listas tabu. Nessas listas são armazenados os atributos que impedem que um movimento realizado pela troca de mando de campo ou pela troca de jogos entre rodadas resulte no retorno a uma solução já gerada anteriormente. Mais precisamente, na lista L_{campo} armazena-se a dupla <rodada, jogo> e na lista L_{rodada} armazena-se os valores <rodada1, t1_r1, t2_r1, rodada2, t1_r2, t2_r2>. As outras duas modificações dizem respeito à geração de vizinhos. No algoritmo Busca Tabu original a geração de vizinhos se dá através do

cálculo da vizinha completa a partir de uma solução corrente *s*. Dentre todos os vizinhos existentes o vizinho *s'* escolhido é aquele que apresentar o melhor valor da função de avaliação. No algoritmo proposto neste trabalho não é calculada toda a vizinhança existente para uma certa solução *s*. A cada passo da fase de geração de vizinhança é feita uma verificação da solução *s* corrente com o vizinho *s'* calculado. Caso esse vizinho *s'* seja uma solução melhor que a solução corrente *s,* ele é aceito e passa a ser a nova solução corrente e, o cálculo da vizinhança é interrompido. Com esse tipo de implementação, toda vez que durante o cálculo da vizinhança for obtida uma solução que melhore o valor da solução corrente, o algoritmo não analisará a vizinhança completa, economizando tempo de processamento. Outra modificação feita também visa diminuir o número de vizinhos analisados, pois durante a fase de geração de vizinhança, um novo vizinho *s'* é calculado sempre que o anterior não for de melhora. Com isso tem-se, no pior caso, o cálculo da vizinhança completa, ou seja, não existir nenhum vizinho que melhore a solução corrente. Como este não é um fato difícil de acontecer, no algoritmo proposto a vizinhança calculada será apenas uma parte (por exemplo, 30% ou 50%) da vizinhança total. Teremos então calculado uma vizinhança parcial.

Essa variação do algoritmo de Busca Tabu é conhecida como algoritmo de Busca Tabu com Primeiro Vizinho de Melhora ou Um Percentual da Vizinhança Analisada. A figura 4.9 ilustra o algoritmo proposto.

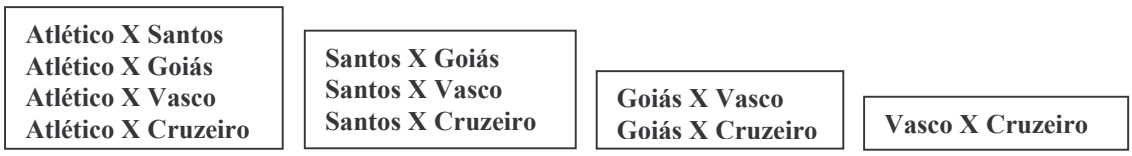
```
Procedimento BT(f(.), N(.), |V|, |T|, BTmax, s)
1.  s* ← s;                    {Melhor solução obtida até o momento}
2.  Iter ← 0;                  {Contador do número de iterações}
3.  MelhorIter ← 0;            {Iteração mais recente que forneceu s*}
4.  L^k ← ∅;                   {Listas Tabu: L^1: L_mando; L^2: L_rodada}
5.  enquanto ( Iter – MelhorIter < BTmax ) faça
6.      Iter ← iter + 1;
7.      Seja s` ← s ⊕ m o primeiro elemento que melhorou s ou o melhor elemento
            de V ⊂ (N^1(s) U N^2(s)) tal que o movimento m não seja tabu (m ∉ L^k);
8.      L^k ← L^k – {movimento mais antigo} + {movimento que gerou s`};
9.      s ← s`;
10.     se ( f(s) < f(s*) ) então
11.         s* ← s;
12.         MelhorIter ← Iter;
13.     fim-se
14.  fim-enquanto
15.  s ← s*;
16.  Retorne s;
Fim BT
```

Figura 4.9 – Algoritmo Busca Tabu aplicado aos problemas

4.8. Funcionamento Básico do Algoritmo

Passo 1 – *Geração da Lista de Confrontos*

Nesse passo é gerada uma lista contendo todos os confrontos possíveis no campeonato, ou seja, todos os times são combinados dois a dois sem gerar repetição. Exemplo: para uma competição envolvendo Atlético, Santos, Goiás, Vasco e Cruzeiro, tem-se os seguintes confrontos:



Uma vez gerados todos os confrontos possíveis (sem repetição) tem-se que resolver o problema do mando de campo. Para resolver esse problema adotou-se o uso de um **sinal**. Como se optou por trabalhar com índices (números inteiros) para representar os times, o uso do sinal no índice do primeiro time indica o mando de campo. Veja no exemplo abaixo:

\+ Atlético X Vasco ➔ Mando de campo do Atlético
– Atlético X Vasco ➔ Mando de campo do Vasco

Passo 2 – *Geração de Solução Inicial*

Conforme descrito na seção 4.4, o algoritmo de geração de solução inicial depende da lista de confrontos. Uma vez que essa lista já tenha sido gerada o algoritmo a percorre retirando cada um dos jogos e preenchendo a matriz que representa a tabela de solução.

Passo 3 – *Execução do Simulated Annealing*

O terceiro passo foi dividido em duas etapas. Na **primeira etapa** utiliza-se o algoritmo *Simulated Annealing* em sua forma adaptada para problema. Nessa etapa tem-se como objetivo a resolução dos seguintes requisitos:

a) Um time não pode jogar mais de uma vez na mesma rodada;
b) Nas duas primeiras rodadas que cada time participar, um jogo deve ser realizado em sua sede e outro fora de sua sede. Por exemplo, se na primeira rodada o confronto de um time for dentro de casa, então na segunda rodada o confronto deste time deve ser fora de casa;
c) As duas últimas rodadas que cada time participar devem ter a mesma configuração de seus dois primeiros confrontos. Exemplificando, se os dois primeiros confrontos de um time foram jogar fora de casa e depois em casa, respectivamente, então os dois últimos confrontos desse time devem ser, respectivamente, jogar fora de casa e depois em casa;
d) Não pode haver jogos entre clubes do mesmo estado na última rodada.

Essa estratégia foi adotada devido ao seguinte fato: os requisitos listados anteriormente estão diretamente ligados, ou seja, o atendimento de um depende diretamente do outro. Portanto, para primeira etapa executa-se o algoritmo *Simulated Annealing* adaptado, ou seja, com os dois movimentos definidos (troca de jogos entre rodadas e troca de mando de

campo) e com reaquecimento, com o objetivo de gerar o escalonamento, sem se preocupar com as definições de mando de campo. Essa etapa só termina quando os requisitos listados acima forem atendidos ou quando o tempo de execução máximo, estabelecido pelo usuário, terminar.

Na **segunda etapa** utiliza-se o algoritmo *Simulated Annealing* em sua forma básica, realizando apenas o movimento de troca de mando de campo para atender os seguintes requisitos:

a) O número de jogos realizados fora de casa deve ser igual ao número de jogos em casa. Quando o número de jogos for ímpar, o time com melhor classificação no ranking da CBF, joga uma partida a mais em casa;
b) Evitar que um time jogue mais de duas vezes consecutivas em casa;
c) Evitar que um time jogue mais de duas vezes consecutivas fora de casa;
d) Minimizar a distância total percorrida pelos times;
e) Minimizar a diferença entre a distância percorrida pelo time que se deslocou mais e a distância percorrida pelo time que se deslocou menos.

Todos os requisitos tratados na primeira etapa de execução do *Simulated Annealing* são essenciais, portanto devem ser atendidos. Entre os requisitos listados para a execução da segunda etapa apenas o requisito (a) é essencial, os demais são não essenciais e devem ser atendidos sempre que for possível.

Passo 4 – *Refinamento da Solução com o Algoritmo Busca Tabu*

Terminada a segunda etapa de execução do *Simulated Annealing* inicia-se a execução do algoritmo Busca Tabu, tendo como solução inicial a melhor solução encontrada até o momento. Com isso busca-se o refinamento das soluções geradas pelo *Simulated Annealing*.

Essa ordem de execução foi adotada pelo fato do algoritmo Busca Tabu ser altamente dependente de uma boa solução inicial, o que não acontece com o algoritmo *Simulated Annealing*, o qual, mesmo partindo de uma solução inicial aleatória, consegue gerar boas soluções em espaços de tempo curtos, quando comparado a outras metaheurísticas.

5. Sistema Desenvolvido

5.1. Descrição do Sistema

O sistema desenvolvido, chamado "PJCB Solver" tem o intuito de tentar solucionar o Problema de Programação de Jogos do Campeonato Brasileiro de Futebol, e ao mesmo tempo ser utilizado por qualquer pessoa que trabalha ou se interessa por esse problema. Ele foi desenvolvido na linguagem C++ com o compilador C++ Builder 5.0, devido ao poder e desempenho da linguagem e a quantidade de recursos da ferramenta.

O sistema tenta oferecer um ambiente de uso agradável, flexível e de fácil utilização. 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.

O funcionamento básico do sistema pode ser descrito em 4 etapas:

Primeira Etapa ➔ *Dados de entrada do problema a ser tratado*: Nessa etapa o usuário deverá criar os arquivos de entrada de dados para o programa. A criação desses arquivos deve ser realizada fora do ambiente do programa. A seção 5.2 descreve cada um dos arquivos de entrada.

Segunda Etapa ➔ *Abertura do arquivo com os dados da programação de jogos*: Nessa etapa o usuário, já no ambiente do "PJCB Solver", seleciona um arquivo (criado previamente) que contém os dados do Problema de Programação de Jogos a ser resolvido.

Terceira Etapa ➔*Escolha dos requisitos a serem tratados e dos algoritmos a serem executados pelo programa*: Nessa etapa o usuário escolhe quais requisitos deverão ser atendidos e quais algoritmos serão executados.

Quarta Etapa ➔ *Execução do Simulated Annealing e/ou Busca Tabu*: Nessa etapa o usuário executa o programa. Inicia-se então a execução dos métodos selecionados. No caso de escolha dos dois métodos a seqüência de execução dos algoritmos é a seguinte: *Simulated Annealing,* seguida pela execução do algoritmo Busca Tabu.

Obs: Em qualquer momento antes da quarta etapa, o usuário ainda pode alterar os valores dos parâmetros do *Simulated Annealing,* do Busca Tabu e os valores dos pesos utilizados para penalizar a ocorrência de uma inviabilidade.

Terminada a quarta etapa, será exibido o resultado encontrado. O usuário poderá ver a tabela de jogos gerada, bem como os valores de cada requisito tratado pelo programa.

5.2. Arquivos de Entrada do Sistema

Foram definidas 3 estruturas básicas de arquivo contendo os dados de entrada para o programa. São eles: arquivos de dados (*.dat), arquivo de distâncias (*.dist) e arquivo de times (*.tim). A seguir será descrita cada uma dessas estruturas de arquivo.

5.2.1. Arquivo de Dados (Extensão *.dat)

Os arquivos de dados *.dat são os arquivos de entrada principais do programa. São neles que estarão todos os dados relativos à montagem da tabela de jogos, bem como os caminhos para os arquivos de distâncias e de times. Sua estrutura segue o seguinte modelo:

```
Nº de turnos, Nº de times, Nº de rodadas
Caminho para o arquivo de distâncias *.dist
Caminho para o arquivo de times *.tim
```

Figura 5.1 – Modelo do arquivo de dados *.dat

Exemplo: Para o campeonato de 2002 temos a seguinte estrutura de arquivo:

```
1, 26, 29
C:\PJCB\campeonato2002.dist
C:\PJCB\campeonato2002.tim
```

Figura 5.2 – Exemplo de um arquivo de dados

5.2.2. Arquivo de Distâncias (Extensão *.dist)

Os arquivos de distâncias *.dist são os arquivos que guardam os dados referentes às distâncias entre os times participantes do campeonato, bem como as informações sobre os jogos entre times do mesmo estado. Sua estrutura segue o seguinte modelo:

```
Índice time1, Índice time2, distância, jogo estadual
```

Figura 5.3 – Modelo do arquivo de distâncias *.dist

O campo referente aos jogos estaduais deve assumir valor 1 para jogos entre clubes do mesmo estado e 0 caso contrário.

Exemplo: Para o campeonato de 2002 temos o seguinte fragmento do arquivo de distâncias:

```
1, 1, 0, 1
1, 2, 0, 1
1, 3, 586, 0
1, 4, 586, 0
```

Figura 5.4 – Exemplo de um arquivo de distâncias

onde os índices 1 = Atlético-MG, 2 = Cruzeiro, 3 = São Paulo, 4 = São Caetano.

O arquivo de distâncias deve conter a distância de cada time para todos os outros, para que possa ser montada uma matriz de distância entre todos os clubes.

5.2.3. Arquivo de Times (Extensão *.tim)

Os arquivos de times *.tim são usados para carregar o nome dos clubes participantes do campeonato. Para que a impressão da tabela seja realizada de forma correta os nomes dos times devem ser colocados um em cada linha e seguindo a ordem dos índices que representarão esses times. Por exemplo: suponha que seja definida uma estrutura de índices onde 1 = Atlético-MG, 2 = Cruzeiro, 3 = São Paulo, 4 = São Caetano. A ordem de preenchimento dos nomes dos clubes no arquivo deve ser:

```
Atlético_MG
Cruzeiro
São_Paulo
São_Caetano
```

Figura 5.5 – Exemplo de um arquivo de times

5.3. Recursos Oferecidos Pelo Sistema

5.3.1. Alteração de Parâmetros do *Simulated Annealing* e Busca Tabu

O sistema permite ao usuário alterar os parâmetros do algoritmo *Simulated Annealing* e do algoritmo Busca Tabu. Isso pode ser feito através da tela representada pela figura 5.6.

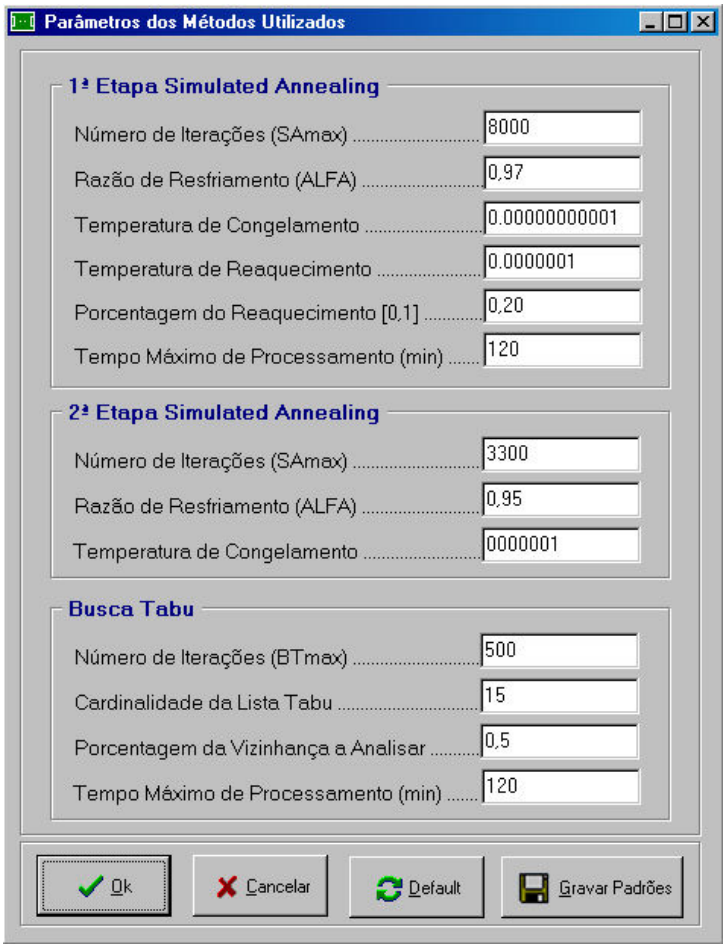


Figura 5.6 – Tela de alteração de parâmetros do SA e do BT

1ª Etapa do *Simulated Annealing*:

a) *Número de Iterações (SAmax)*: Número máximo de iterações realizadas a cada temperatura da primeira etapa do *Simulated Annealing*.
b) *Razão de Resfriamento (ALFA)*: parâmetro de resfriamento da temperatura na primeira etapa do *Simulated Annealing*.
c) *Temperatura de Congelamento*: Valor da temperatura de congelamento da primeira etapa, ou seja, valor de temperatura no qual o *Simulated Annealing* termina sua execução.
d) *Temperatura de Reaquecimento*: Valor da temperatura onde o reaquecimento é realizado, ou seja, quando a temperatura corrente for menor que a temperatura de reaquecimento ela é reaquecida.
e) *Porcentagem do Reaquecimento [0,1]*: Valor do reaquecimento dado em porcentagem, onde 0 representa 0% e 1 representa 100%.
f) *Tempo Máximo de Processamento (min)*: valor, em minutos, do tempo máximo de execução da primeira etapa do *Simulated Annealing*.

2ª Etapa do *Simulated Annealing*:

a) *Número de Iterações (SAmax)*: Número máximo de iterações realizadas a cada temperatura da segunda etapa do *Simulated Annealing*.
b) *Razão de Resfriamento (ALFA)*: parâmetro de resfriamento da temperatura na segunda etapa do *Simulated Annealing*.
c) *Temperatura de Congelamento*: Valor da temperatura de congelamento da segunda etapa, ou seja, valor de temperatura no qual o *Simulated Annealing* termina sua execução.

Busca Tabu:

a) *Número de Iterações (BTmax)*: Número máximo de iterações sem melhora executadas pelo Busca Tabu.
b) *Cardinalidade da Lista Tabu*: Tamanho máximo da Lista Tabu.
c) *Porcentagem da Vizinhança a Analisar*: Porcentagem da vizinhança a ser analisada. Esse valor está definido no intervalo [0,1], onde 0 representa 0% e 1 representa 100%.
d) *Tempo Máximo de Processamento (min)*: valor, em minutos, do tempo máximo de execução do Busca Tabu.

Ao se clicar no botão "Ok", da tela representada pela figura 5.6, os valores digitados pelo usuário passam a ser utilizados pelo algoritmo durante a execução do programa.

Ao se clicar no botão "Cancelar", da tela representada pela figura 5.6, os valores dos parâmetros não sofrem alteração.

Ao se clicar no botão "Default", da tela representada pela figura 5.6, os valores padrões dos parâmetros são restaurados a partir do arquivo "parametros.ini".

Ao se clicar no botão "Gravar Padrões", da tela representada pela figura 5.6, os valores padrões dos parâmetros serão alterados diretamente no arquivo "parametros.ini", ou seja, são definidos novos valores padrões.

5.3.2. Alteração dos Pesos dos Requisitos

Na tela apresentada na figura 5.7 os usuários poderão alterar os valores dos pesos definidos para as inviabilidades.

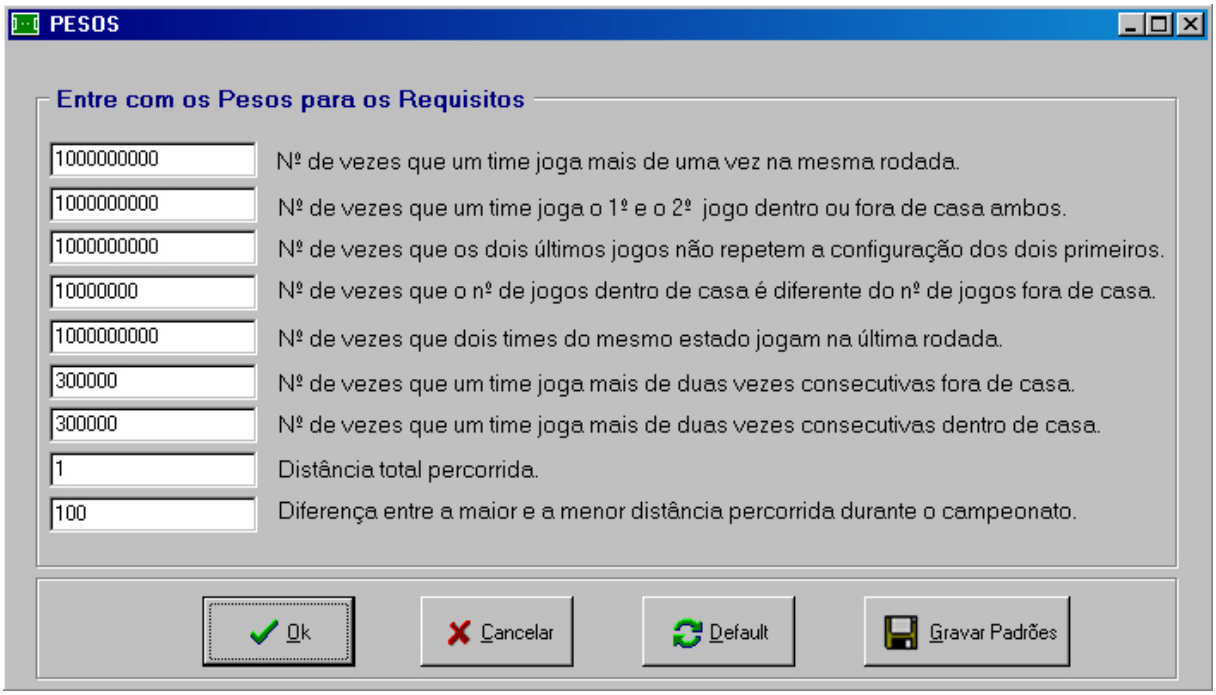


Figura 5.7 – Tela de alteração dos pesos das inviabilidades

a) *Nº de vezes que um time joga mais de uma vez na mesma rodada*: valor utilizado para penalizar cada jogo excedente, do time, na mesma rodada. Por exemplo, se um time joga três vezes na mesma rodada tem-se duas penalizações, uma para cada jogo excedente.
b) *Nº de vezes que um time joga o 1º e o 2º jogo dentro ou fora de casa ambos*: valor utilizado para penalizar cada vez que os dois primeiros jogos não possuem o mando de campo invertido.
c) *Nº de vezes que os dois últimos jogos não repetem a configuração dos dois primeiros*: valor utilizado para penalizar cada vez em que os dois últimos jogos de cada time não repetem o mando dos dois primeiros jogos.
d) *Nº de vezes que o nº de jogos dentro de casa é diferente do nº de jogos fora de casa*: valor utilizado para penalizar cada vez que o número de jogos dentro de casa for diferente do número de jogos fora de casa, para cada time.
e) *Nº de vezes que dois times do mesmo estado jogam na última rodada*: valor utilizado para penalizar o número de vezes que dois times do mesmo estado se confrontam na última rodada.
f) *Nº de vezes que um time joga mais de duas vezes consecutivas fora de casa*: valor utilizado para penalizar o número de jogos além de dois consecutivos fora de casa, para cada time.
g) *Nº de vezes que um time joga mais de duas vezes consecutivas dentro de casa*: valor utilizado para penalizar o número de jogos além de dois consecutivos dentro de casa, para cada time.
h) *Distância total percorrida*: valor utilizado para penalizar a distância total percorrida com o intuito de minimizá-la.

i) *Diferença entre a maior e a menor distância percorrida durante o campeonato*: valor utilizado para penalizar a diferença entre as distâncias percorridas pelo time que mais se deslocou e o time que menos se deslocou com intuito de minimizá-la.

Ao se clicar no botão "Ok", da tela representada pela figura 5.7, os valores digitados pelo usuário passam a ser utilizados pelo algoritmo durante a execução do programa.

Ao se clicar no botão "Cancelar", da tela representada pela figura 5.7, os valores dos pesos não sofrem alteração.

Ao se clicar no botão "Default", da tela representada pela figura 5.7, os valores padrões dos pesos são restaurados a partir do arquivo "pesos.ini".

Ao se clicar no botão "Gravar Padrões" da tela representada pela figura 5.7 os valores padrões dos pesos serão alterados diretamente no arquivo "pesos.ini".

5.3.3. Escolha dos Requisitos e Algoritmos a serem Executados

Na tela apresentada na figura 5.8 os usuários poderão escolher quais requisitos serão considerados na execução dos métodos, sendo estes escolhidos nessa mesma tela.

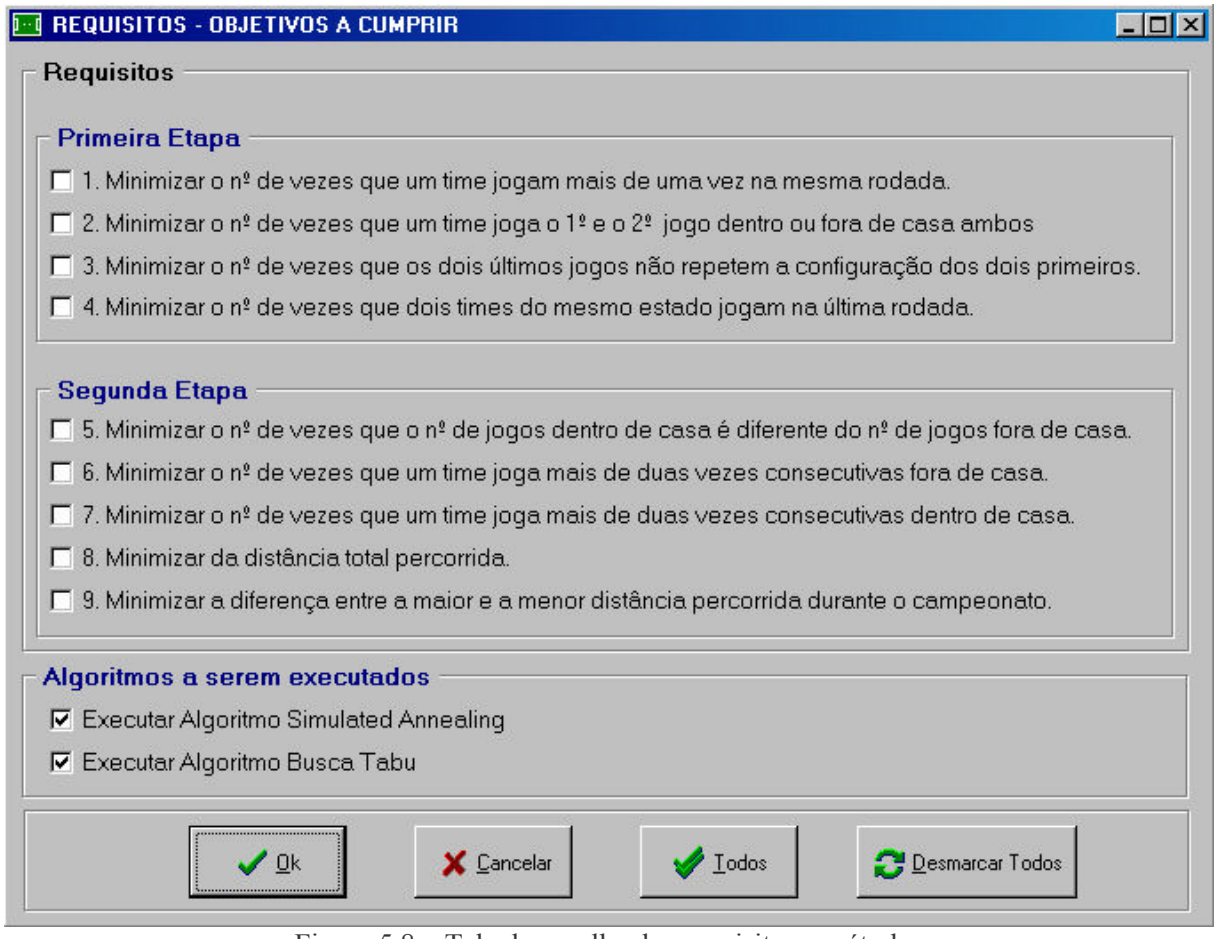


Figura 5.8 – Tela de escolha dos requisitos e métodos

*"**Primeira Etapa**"*: Requisitos a serem considerados na execução da primeira etapa do *Simulated Annealing*.

"Segunda Etapa": Requisitos a serem considerados na execução da segunda etapa do *Simulated Annealing*.

"Algoritmos a serem executados": Escolha dos algoritmos que serão executados pelo programa.

Obs: No algoritmo *Simulated Annealing* a execução ocorre em duas etapas, sendo os requisitos acima considerados em suas respectivas etapas. No algoritmo Busca Tabu a execução ocorre em uma única etapa, portanto todos os requisitos marcados (primeira e segunda etapas) são considerados durante a sua execução.

Ao se clicar no botão "Ok", da tela representada pela figura 5.8, os requisitos e algoritmos escolhidos pelo usuário passam a ser utilizados na execução do programa.

Ao se clicar no botão "Cancelar", da tela representada pela figura 5.8, nenhum requisito será considerado.

Ao se clicar no botão "Todos", da tela representada pela figura 5.8, todos os requisitos são marcados para serem considerados pelos algoritmos.

Ao se clicar no botão "Desmarcar Todos", da tela representada pela figura 5.8, todos os requisitos são desmarcados.

5.3.4. Acompanhamento da Execução do Algoritmo

Na tela representada pela figura 5.9 o usuário pode acompanhar todos os detalhes da execução do programa.

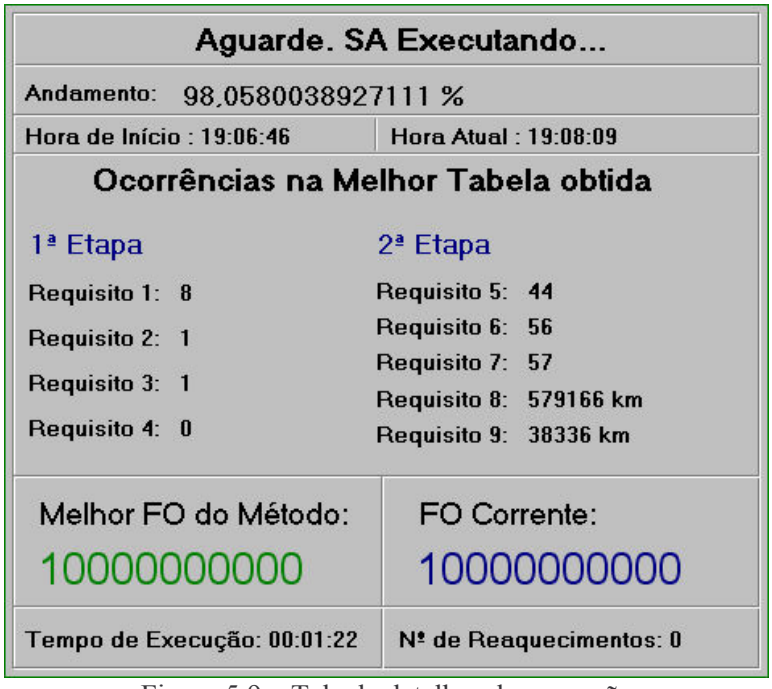


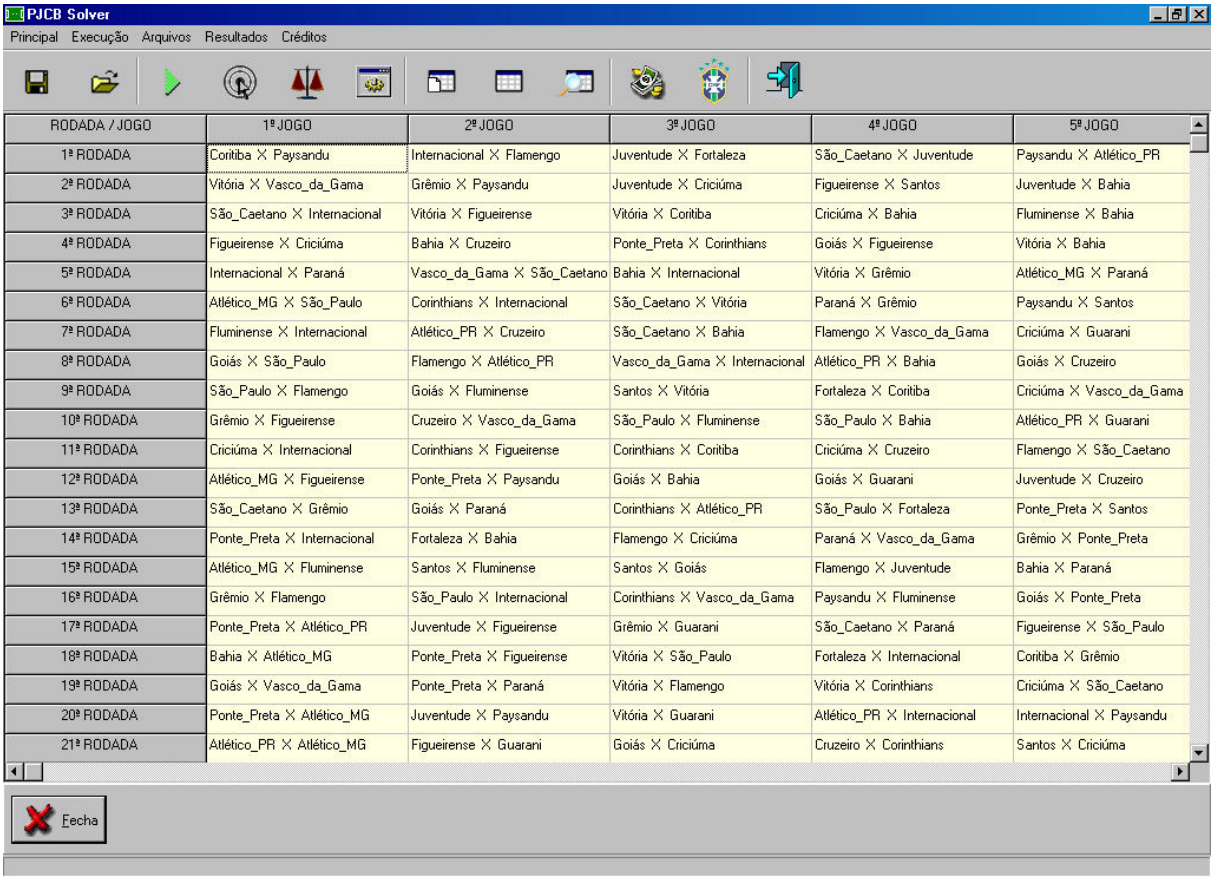
Figura 5.9 – Tela de detalhes da execução

a) *Andamento*: andamento da execução.
b) *Hora de Início*: horário que o algoritmo começou sua execução.
c) *Hora Atual*: exibe a hora atual.
d) *Requisito 1*: Nº de vezes que um time joga mais de uma vez na mesma rodada.
e) *Requisito 2*: Nº de vezes que um time joga os dois primeiros jogos ou dentro de casa ou fora de casa.

f) *Requisito 3*: Nº de vezes que os dois últimos jogos não repetem a configuração dos dois primeiros.
g) *Requisito 4*: Nº de vezes que dois times do mesmo Estado jogam na última rodada
h) *Requisito 5*: Nº de vezes que o nº de jogos dentro de casa é diferente do nº de jogos fora de casa.
i) *Requisito 6*: Nº de vezes que um time joga mais de duas vezes consecutivas fora de casa.
j) *Requisito 7*: Nº de vezes que um time joga mais de duas vezes consecutivas dentro de casa.
k) *Requisito 8*: Distância total percorrida.
l) *Requisito 9*: Diferença entre a maior e a menor distância percorrida durante o campeonato.
m) *Melhor FO do método*: melhor valor da função de avaliação encontrada pelo método que está executando (*SA* ou BT).
n) *FO Corrente*: valor da função de avaliação corrente.
o) *Tempo de execução*: tempo que o algoritmo está executando.
p) *Nº de Reaquecimentos*: número de reaquecimentos na primeira etapa do *SA*.

5.3.5. Visualização da Tabela de Jogos Inicial e Obtida

Na tela representada pela figura 5.10 o usuário pode visualizar a tabela de jogos inicial e a tabela de jogos obtida pelo sistema.



RODADA / JOGO	1º JOGO	2º JOGO	3º JOGO	4º JOGO	5º JOGO
1ª RODADA	Coritiba X Paysandu	Internacional X Flamengo	Juventude X Fortaleza	São_Caetano X Juventude	Paysandu X Atlético_PR
2ª RODADA	Vitória X Vasco_da_Gama	Grêmio X Paysandu	Juventude X Criciúma	Figueirense X Santos	Juventude X Bahia
3ª RODADA	São_Caetano X Internacional	Vitória X Figueirense	Vitória X Coritiba	Criciúma X Bahia	Fluminense X Bahia
4ª RODADA	Figueirense X Criciúma	Bahia X Cruzeiro	Ponte_Preta X Corinthians	Goiás X Figueirense	Vitória X Bahia
5ª RODADA	Internacional X Paraná	Vasco_da_Gama X São_Caetano	Bahia X Internacional	Vitória X Grêmio	Atlético_MG X Paraná
6ª RODADA	Atlético_MG X São_Paulo	Corinthians X Internacional	São_Caetano X Vitória	Paraná X Grêmio	Paysandu X Santos
7ª RODADA	Fluminense X Internacional	Atlético_PR X Cruzeiro	São_Caetano X Bahia	Flamengo X Vasco_da_Gama	Criciúma X Guarani
8ª RODADA	Goiás X São_Paulo	Flamengo X Atlético_PR	Vasco_da_Gama X Internacional	Atlético_PR X Bahia	Goiás X Cruzeiro
9ª RODADA	São_Paulo X Flamengo	Goiás X Fluminense	Santos X Vitória	Fortaleza X Coritiba	Criciúma X Vasco_da_Gama
10ª RODADA	Grêmio X Figueirense	Cruzeiro X Vasco_da_Gama	São_Paulo X Fluminense	São_Paulo X Bahia	Atlético_PR X Guarani
11ª RODADA	Criciúma X Internacional	Corinthians X Figueirense	Corinthians X Coritiba	Criciúma X Cruzeiro	Flamengo X São_Caetano
12ª RODADA	Atlético_MG X Figueirense	Ponte_Preta X Paysandu	Goiás X Bahia	Goiás X Guarani	Juventude X Cruzeiro
13ª RODADA	São_Caetano X Grêmio	Goiás X Paraná	Corinthians X Atlético_PR	São_Paulo X Fortaleza	Ponte_Preta X Santos
14ª RODADA	Ponte_Preta X Internacional	Fortaleza X Bahia	Flamengo X Criciúma	Paraná X Vasco_da_Gama	Grêmio X Ponte_Preta
15ª RODADA	Atlético_MG X Fluminense	Santos X Fluminense	Santos X Goiás	Flamengo X Juventude	Bahia X Paraná
16ª RODADA	Grêmio X Flamengo	São_Paulo X Internacional	Corinthians X Vasco_da_Gama	Paysandu X Fluminense	Goiás X Ponte_Preta
17ª RODADA	Ponte_Preta X Atlético_PR	Juventude X Figueirense	Grêmio X Guarani	São_Caetano X Paraná	Figueirense X São_Paulo
18ª RODADA	Bahia X Atlético_MG	Ponte_Preta X Figueirense	Vitória X São_Paulo	Fortaleza X Internacional	Coritiba X Grêmio
19ª RODADA	Goiás X Vasco_da_Gama	Ponte_Preta X Paraná	Vitória X Flamengo	Vitória X Corinthians	Criciúma X São_Caetano
20ª RODADA	Ponte_Preta X Atlético_MG	Juventude X Paysandu	Vitória X Guarani	Atlético_PR X Internacional	Internacional X Paysandu
21ª RODADA	Atlético_PR X Atlético_MG	Figueirense X Guarani	Goiás X Criciúma	Cruzeiro X Corinthians	Santos X Criciúma

Figura 5.10 – Tela de visualização da tabela de jogos inicial e obtida pelo sistema

5.3.6. Visualização dos Resultados das Tabelas Oficiais de 2002 e 2003

O usuário pode carregar os dados das tabelas oficiais de 2002 e 2003. Na tela representada pela figura 5.11 o usuário deve escolher o campeonato. Na tela representada pela figura 5.12 os dados são mostrados.

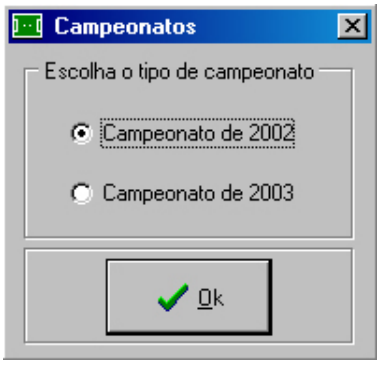


Figura 5.11 – Tela de escolha do campeonato

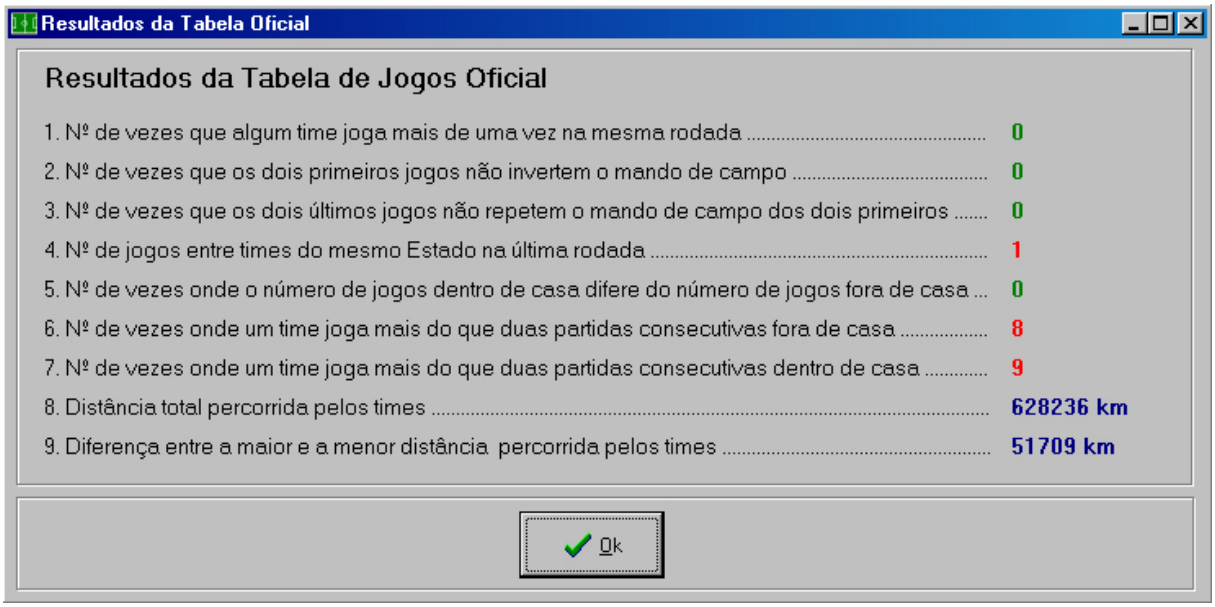


Figura 5.12 – Tela visualização dos resultados da tabela oficial de 2002 e 2003

5.3.7. Gerar Nova Solução Inicial

O sistema permite que o usuário gere novas soluções iniciais, desde que os arquivos de dados tenham sido carregados. Para gerar uma nova solução inicial basta clicar no botão representado pela figura 5.13.

Figura 5.13 – Botão de geração de solução inicial

5.3.8. Salvar uma Tabela Obtida

Após a execução do programa e obtenção de uma tabela de jogos o usuário poderá salvar esta tabela em formato de texto (*.txt) ou em formato de planilha do Microsoft Excel (*.xls). Para salvar a tabela basta clicar no botão representado pela figura 5.14.

Figura 5.14 – Botão para salvar a tabela de jogos obtida

5.4. Outras Telas do Sistema

1) Tela de entrada de arquivo com os dados da programação de jogos

Na tela representada pela figura 5.15 o usuário seleciona o arquivo contendo os dados da programação de jogos.

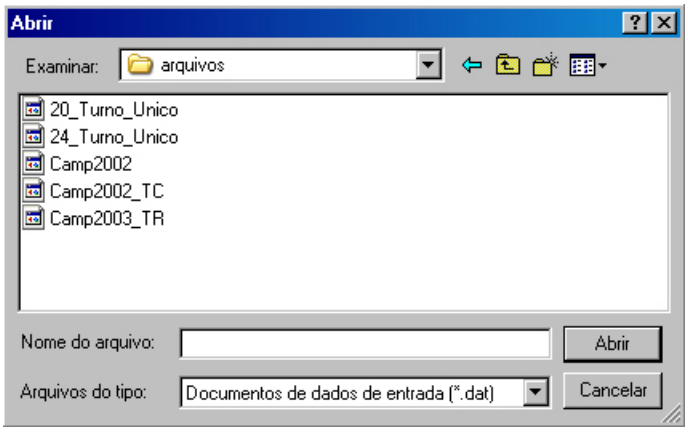


Figura 5.15 – Tela de abertura de arquivo de dados

2) Tela de visualização dos resultados obtidos

Na tela abaixo (figura 5.16) o usuário pode visualizar as resultados obtidos na última execução do algoritmo. Nela são mostrados todos os valores dos requisitos tratados, sendo que estes assumem cor verde quando o requisito não apresenta inviabilidade e cor vermelha quando o requisito possui inviabilidade. Dados sobre o campeonato também são apresentados, tais como: número de times participantes, número de rodadas, número de turnos e número de jogos disputados. São apresentados também o tempo de execução e o status da tabela obtida, podendo este último assumir os seguintes valores:

- "Tabela Viável" ➔ Cor verde, indicando solução viável.
- "Tabela Inviável" ➔ Cor vermelha indicando solução inviável.

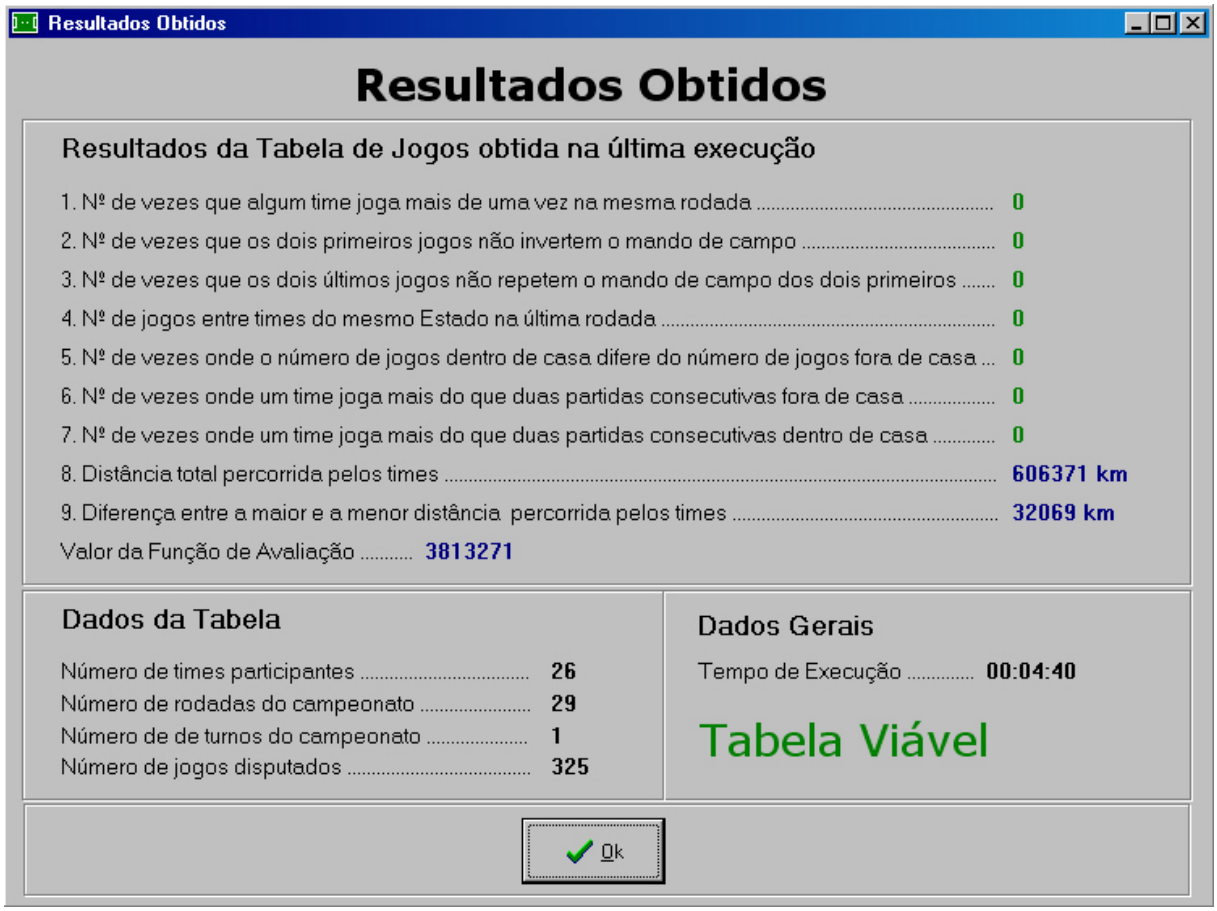


Figura 5.16 – Visualização dos resultados obtidos

3) Tela para salvar as tabelas geradas

Na tela representada pela figura 5.17 o usuário poderá escolher um nome para o arquivo que conterá os dados da tabela de jogos gerada após a execução do sistema.

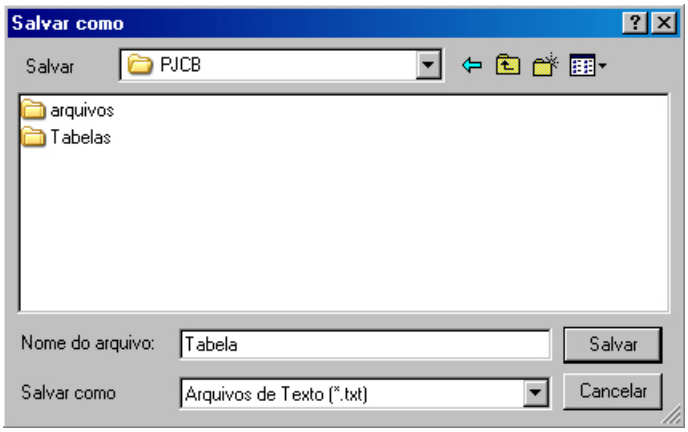


Figura 5.17 – Tela para salvar as tabelas geradas

4) Tela principal do sistema

A partir da tela abaixo (figura 5.18) o usuário executa todas as demais funções do programa, sendo que essas funções podem ser acessadas através do menu principal ou através dos botões de atalho.

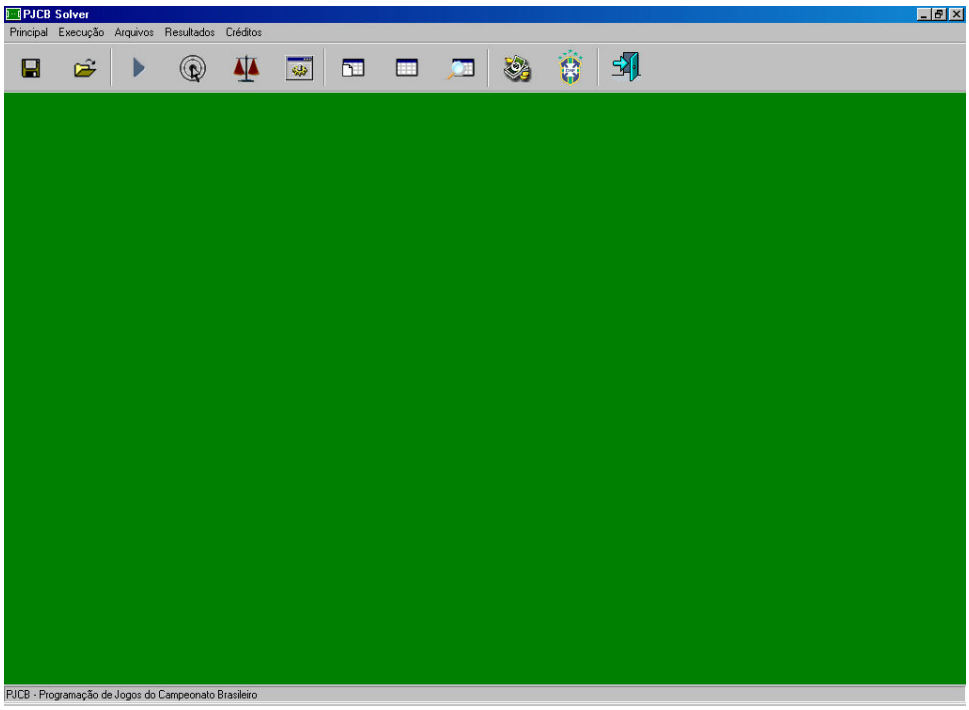


Figura 5.18 – Tela principal do PJCB Solver

6. Resultados Obtidos

O algoritmo desenvolvido foi implementado na linguagem C++ usando o compilador Borland C++ Builder 5.0 e testado em um microcomputador PC AMD Atlhon, 1,53 GHz, com 256 MB de RAM sob sistema operacional *Windows XP*.

Inicialmente, a metaheurística *Simulated Annealing* e Busca Tabu passaram por uma bateria preliminar de testes visando à calibragem de seus parâmetros. Os valores dos parâmetros utilizados, conforme notação adotada na seção 2.4 podem ser visualizados na Tabela 6.1 (primeira etapa do *Simulated Annealing*), Tabela 6.2 (segunda etapa do *Simulated Annealing*) e Tabela 6.3 (algoritmo Busca Tabu).

Parâmetros da 1ª Etapa do *SA*	
Nº de Iterações (**SAmax**)	9000
Temperatura Inicial ($\mathbf{T_0}$)	*
Razão de Resfriamento ($\boldsymbol{\alpha}$)	0.97
Tempo Máximo de Processamento	120 minutos

Tabela 6.1 – Parâmetros da primeira etapa do algoritmo *Simulated Annealing*

Parâmetros da 2ª Etapa do *SA*	
Nº de Iterações (**SAmax**)	3300
Temperatura Inicial ($\mathbf{T_0}$)	*
Razão de Resfriamento ($\boldsymbol{\alpha}$)	0.97

Tabela 6.2 – Parâmetros da segunda etapa do algoritmo *Simulated Annealing*

Parâmetros do Busca Tabu	
Nº de Iterações (**BTmax**)	500
Cardinalidade da Lista Tabu (|**T**|)	15
Vizinhança Analisada (**V**)	0.5 (50%)
Tempo Máximo de Processamento	120 minutos

Tabela 6.3 – Parâmetros do algoritmo Busca Tabu

Foram realizados 10 testes para cada tipo de campeonato considerado, cada qual partindo de uma semente diferente de números aleatórios. Os tipos de campeonato considerados diferenciam-se pelo número de turnos (turno único ou turno e returno) e número de times (20, 24 ou 26 times). Foram realizadas as seguintes combinações:

- Campeonato com 26 times e turno único disputado em 29 rodadas (Campeonato Brasileiro de Futebol de 2002);
- Campeonato com 26 times e turno único disputado em 25 rodadas (rodadas completas ou cheias);
- Campeonato com 24 times e dois turnos (turno e returno) disputados em 46 rodadas (Campeonato Brasileiro de Futebol de 2003);
- Campeonato com 20 times e dois turnos (turno e returno) disputados em 38 rodadas (previsão para os Campeonatos Brasileiros a partir de 2006).

* T_0 gerada por simulação, sendo a temperatura na qual 95% dos movimentos são aceitos.

Como base de comparação foram utilizados os valores obtidos com as tabelas oficiais do Campeonato Brasileiro de Futebol de 2002 para o campeonato de turno único e do Campeonato Brasileiro de Futebol de 2003, ambos da primeira divisão. Os resultados para o campeonato de 2002 podem ser visualizados na Tabela 6.4 e os resultados para o campeonato de 2003 na Tabela 6.5.

Legenda para as Tabelas 6.4, 6.5, 6.6, 6.7, 6.8, 6.9 e 6.10:

- **Requisito 1**: número de vezes que um time jogou mais de uma vez com algum outro time;
- **Requisito 2**: número de vezes que um time jogou mais de uma vez na mesma rodada;
- **Requisito 3**: número de vezes que as duas primeiras rodadas de cada time não possuem o mando de campo invertido;
- **Requisito 4**: número de vezes que as duas últimas rodadas de cada time não repetem o mando de campo das duas primeiras rodadas;
- **Requisito 5**: número de jogos entre times do mesmo estado na última rodada;
- **Requisito 6**: número de vezes que o número de jogos dentro de casa difere do número de jogos fora de casa em mais de uma unidade (devido ao número ímpar de jogos), para cada clube.
- **Requisito 7**: número de vezes que um time joga mais de duas vezes consecutivas fora de casa;
- **Requisito 8**: número de vezes que um time joga mais de duas vezes consecutivas dentro de casa;
- **Requisito 9**: distância total percorrida pelos times;
- **Requisito 10**: diferença entre a distância percorrida pelo time que se deslocou mais para o time que se deslocou menos e;
- **Função Objetivo**: valor da função de avaliação adotada.

Tabela Oficial de 2002	
Requisito	**Valor**
Requisito 1	**0**
Requisito 2	**0**
Requisito 3	**0**
Requisito 4	**0**
Requisito 5	**1**
Requisito 6	**0**
Requisito 7	**8**
Requisito 8	**9**
Requisito 9	**628.236 km**
Requisito 10	**51.709 km**
Função Objetivo	**1.022.799.136**

Tabela 6.4 – Resultados oficiais da tabela do campeonato de 2002

Nota-se que, no campeonato disputado no ano de 2002, a tabela oficial de jogos apresentou alguns problemas em relação aos requisitos 5, 7 e 8. O requisito 5 (nenhum jogo entre times do mesmo Estado na última rodada) é um requisito essencial, portanto deveria ter sido atendido, segundo as convenções da CBF. Os requisitos 7 e 8 são requisitos não essenciais, portanto não tinham obrigação de atendimento. Porém eles controlam o número de jogos consecutivos dentro e fora da sede de cada time, interferindo diretamente no desempenho daqueles afetados por essas ocorrências.

Com a tabela oficial do campeonato disputado no ano de 2003 não ocorreu problema com inviabilidades em nenhum dos requisitos listados. Os resultados podem ser visualizados na tabela abaixo.

Tabela Oficial de 2003	
Requisito	**Valor**
Requisito 1	**0**
Requisito 2	**0**
Requisito 3	**0**
Requisito 4	**0**
Requisito 5	**0**
Requisito 6	**0**
Requisito 7	**0**
Requisito 8	**0**
Requisito 9	**1.379.645 km**
Requisito 10	**97.712 km**
Função Objetivo	**11.150.845**

Tabela 6.5 – Resultados oficiais da tabela do campeonato de 2003

Os pesos utilizados para penalizar a ocorrência de inviabilidades nos requisitos são mostrados na Tabela 6.6.

Pesos Utilizados pelo Algoritmo	
Requisito	**Valor**
Requisito 1	**1000000000**
Requisito 2	**1000000000**
Requisito 3	**1000000000**
Requisito 4	**10000000**
Requisito 5	**1000000000**
Requisito 6	**2000000000**
Requisito 7	**1000000**
Requisito 8	**1000000**
Requisito 9	**1**
Requisito 10	**100**

Tabela 6.6 – Pesos utilizados pelo algoritmo

Vale ressaltar que não há conhecimento por parte do autor deste trabalho acerca do procedimento adotado pelos especialistas que produziram as tabelas de jogos praticadas no ano de 2002 e no ano de 2003. Pode ter havido apenas a preocupação com a geração de uma tabela viável, seguida ou não por técnicas de refinamento em busca do atendimento de objetivos, não necessariamente os mesmos adotados neste trabalho. Sendo assim, as soluções que estão sendo comparadas podem ter sido obtidas a partir de problemas de otimização com formulação distinta. Como, neste trabalho, o problema foi formulado procurando-se considerar aspectos de grande interesse prático, justifica-se a comparação realizada com uma solução adotada na prática, independentes dos detalhes de implementação desta última.

1) Resultados para o Campeonato com 26 times e turno único disputado em 29 rodadas (Campeonato Brasileiro de Futebol de 2002)

Requisito	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5
Requisito 1	0	0	0	0	0
Requisito 2	0	0	0	0	0
Requisito 3	0	0	0	0	0
Requisito 4	0	0	0	0	0
Requisito 5	0	0	0	0	0
Requisito 6	0	0	0	0	0
Requisito 7	0	0	0	0	0
Requisito 8	0	0	0	0	0
Requisito 9	605.398 km	598.088 km	615.587 km	605.992 km	607.093 km
Requisito 10	37.752 km	35.071 km	36.940 km	38.577 km	34.589 km
Função Objetivo	4.380.598	4.105.188	4.309.587	4.463.692	4.065.993
Tempo	00:05:14 min	00:05:14 min	00:04:54 min	00:05:49 min	00:06:32 min
SA	Sim	Sim	Sim	Sim	Sim
BT	Não	Não	Não	Sim	Sim

Tabela 6.7 – Resultados obtidos para o campeonato de 2002 disputado em 29 rodadas

Os resultados apresentados na Tabela 6.7 mostram claramente que a tabela gerada pelo algoritmo representa, em todos os casos acima, uma solução de qualidade muito superior à da tabela oficial praticada em 2002. Além da eliminação de todas as inviabilidades, conseguiu-se cumprir os objetivos de minimização da distância total percorrida e da diferença entre a maior e menor distância percorrida pelos times.

2) Resultados para o Campeonato com 26 times e turno único disputado em 25 rodadas (rodadas completas ou cheias)

Requisito	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5
Requisito 1	0	0	0	0	0
Requisito 2	0	0	0	0	0
Requisito 3	0	0	0	0	0
Requisito 4	0	0	0	0	0
Requisito 5	0	0	0	0	0
Requisito 6	0	0	0	0	0
Requisito 7	0	0	0	0	0
Requisito 8	0	0	0	0	0
Requisito 9	609.651 km	608.508 km	616.087 km	609.476 km	605.243 km
Requisito 10	32.986 km	38.348 km	31.170 km	31.575 km	33.283 km
Função Objetivo	3.908.251	4.443.308	3.733.087	3.766.976	3.933.543
Tempo	00:15:00 min	00:43:23 min	01:29:09 min	00:17:55 min	00:43:56 min
AS	Sim	Sim	Sim	Sim	Sim
BT	Não	Não	Não	Sim	Sim

Tabela 6.8 – Resultados obtidos para o campeonato de 2002 disputado em 25 rodadas

Os resultados apresentados na Tabela 6.8 mostram que, mesmo em turno completo (o que representa uma maior dificuldade de resolução), o algoritmo conseguiu gerar tabelas de qualidade superior à praticada no campeonato de 2002.

3) Resultados para o Campeonato com 24 times e dois turnos (turno e returno) disputados em 46 rodadas (Campeonato Brasileiro de Futebol de 2003)

Requisito	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5
Requisito 1	0	0	0	0	0
Requisito 2	0	0	0	0	0
Requisito 3	0	0	0	0	0
Requisito 4	0	0	0	0	0
Requisito 5	0	0	0	0	0
Requisito 6	0	0	0	0	0
Requisito 7	0	0	0	0	0
Requisito 8	0	0	0	0	0
Requisito 9	1.253.318 km	1.248.887 km	1.241.942 km	1.248.021 km	1251237 km
Requisito 10	65.573 km	72.625 km	65.433 km	63.681 km	71350 km
Função Objetivo	7.810.618	8.511.387	7.785.242	7.616.121	8.386.237
Tempo	00:34:50 min	00:14:21 Min	00:46:21 min	00:14:10 min	00:17:40 min
SA	Sim	Sim	Sim	Sim	Sim
BT	Não	Não	Não	Sim	Sim

Tabela 6.9 – Resultados obtidos para o campeonato de 2003 disputado em turno e returno

Os resultados apresentados pela Tabela 6.9 mostram que o algoritmo conseguiu, em todos as execuções, gerar uma tabela de jogos atendendo a todos os requisitos (seja eles essenciais ou não-essenciais) e ainda melhorar a distância total percorrida pelos times, bem como a diferença entre a distância percorrida pelo time que mais se deslocou e a distância percorrida pelo time que menos se deslocou, em relação à tabela oficial praticada em 2003.

4) Resultados para o Campeonato com 20 times e dois turnos (turno e returno) disputados em 38 rodadas (Previsão para os Campeonatos Brasileiros de Futebol a partir de 2006)

Requisito	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5
Requisito 1	0	0	0	0	0
Requisito 2	0	0	0	0	0
Requisito 3	0	0	0	0	0
Requisito 4	0	0	0	0	0
Requisito 5	0	0	0	0	0
Requisito 6	0	0	0	0	0
Requisito 7	0	0	0	0	0
Requisito 8	0	0	0	0	0
Requisito 9	618.231 km	632.577 km	633.206 km	635.390 km	612.794 km
Requisito 10	32.177 km	39.315 km	35.932 km	38.675 km	39.855 km
Função Objetivo	3.835.931	4.564.077	4.226.406	4.502.890	4.598.294
Tempo	00:07:21 min	00:11:52 min	00:27:00 min	00:56:28 min	00:48:41 min
SA	Sim	Sim	Sim	Sim	Sim
BT	Não	Não	Não	Sim	Sim

Tabela 6.10 – Resultados obtidos para os campeonatos disputados por 20 times em turno e returno

Para o campeonato disputado por 20 times o algoritmo obteve soluções sem qualquer inviabilidade, gerando tabelas viáveis em todos as execuções.

7. Conclusões

7.1. Análise dos Resultados e Conclusões

Nesta seção são feitas discussões acerca dos resultados obtidos. São apresentadas as melhoras obtidas em relação às tabelas oficiais e demonstram-se os valores que comprovam o desempenho do algoritmo proposto. Os desvios apresentados são obtidos a partir da seguinte fórmula:

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

7.1.1. Campeonato com 26 Times e Turno Único Disputado em 29 Rodadas

Melhores Valores das Funções Objetivo				
Execução 1	**Execução 2**	**Execução 3**	**Execução 4**	**Execução 5**
4.380.598	**4.105.188**	**4.309.587**	**4.463.692**	**4.065.993**

Tabela 7.1 – Função objetivo das execuções para o campeonato com 26 times e 29 rodadas

Desempenho do Algoritmo	
Melhor Função objetivo encontrada	**4.065.993**
Média	**4265011,6**
Desvio	**4,9 %**

Tabela 7.2 – Análise de desempenho do algoritmo para o campeonato com 26 times e 29 rodadas

Melhoras Obtidas pela Melhor Solução Encontrada em Relação à Tabela Oficial Praticada 2002	
Requisito	**Melhora**
Requisito 5	**Inviabilidade Eliminada**
Requisito 7	**Ocorrências Eliminadas**
Requisito 8	**Ocorrências Eliminadas**
Requisito 9	**3,37 %**
Requisito 10	**33,1 %**
Função Objetivo	**99,6 %**

Tabela 7.3 – Melhora obtida em relação à tabela oficial praticada em 2002

Pode-se notar que a melhor solução encontrada apresenta uma melhora significativa em relação à tabela oficial praticada em 2002. Em todas as soluções não houve a ocorrência de nenhum tipo de inviabilidade, além da melhora na distância total percorrida (21.143 km a menos) e na diferença entre a maior e a menor distância percorrida pelos times (17.120 km a menos).

Além disso, em todos os testes o algoritmo conseguiu atender os requisitos essenciais, ou seja, obteve, em cada caso, tabelas de jogos viáveis, ou seja, praticáveis.

Pela análise da Tabela 7.2 verifica-se que a técnica de solução proposta se mostrou

robusta, pois partindo de diferentes soluções iniciais chega-se a soluções finais que diferem da melhor solução conhecida em apenas 4,9%.

Portanto, os resultados obtidos validam a utilização dessa técnica para a resolução do Problema de Programação de Jogos do Campeonato Brasileiro de Futebol disputado em turno único.

Uma tabela de jogos encontrada para este tipo de campeonato é apresentada no Apêndice A.

7.1.2. Campeonato com 26 Times e Turno Único Disputado em 25 Rodadas

Melhores Valores das Funções Objetivo				
Execução 1	**Execução 2**	**Execução 3**	**Execução 4**	**Execução 5**
3.908.251	**4.443.308**	**3.733.087**	**3.766.976**	**3.933.543**

Tabela 7.4 – Função objetivo das execuções para o campeonato com 26 times e 25 rodadas

Desempenho do Algoritmo	
Melhor Valor da Função de Avaliação	**3.733.087**
Média	**3.957.033**
Desvio	**6,0 %**

Tabela 7.5 – Análise de desempenho do algoritmo para o campeonato com 26 times e 25 rodadas

Melhoras Obtidas pela Melhor Solução Encontrada em Relação à Tabela Oficial Praticada 2002	
Requisito	**Melhora**
Requisito 5	**Inviabilidade Eliminada**
Requisito 7	**Ocorrências Eliminadas**
Requisito 8	**Ocorrências Eliminadas**
Requisito 9	**1,93 %**
Requisito 10	**39,7 %**
Função Objetivo	**99,6 %**

Tabela 7.6 – Melhora obtida, com turno completo, em relação à tabela oficial praticada em 2002

Nota-se que a melhor solução encontrada apresenta uma melhora significativa em relação à tabela oficial praticada em 2002. Vale ainda ressaltar que esta solução possui as rodadas completas, ou seja, todo time joga em toda rodada, o que aumenta muito a dificuldade de se encontrar uma solução viável. Além disso, em todas as soluções não houve a ocorrência de nenhum tipo de inviabilidade. A melhora na distância total percorrida (12.149 km a menos) e na diferença entre a maior e a menor distância percorrida pelos times (20.539 km a menos) foi significativa.

Em todos os testes realizados o algoritmo conseguiu atender os requisitos essenciais gerando tabelas de jogos viáveis.

Pela análise da Tabela 7.5 verifica-se que a técnica de solução proposta se mostrou robusta, pois partindo de diferentes soluções iniciais chega-se a soluções finais que diferem da melhor solução conhecida em apenas 6,0 %.

Uma tabela de jogos encontrada para este tipo de campeonato é apresentada no Apêndice B.

7.1.3. Campeonato com 24 Times e Dois Turnos (turno e returno) Disputados em 46 Rodadas

Melhores Valores das Funções Objetivo				
Execução 1	**Execução 2**	**Execução 3**	**Execução 4**	**Execução 5**
7.810.618	**8.511.387**	**7.785.242**	**7.616.121**	**8.386.237**

Tabela 7.7 – Função objetivo das execuções para o campeonato com 24 times e turno e returno

Desempenho do Algoritmo	
Melhor Valor da Função de Avaliação	**7.616.121**
Média	**8.021.921**
Desvio	**5,3 %**

Tabela 7.8 – Análise de desempenho para o campeonato com 24 times e turno e returno

Melhoras Obtidas pela Melhor Solução Encontrada em Relação à Tabela Oficial Praticada 2003	
Requisito	**Melhora**
Requisito 9	**9,55 %**
Requisito 10	**34,83 %**
Função Objetivo	**31,70 %**

Tabela 7.9 – Melhora obtida em relação à tabela oficial praticada em 2003

Pode-se notar que a melhor solução encontrada para este tipo de campeonato apresenta uma melhora expressiva na distância total percorrida (131.624 km a menos) e na diferença entre a maior e a menor distância percorrida pelos times (34.031 km a menos).

Além disso, em todos os testes o algoritmo conseguiu atender os requisitos essenciais, ou seja, obteve, em cada caso, tabelas de jogos viáveis, portanto praticáveis.

Pela análise da Tabela 7.8 verifica-se que a técnica de solução proposta se mostrou robusta, pois partindo de diferentes soluções iniciais chega-se a soluções finais que diferem da melhor solução conhecida em apenas 5,3%.

Conclui-se então que os resultados obtidos validam a utilização dessa técnica para a resolução do Problema de Programação de Jogos do Campeonato Brasileiro de Futebol disputado em turno e returno.

Uma tabela de jogos encontrada para este tipo de campeonato é apresentada no Apêndice C.

7.1.4. Campeonato com 20 Times e Dois Turnos (turno e returno) Disputados em 38 Rodadas

Melhores Valores das Funções Objetivo				
Execução 1	**Execução 2**	**Execução 3**	**Execução 4**	**Execução 5**
3.835.931	**4.564.077**	**4.226.406**	**4.502.890**	**4.598.294**

Tabela 7.10 – Função objetivo das execuções para o campeonato com 20 times e turno e returno

Desempenho do Algoritmo	
Melhor Valor da Função de Avaliação	**3.835.931**
Média	**4.345.516,6**
Desvio	**13,3 %**

Tabela 7.11 – Análise de desempenho para o campeonato com 20 times e turno e returno

O algoritmo conseguiu em todos os casos gerar uma tabela de jogos sem nenhuma inviabilidade, mostrando também robusto para este tipo de campeonato (apesar de obter um desvio de 13,3 % em relação à melhor solução obtida).

Obs: Os dados utilizados para realizar a montagem da tabela para o campeonato disputado por 20 times são os mesmos utilizados para o campeonato de 2002, porém com o número de times reduzido para 20. Os times que participaram desta simulação foram escolhidos seguindo a ordem de disposição no arquivo de 2002 (Apêndice A), ou seja, foram escolhidos os 20 primeiros times deste campeonato.

Uma tabela de jogos encontrada é apresentada no Apêndice D.

7.2. Sugestões Para Trabalhos Futuros

- Implementar outros requisitos que não foram abordados neste trabalho, com o objetivo de melhorar ainda mais as soluções obtidas. Um exemplo de possível melhoria seria agrupar um certo número de clássicos em uma única rodada, fato esse observado na tabela do campeonato disputado em 2003;
- Utilizar outros métodos na resolução do Problema de Programação de Jogos visando à comparação com o *Simulated Annealing* e a abordagem híbrida com o Busca Tabu.
- Ao invés de abordar somente a minimização da distância percorrida pelos times, abordar a minimização dos gastos com o deslocamento dos mesmos e compará-los com os gastos com manutenção dos times fora de suas sedes.
- Fazer uma busca local na solução final produzida pelo sistema, conforme sugerido por Roberto Carlos Vieira Pontes em comunicação pessoal. A idéia proposta consiste em incluir no algoritmo descrito à seção 4.8 o seguinte passo:

  "Passo 5 - Refinamento da Solução por Enumeração Restrita de Tabelas Simétricas:

  A solução final obtida representa um molde que pode conter inúmeras soluções simétricas, realizando-se permutações que correspondem à troca dos números dos clubes em cada nova solução.

  Se todos os clubes pertencessem a estados diferentes, para o problema de 24 times, teríamos 24! soluções diferentes a partir do quadro numérico da solução final obtida pelo passo anterior. Todas essas permutações corresponderiam a soluções que podem ter valores diferentes para a função objetivo do problema.

  É interessante observar que a maioria das restrições permanecem sem ser violadas em cada sucessiva permutação, com exceção dos requisitos que se referem a jogos de estados diferentes na última rodada, para o problema de turno único e de turno e returno, e a restrição da quantidade de jogos em casa e fora, segundo critérios de ranking da CBF, para os problemas de turno único.

  O número de soluções diferentes que podem ser obtidas a partir de um conjunto de 24 clubes, onde 6 são de São Paulo, 3 do Rio de Janeiro, 3 do Rio Grande do Sul, 3 do Paraná, 2 de Minas Gerais, 2 da Bahia , 2 de Santa Catarina, 1 de Belém, 1 do Ceará e 1 de Goiás, é dado pela fórmula da Permutação com Repetição:

  $$P^{24}_{6,3,3,3,2,2,2} = \frac{24!}{6!3!3!3!2!2!2!} = 498.688.594.500.096.000 \text{ permutações diferentes.}$$

  Como o número de permutações a serem avaliadas é muito grande, o algoritmo poderia realizar apenas uma enumeração restrita dessas combinações, por exemplo, em torno de 1 milhão, guardando sempre a melhor solução obtida".

Referências Bibliográficas

BASTOS, M. P. & RIBEIRO, C.C. - Reactive Tabu Search with Path Relinking for the Steiner Problem in Graphs. In *Proceedings of the Third Metaheuristics International Conference*, pages 31-36, Angra dos Reis, Brazil, 1999.

BATTITI, R. - Reactive Search: Toward Self-Tuning Heuristics. In V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, and G.D. Smith, editors, *Modern Heuristic Search Methods*, chapter 4, pages 61-83. John Wiley & Sons, New York, 1996.

BATTITI, R. & TECCHIOLLI, G. - The Reactive Tabu Search. *ORSA Journal of Computing*, 6:126-140, 1994.

BEAN, J. C. & BIRGE, J. R. - Reducting Travelling Costs and Player Fatigue in National Basketball Association. Interfaces 10;3, p.98-102, 1980.

CONCILIO, R. - *Contribuições à solução de problemas de escalonamento pela aplicação Conjunta de computação evolutiva e otimização com restrições*. Dissertação de Mestrado, Departamento de Engenharia de Computação e Automação Industrial - Faculdade de Engenharia Elétrica e de Computação – UNICAMP, Campinas, SP: [s.n.], 2000.

CONCÍLIO, R. & ZUBEN, F. J. – Uma Abordagem Evolutiva para Geração Automática de Turnos Completos em Torneios. *Revista Controle & Automação*. Vol. 13, n. 2, p. 105-122, 2002.

COSTA, D. – An Evolutionary Tabu Search Algorithm and the NHL Scheduling Problem. *INFORS*. Vol. 33, n. 3, p. 161-178, 1995.

DAMMEYER, F. & VOB, S. - Dynamic tabu list management using the reverse elimination method. In P.L. Hammer, editor, Tabu Search, volume 41 of *Annals of Operations Research*, pages 31-46. Baltzer Science Publishers, Amsterdan, 1993.

DOWSLAND, K. A. - Simulated Annealing. In C.R. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, *Advanced Topics in Computer Science Series*, chapter 2, p 20-69. Blackwell Scientific Publications, London, 1993.

DOWSLAND, K.A. SIMULATED ANNEALING - C.R. Reeves, editor, Modern Heuristic Techniques for Combinatorial Problems, Advanced Topics in *Computer Science Series*, chapter 2, pages 20-69. Blackwell Scientific Publications, London, 1993.

GLOVER, F. - Future paths for Integer Programming and links to Artificial Intelligence. *Computers and Operations Research*, 5:553-549, 1986.

GLOVER, F. - Tabu Search: Part I. *ORSA Journal of Computing*, 1:190-206, 1989.

GLOVER, F. - Tabu Search: Part II. *ORSA Journal of Computing*, 2:4-32, 1990.

GLOVER, F. & LAGUNA, M. - Tabu Search. In C.R. Reeves, editor, Modern Heuristic Techniques for Combinatorial Problems, *Advanced Topics in Computer Science Series*, chapter 3, pages 70-150. Blackwell Scientific Publications, London, 1993.

GLOVER, F. & LAGUNA, M. - Tabu Search. *Kluwer Academic Publishers*, Boston, 1997.

GLOVER, F.; TAILLARD, E. & WERRA, D. - A user's guide to tabu search. In P.L. Hammer, editor, Tabu Search, volume 41 of *Annals of Operations Research*, pages 3-28. Baltzer Science Publishers, Amsterdan, 1993.

HAMIEZ, J. P. & HAO, J. K. - Solving the Sports League Scheduling Problem with Tabu Search.

HANAFI, S. - On the Convergence of Tabu Search. To appear in *Journal of Heuristics.*

HANSEN, P. - The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy, 1986.

HENZ, M. - Scheduling a Major College Basketball Conference - Revisited. In: *Operations Research* 49/1, 2001.

HERTZ, A. & WERRA, D. - The tabu search metaheuristic: how we used it. *Annals of Mathematics and Artificial Intelligence*, 1: p.111-121, 1990.

KIRKPATRICK, S.; GELLAT, D. C. & VECCHI, M. P. - Optimization by Simulated Annealing. *Science*,. Vol. 220, p. 671-680, 1983.

KIRKPATRICK, S.; GELLAT, D.C. & VECCHI, M.P. - Optimization by Simulated Annealing. Science, 220:671-680, 1983.

MIYASHIRO, R.; IWASAKI, H. & MATSUI, T. – Characterizing Feasible Pattern Sets with a Minimun Number of Breaks, In: *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling* (PATAT 2002), p. 311-313, 2002.

NEMHAUSER, G. L. & TRICK, M. A. – Scheduling a Major College Basketball Conference. *Operations Research*,. Vol. 46, n. 1, p. 1-8, 1998.

RIBEIRO, C.C. - Metaheuristics and Applications. In *Advanced School on Artificial Intelligence*, Estoril, Portugal, 1996.

SCHAEFER, A. - Tabu search techniques for large high-school timetabling problems. In *Proceedings of the 30th National Conference on Artificial Intelligence*, pages 363-368, 1996.

SOUZA, M. J. F. – Inteligência Computacional para Otimização, Notas de Aula, Departamento de Computação, Universidade Federal de Ouro Preto, MG. Disponível em: http://www.decom.ufop.br/prof/marcone.

TRICK, M. A. - A Schedule-Then-Break Approach to Sports Timetabling, *Lecture Notes in Computer Science*. Vol. 2079, p. 242-253, 2000.

WERRA, D. - Tabu Search Techniques: A Tutorial and an Application to Neural Networks. *OR Spektrum*, 11:131-141, 1989.

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO - DECOM
CAMPUS UNIVERSITÁRIO - MORRO DO CRUZEIRO
CEP 35400-000 - OURO PRETO - MINAS GERAIS