UMA ABORDAGEM EVOLUTIVA PARA GERAÇÃO AUTOMÁTICA DE TURNOS COMPLETOS EM TORNEIOS

Ricardo Concilio[*]
ricardo.concilio@maua.br

Fernando J. Von Zuben[†]
vonzuben@dca.fee.unicamp.br

[*]Departamento Fundamental, Escola de Engenharia Mauá, São Caetano do Sul, SP – Brasil, 09580-900

[†]Faculdade de Engenharia Elétrica e de Computação (DCA/FEEC), Universidade Estadual de Campinas (Unicamp), C. P. 6101, 13083-970, Campinas, SP – Brasil

ABSTRACT

This paper presents contributions to the solution of assignment problems, more precisely to the generation of a complete set of rounds in tournaments. It represents a practical problem of high interest, being characterized by feasibility aspects and a combinatorial explosion of solution candidates. In this case, the direct actuation of an expert and the use of conventional search tools generally guide to unsatisfactory results. The proposed solution strategy is based on the joint application of evolutionary computation, local search and restriction-based optimization. Although other evolutionary approaches have already been proposed in the literature, the one considered here innovates, since it suggests a compact genetic codification in conjunction with an algorithm to expand the code. When compared with the solutions already implemented to deal with real-world assignment problems, the ones obtained from the solution strategy proposed in this work presented better performance and the required amount of computational resource to produce the solution is reasonable. The joint application of evolutionary computation, local search and restriction-based optimization may be extended to deal with other assignment problems, assuming the existence of a compact genetic codification and the availability of an algorithm for restriction-based optimization.

KEYWORDS: Evolutionary computation, constraint optimization, compact representation, code expansion, local search.

RESUMO

Este artigo apresenta contribuições junto à solução de problemas de escalonamento, mais precisamente na geração de turnos completos em torneios. Trata-se de um problema de grande interesse prático, caracterizado por questões de factibilidade e uma explosão combinatória de candidatos à solução. Sendo assim, a atuação direta de um especialista e a aplicação de ferramentas convencionais de busca geralmente não conduzem a resultados satisfatórios. A estratégia de solução proposta está baseada na aplicação conjunta de computação evolutiva, busca local e otimização baseada em restrições. Embora outras abordagens evolutivas já tenham sido propostas na literatura, a empregada aqui inova ao sugerir uma representação genética compacta aliada a um algoritmo de expansão de código. Comparadas às soluções já implementadas para problemas reais de escalonamento, aquelas obtidas a partir da estratégia de solução proposta neste trabalho apresentaram melhor desempenho e a quantidade de recursos computacionais requeridos para produzir a solução é aceitável. A aplicação conjunta de computação evolutiva, busca local e técnicas de otimização baseada em restrições pode ser estendida ao tratamento de outros problemas de escalonamento, supondo a existência de uma codificação genética compacta e a disponibilidade de um algoritmo de otimização baseado em restrições.

Artigo submetido em 20/12/00
1a. Revisão em 29/05/01
Aceito sob recomendação do Ed. Assoc. Prof. Paulo E. Miyagi

PALAVRAS-CHAVE: Computação evolutiva, otimização com restrições, representação compacta, expansão de código, busca local.

1 INTRODUÇÃO

Problemas de otimização, com restrições de diversas naturezas, envolvem basicamente uma função-objetivo a ser minimizada e um conjunto de restrições que devem ser simultaneamente atendidas, na forma:

$$\begin{cases} \min\limits_{x} & < \text{função - objetivo dependente de } x > \\ s.a. & < \text{conjunto de restrições impostas a } x > \end{cases}$$

Duas etapas estão envolvidas no processo de obtenção da solução (Sherali *et al.*, 1993):

1. formulação matemática do problema, procurando descrever todos os aspectos relevantes, seja em termos de objetivos a serem atingidos ou restrições a serem atendidas;
2. obtenção da solução pela aplicação de ferramentas de otimização, desenvolvidas a partir da formulação matemática do problema.

Dependendo da natureza do problema, a sua formulação leva a um processo de solução que pode ser expresso algebricamente, de forma fechada. No entanto, grande parte dos problemas de otimização apresentam características (por exemplo, não-linearidades) que impedem a obtenção de uma solução na forma fechada, requerendo a aplicação de processos iterativos de busca da solução, a partir de uma condição inicial. A busca iterativa é geralmente implementada em computador, devido ao elevado custo de processamento associado à sua execução.

A presença de restrições de diversas naturezas introduz uma complexidade adicional, por dividir o espaço de candidatos à solução em duas classes: soluções factíveis (que atendem a todas as restrições simultaneamente) e soluções infactíveis (que violam uma ou mais restrições).

Além disso, outra questão a ser considerada é o comportamento de convergência do processo iterativo. Três condições são desejáveis: garantia de convergência, tempo de convergência compatível com as necessidades de cada aplicação e convergência para uma solução de boa qualidade. Basicamente, quanto mais complexo o problema de otimização e quanto maior seu tamanho (número de variáveis envolvidas), menor a chance de ter atendidas estas três condições simultaneamente, principalmente quando o método de otimização empregado não é suficientemente poderoso na execução do processo de busca da solução.

Neste artigo, serão considerados justamente problemas com elevado grau de complexidade, com múltiplos objetivos devidamente ponderados, múltiplas restrições e grande número de variáveis, apresentando uma explosão combinatória de candidatos à solução, de modo que uma busca exaustiva pela solução ótima, dentre as soluções candidatas, representa um procedimento computacionalmente intratável (Papadimitriou & Steiglitz, 1982).

A tratabilidade de problemas de otimização com este grau de complexidade é conseguida caso se abra mão da solução ótima, em prol da obtenção de uma solução de boa qualidade, embora muitas vezes não se tenha condições de saber o quão distante da solução ótima se encontra a solução obtida.

Dentre outras técnicas, a computação evolutiva tem se destacado no tratamento deste tipo de problema (Bäck *et al.*, 1997), pelas seguintes razões (Goldberg, 1989):

- por trabalhar com múltiplos candidatos à solução ao mesmo tempo em uma dada geração;
- por privilegiar melhores soluções candidatas na composição das próximas gerações;
- por ser capaz de detectar regiões promissoras no espaço de soluções e atingir soluções de boa qualidade sem a necessidade de avaliar um grande número de candidatos;
- por permitir a aplicação de procedimentos de busca local, de modo a otimizar localmente as soluções propostas pelo processo evolutivo.

A grande dificuldade na aplicação de computação evolutiva a problemas multi-objetivo e com múltiplas restrições é como chegar a uma representação genotípica das soluções candidatas e como obter novas soluções candidatas que sejam factíveis, a partir da aplicação de operadores genéticos (Coello, 1999; Michalewicz, 1997, C5.5). Uma das contribuições deste trabalho está na solução deste problema de representação e na produção de soluções factíveis para o caso específico de geração de turnos completos em torneios de competição, baseados em disputas envolvendo dois participantes adversários. Este tipo de torneio se manifesta em muitos contextos, incluindo obviamente campeonatos e competições, mas também procedimentos de laboratório envolvendo manipulações de componentes dois a dois, estratégias de *marketing* e análise multivariada. O que se busca essencialmente é aplicar um procedimento sistemático de avaliação da qualidade relativa de cada componente de um determinado conjunto finito por comparação individual com todos os elementos do mesmo conjunto.

1.1 Geração Automática de Turnos Completos em Torneios

Em torneios envolvendo disputas dois a dois entre os participantes, sendo cada disputa denominada um jogo do torneio, usualmente devem ser atendidas as seguintes restrições básicas:

- cada participante deve jogar contra todos os outros uma única vez;
- cada participante deve obrigatoriamente jogar uma única vez na mesma rodada, a qual representa o menor conjunto de jogos que inclua todos os participantes;
- supondo que existe distinção de mando de jogo, a diferença entre o número de jogos no domínio de cada participante e fora dele deve ser no máximo de 1, considerando todas as rodadas.

Se n é o número de participantes, então um turno completo será composto por $n-1$ rodadas, sendo que existe uma explosão combinatória de candidatos factíveis à solução (turnos completos). É suposto aqui que n é par. Na eventualidade de n ser ímpar, basta introduzir um participante fictício, indicando que seu adversário naquela rodada irá folgar.

Em face da existência de um grande número de candidatos factíveis à solução, é necessário estabelecer os objetivos a serem otimizados, de modo que existam soluções factíveis de melhor qualidade, ou seja, que maximizam o atendimento dos objetivos, as quais devem ser encontradas por um processo de busca iterativa. Neste trabalho, os objetivos a serem atendidos são os seguintes:

- minimizar o número de vezes em que qualquer participante do torneio jogue r ou mais vezes seguidas dentro (ou fora) de seus domínios, sendo que o valor de r depende de n;
- minimizar a diferença entre o maior e o menor percurso a ser realizado pelos participantes do torneio para completar seus jogos. Uma medida de variância entre os percursos poderia também ser minimizada neste caso.

1.2 Aspectos Gerais da Estratégia de Solução

Para viabilizar a aplicação da estratégia de solução proposta neste trabalho, serão desenvolvidos um código genético compacto e um algoritmo de expansão de código associado a um gerador de números pseudo-aleatórios.

É fundamental que se garanta a produção de um único e mesmo fenótipo (turno completo) para cada representação compacta, justificando a existência de uma única semente para cada indivíduo e da propriedade de repetitividade do gerador de números pseudo-aleatórios (veja Apêndice A), a ser empregado no processo de expansão de código. Uma vez que o código genético compacto é sempre um indivíduo factível e que os operadores genéticos sejam aplicados somente a ele, os descendentes gerados também serão factíveis.

O processo de geração de soluções factíveis, pela aplicação do algoritmo de expansão de código, se dá pela implementação de sofisticados mecanismos de reparação, com convergência garantida. É importante ressaltar que o problema estudado tem características próprias que indicam o uso da computação evolutiva juntamente com processos de busca local, conduzindo aos algoritmos meméticos (Moscato, 1989). Sendo assim, além da expansão responsável por produzir um código factível, uma busca local é empregada para refinar o código na direção de ótimos locais. Uma vez que as soluções factíveis da geração corrente tenham sido obtidas e refinadas localmente, uma medida da sua qualidade poderá ser obtida e utilizada nas etapas subseqüentes do processo evolutivo.

2 ABORDAGEM EVOLUTIVA PARA PROBLEMAS COM RESTRIÇÕES

2.1 Considerações Iniciais

Conforme Eiben & Ruttkay (1997, C5.7), a aplicação de algoritmos evolutivos na solução de problemas com restrições que devem ser atendidas (*constraint-satisfaction problem – CSP*) é interessante sob dois pontos de vista. Por um lado, não se pode esperar que um algoritmo clássico de busca possa alcançar com facilidade a solução de um *CSP*. Determinadas heurísticas de busca apresentam-se bastante eficazes em certos casos, mas falham em outros por restringir o escopo da busca. Ainda segundo Eiben & Ruttkay (1997, C5.7), algumas instâncias de *CSPs* podem ser resolvidas pela diversificação da busca, pela manutenção de vários candidatos à solução em paralelo e pela aplicação de heurísticas que agreguem mecanismos de construção aleatória de novos candidatos à solução. Como esses princípios são essenciais dentro de algoritmos evolutivos, a sua aplicação para a resolução de *CSPs* apresenta-se promissora.

Por outro lado, algoritmos evolutivos são tradicionalmente utilizados em problemas de otimização que não apresentam nenhum tipo de restrição. Como os operadores convencionais de recombinação e mutação não tratam diretamente restrições, a aplicação de algoritmos evolutivos para problemas com estas características não é imediata. Uma extensão desse fato é a impossibilidade de garantir que pais geneticamente factíveis gerem descendentes também factíveis.

2.2 Definições e notação utilizada

A seguir, serão apresentadas algumas definições para posicionar a terminologia utilizada na solução de problemas envolvendo restrições (para maiores detalhes, veja Eiben & Ruttkay, 1997, C5.7).

Definição 1: $S = D_1 \times ... \times D_n$ é denominado espaço de busca livre, sendo obtido a partir de um produto cartesiano dos conjuntos D_i, i=1,...,n.

Definição 2: Um problema de otimização sem restrições (*free optimization problem -FOP*) é um par (S,f), sendo S um espaço de busca livre e f uma função-objetivo em S, devendo ser otimizada (minimizada ou maximizada). A solução de um *FOP* é um elemento $s \in S$ com um valor ótimo de f.

Definição 3: Um problema com restrições que devem ser atendidas (*constraint-satisfaction problem - CSP*) é um par (S,ϕ), sendo S um espaço de busca livre e ϕ uma função booleana em S. A solução de um *CSP* é um $s \in S$ com $\phi(s) =$ verdadeiro.

Definição 4: Um problema de otimização com restrições (*constraint optimization problem - COP*) é uma tripla (S,f,ϕ), sendo S um espaço de busca livre, f uma função-objetivo em S e ϕ uma função booleana em S. A solução de um *COP* é um $s \in S$, com um valor ótimo de f tal que $\phi(s)$=verdadeiro.

Definição 5: Para o caso de *CSPs* e *COPs*, ϕ será chamada de condição de factibilidade e o conjunto $\{s \in S$ / $\phi(s) =$ verdadeiro$\}$ será o espaço de busca factível.

Por intermédio dessas definições, pode-se representar os problemas *FOP*, *CSP* e *COP* respectivamente pelas seguintes notações: (S,f,•), (S,•,ϕ) e (S,f,ϕ), sendo que o símbolo • implica a ausência do argumento correspondente.

2.3 Transformação de CSPs em Problemas que Admitem Solução via Algoritmos Evolutivos

É sabido que a presença de uma função-objetivo (função de *fitness*) a ser otimizada é essencial em algoritmos evolutivos. Pela Definição 3 do item anterior, percebe-se que para um *CSP* (representado pela ênupla (S,•,ϕ)) isto não ocorre.

Para que um algoritmo evolutivo seja aplicado nesse caso, antes será necessário transformá-lo em um *FOP* - (S,f,•) - ou um *COP* - (S,f,ϕ) -, embora esta transformação nem sempre seja possível ou viável.

Definição 6: Considere os problemas P_1 e P_2 como sendo (S,f,•), (S,•,ϕ) ou (S,f,ϕ). P_1 e P_2 são equivalentes se:

$\forall\ s \in S$: s é uma solução de $P_1 \ \Leftrightarrow\ s$ é uma solução de P_2.

Diz-se que P_1 está contido em P_2 se:

$\forall\ s \in S$: s é uma solução de $P_1 \ \Rightarrow\ s$ é uma solução de P_2.

Assim, a solução de um *CSP* por um algoritmo evolutivo requer que este seja transformado em um *FOP* ou um *COP* que o contém e, em seguida, resolver este novo problema.

Vale o comentário de que *FOPs* permitem uma busca livre, de modo que não apresentam dificuldades para os algoritmos evolutivos. Em contrapartida, *COPs* são de difícil solução quando se recorre apenas aos algoritmos evolutivos em sua forma tradicional.

2.4 Espaços de Busca com Restrições

Michalewicz (1997, C5.1) argumenta que muitos dos problemas que apresentam um domínio discreto, tais como *Knapsack problem*, *set covering problem*, *vehicle routing problem* e todos os tipos de escalonamento, *scheduling* e *timetabling*, contêm uma série de restrições que devem ser atendidas simultaneamente.

Acrescenta, ainda, que em geral um espaço de busca S consiste de dois subconjuntos disjuntos de subespaços: factíveis (F) e infactíveis (U), sendo que no decorrer do processo de busca, a população de indivíduos candidatos à solução pode conter elementos que sejam tanto factíveis como infactíveis. A Figura 1 ilustra candidatos à solução caracterizados como factíveis, infactíveis e uma solução ótima. Conforme Michalewicz (1997, C5.1), o problema de como trabalhar com indivíduos infactíveis está longe de ser trivial. De modo geral, duas funções de avaliação devem ser elaboradas: uma delas fará referência ao domínio das soluções factíveis (*eval*$_f$) e a outra às infactíveis (*eval*$_u$), tal que:

$$\text{eval}_f : F \rightarrow \mathbb{R} \quad \text{e} \quad \text{eval}_u : U \rightarrow \mathbb{R}.$$

Dentro desse contexto, há várias técnicas de como trabalhar com espaços de busca restritivos, sendo que cada uma delas apresenta uma abordagem distinta. A seguir, será apresentada uma breve discussão dessas técnicas.

2.4.1 Funções com Penalidade

Segundo Smith & Coit (1997, C5.2), existem dois tipos básicos de funções de penalidade:

- função de penalidade externa (aplicada às soluções infactíveis);
- função de penalidade interna (aplicada às soluções factíveis).

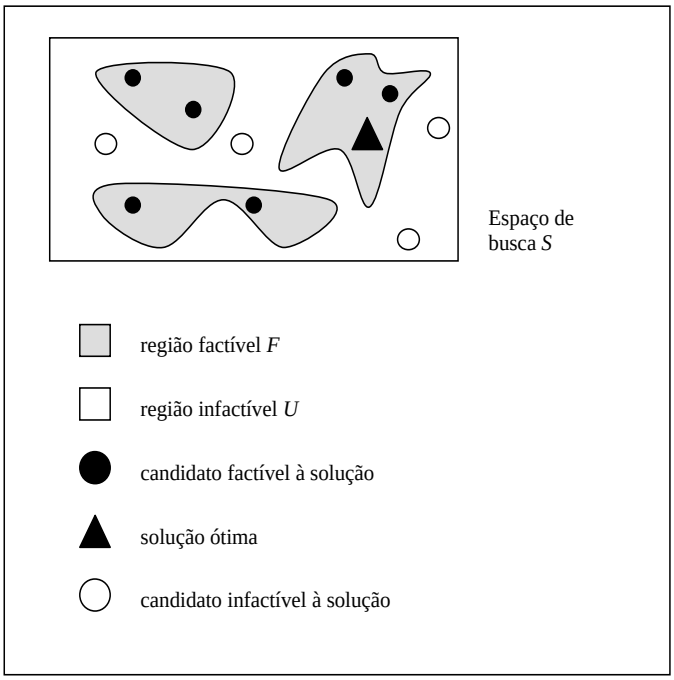

Figura 1: Representação pictórica do espaço de busca S.

Para o caso de função de penalidade interna, a principal idéia envolvida é o fato do problema apresentar uma solução ótima tal que pelo menos uma restrição seja ativada, o que implica em dizer que a solução ótima está no limiar entre a factibilidade e a infactibilidade. Conhecendo-se essa característica, uma penalidade será aplicada às soluções factíveis quando a restrição não for ativada. Ressalta-se que, para o caso de múltiplas restrições, esse tipo de implementação torna-se bastante complexa.

Para funções de penalidade externa, há três tipos de solução:

- método no qual nenhuma solução infactível será considerada;
- função de penalidade parcial, sendo que uma penalidade é aplicada nas proximidades do limite de factibilidade;
- função de penalidade global, sendo que uma penalidade é aplicada na região de infactibilidade.

Dado um problema de otimização, a formulação a seguir caracteriza-se por ser a mais geral:

$$\min_x f(x) \text{ tal que } x \in A \text{ e } x \in B,$$

sendo x um vetor de variáveis de decisão e tal que restrições $x \in A$ são relativamente fáceis de serem atendidas, enquanto restrições $x \in B$ são relativamente difíceis. Como exemplo, o conjunto A pode estar representando variáveis contínuas e o conjunto B variáveis discretas. Assim, o problema pode ser reformulado como:

$$\min_x \ f(x) + p\left(d(x, B)\right) \text{ tal que } x \in A,$$

sendo $d(x, B)$ uma métrica descrevendo a distância do vetor solução x à região B e $p(\cdot)$ sendo uma função de penalidade monotonicamente não-decrescente tal que $p(0) = 0$. Portanto se $x \in B$, então $d(x, B) = 0$, o que leva à ausência de penalidade.

Conforme Smith & Coit (1997, C5.2), várias funções para $p(\cdot)$ têm sido estudadas, como também diferentes tipos de métricas para $d(\cdot)$, incluindo uma contagem do número de restrições violadas. Encontrar uma função de penalidade que seja eficiente é uma tarefa difícil. Grande parte dessa dificuldade está no fato de que a solução ótima freqüentemente está no limiar de uma região de factibilidade. Acrescenta-se, ainda, que ao restringir a busca somente às soluções factíveis, ou ao se impor penalidades muito severas, acaba-se dificultando o encontro da solução ótima. Por outro lado, se a penalidade não for forte o suficiente, tornará a região de busca extremamente ampla e boa parte do tempo do processo será utilizado na exploração de regiões distantes da região de factibilidade.

2.4.2 Decodificadores

Michalewicz (1997, C5.3) afirma que decodificadores oferecem uma opção interessante para as técnicas de computação evolutiva, sendo que um cromossomo (representação do genótipo) fornecerá instruções para um decodificador ou representará a condição inicial para a elaboração de uma solução factível (representação do fenótipo).

Formalmente, um decodificador é uma transformação T de um espaço de elementos codificados (onde não há infactibilidade) para uma região de factibilidade do espaço de soluções candidatas (onde há regiões factíveis e infactíveis). A Figura 2 mostra um exemplo de decodificação, onde a transformação T decodifica um ponto d em um representante s de um subespaço factível.

Será importante considerar que esse processo de decodificação sempre satisfaz as seguintes condições (F é o espaço de soluções factíveis):

- para cada elemento $s \in F$, há pelo menos um elemento d do espaço de elementos codificados;
- a cada elemento d do espaço de elementos codificados corresponde uma única solução factível $s \in F$, ou seja, a transformação T é tal que para cada d está associado um único $s \in F$.

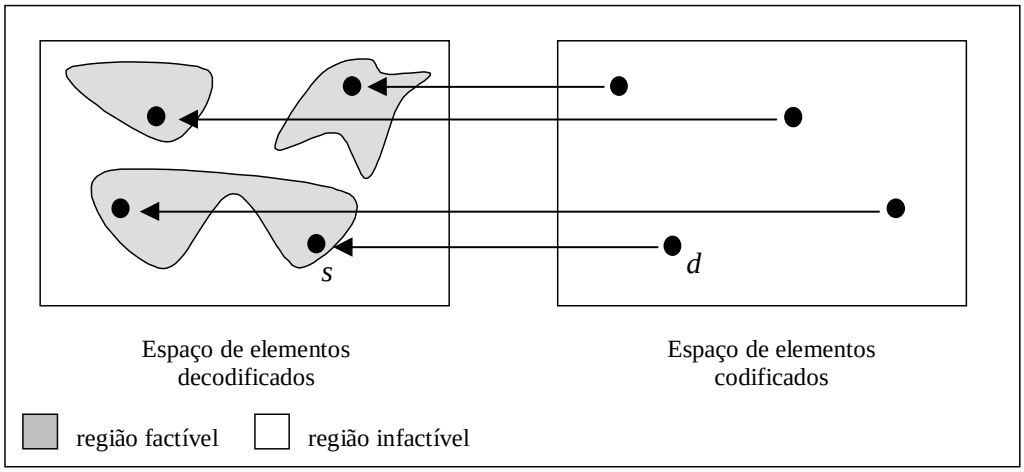

Figura 2: Transformação T que leva elementos codificados a soluções factíveis.

Também é desejável que a transformação T:

- seja computacionalmente rápida;
- deva possuir a característica de que pequenas alterações em elementos do espaço de elementos codificados resultem em pequenas alterações no elemento correspondente do espaço de factibilidade.

2.4.3 Algoritmos de Reparação

Segundo Michalewicz (1997, C5.4), a aplicação de algoritmos de reparação é freqüentemente adotada pela comunidade científica que trabalha com computação evolutiva, acrescentando que para muitos problemas de otimização combinatória é possível fazer com que um indivíduo infactível torne-se factível.

Em relação à função de avaliação, a nova configuração será a seguinte: $eval_u(y) = eval_f(x)$, sendo x a versão factível de y, após a aplicação do algoritmo de reparação.

O processo de reparação dos indivíduos pode ser relacionado com a combinação de aprendizado e evolução (efeito *Baldwin* - Whitley *et al*., 1994). O aprendizado é, em geral, uma busca local pela solução factível mais próxima e as modificações ocorridas (o indivíduo infactível é levado para uma região de factibilidade do espaço) serão incorporadas pelo indivíduo ou, então, o indivíduo original não é alterado e um novo elemento será incluído na população com este novo código genético factível.

A desvantagem desse método é a dependência clara em relação ao problema estudado. Em outras palavras, cada problema terá o seu próprio algoritmo de reparação, não existindo um que possa ser generalizado para todos os casos. Acrescenta-se, também, que a heurística utilizada pelo algoritmo deve variar de caso para caso, buscando aquela que melhor se adapta ao contexto.

Michalewicz (1997, C5.4) diz que, para alguns problemas, esse processo de reparação pode ser tão complexo quanto a busca pela solução ótima do próprio problema (em muitos casos, é o que ocorre com problemas de *scheduling*, *timetabling* e outros).

A substituição de indivíduos está relacionada com o lamarckismo (Whitley *et al*., 1994), que supõe que um indivíduo passa por um processo de adaptação no decorrer da sua vida no sentido de aumentar a sua adaptabilidade, sendo o resultado deste processo incorporado ao seu código genético. Ainda segundo Whitley *et al.* (1994), resultados analíticos e empíricos indicam que estratégias computacionais lamarckistas representam um tipo de busca eficaz.

2.4.4 Operadores que Preservam a Factibilidade

Várias pesquisas têm levado à aplicação bem sucedida de operadores genéticos dedicados, tendo a função de preservar a factibilidade dos indivíduos gerados após a sua aplicação (Michalewicz, 1997, C5.5). Percebe-se que essa classe de operadores incorpora um conhecimento específico do problema. Segundo Surry *et al.* (1995), essa proposta tem a finalidade de construção e utilização de operadores genéticos que tenham a capacidade de atendimento das restrições, de modo que nunca produzam soluções infactíveis.

As principais desvantagens apresentadas por este processo são:

- operadores genéticos dedicados são aplicáveis somente aos problemas para os quais foram projetados;
- uma análise formal desse tipo de ambiente torna-se difícil, embora uma grande quantidade de experimentos evidenciem a aplicabilidade desta técnica.

2.4.5 Outros Métodos para Trabalhar com Restrições

Várias outras heurísticas para trabalhar com restrições em conjunto com técnicas evolutivas têm sido propostas nos últimos anos. Na maioria das vezes é difícil a classificação desses métodos, uma vez que são a combinação de outros ou são baseados em idéias originais ainda não completamente formalizadas. Para maiores detalhes, veja Michalewicz (1997, C5.6).

2.5 Abordagem Proposta para o Tratamento de Otimização com Restrições

De acordo com a nomenclatura adotada neste trabalho, a abordagem proposta para o tratamento de otimização com restrições refere-se a problemas do tipo *COP*: problema de otimização com restrições (*constraint optimization problem*).

Na aplicação de algoritmos evolutivos junto a um *COP*, o emprego de uma codificação restrita a um espaço totalmente factível, seguida de um procedimento de decodificação que produz um elemento factível do espaço original, já foi utilizado na literatura (Costa, 1999). Uma generalização da abordagem de Costa (1999) será implementada aqui, com o uso de uma codificação compacta, seguida da aplicação de um algoritmo de expansão de código. A seção 4 vai apresentar todos os detalhes do tratamento adotado para se implementar o processo de solução. Portanto, o propósito desta presente seção é posicionar a abordagem de solução frente à formalização apresentada na seção 2. Deste modo, na seção 4 será possível se concentrar apenas nos detalhes de implementação.

Como é sabido, por um lado, que algoritmos evolutivos em sua composição tradicional encontram dificuldades no tratamento de problemas de otimização com restrições, também é fato, por outro lado, que a aplicação isolada de técnicas de otimização baseada em restrições podem encontrar sérias dificuldades em virtude da existência de ótimos locais de baixa qualidade. Sendo assim, a aplicação conjunta de ambas as metodologias mostra-se promissora pelas seguintes razões:

- O processo evolutivo vai poder atuar somente sobre o código compacto, junto ao qual não há infactibilidade (ou então de modo que a factibilidade possa ser prontamente conquistada), fazendo com que sua eficácia não seja reduzida pela existência de etapas infactíveis;
- embora as técnicas de otimização baseada em restrições, que irão desempenhar o papel de expansão de código, continuem produzindo apenas soluções factíveis com otimalidade local, a existência de uma população de candidatos à solução sendo evoluída, juntamente com a aplicação de procedimentos de busca local, aumentam significativamente as chances de se obter ótimos locais de boa qualidade.

No entanto, para que esta conjugação de técnicas produza melhores resultados que a aplicação isolada de cada uma delas, é preciso estabelecer critérios para definir o nível de compactação da representação genética, de modo a equilibrar o papel a ser desempenhado por ambas as técnicas. Por exemplo, se a compactação for além do que poderia ser considerado como adequado, o espaço de busca para o processo evolutivo será muito reduzido e o papel da expansão de código passaria a ser muito próximo daquele vinculado à sua aplicação isolada. E vice-versa. Existe portanto uma solução de compromisso para o processo de definição do nível de compactação.

Funções de penalidade É utilizada uma função de penalidade externa, que não irá permitir nenhuma solução infactível como solução candidata. Este tipo de penalidade está entre as mais severas, podendo dificultar o encontro da solução ótima caso não venha acompanhada de ferramentas de apoio, como algoritmos de reparação e de busca local.

Decodificadores A representação genética compacta irá desempenhar o papel de condição inicial para um algoritmo de expansão de código, que corresponde essencialmente a um decodificador (veja Figura 2).

Vale aqui um comentário a respeito da seguinte propriedade que deve estar presente em um decodificador para garantir sua completitude:

- para cada elemento factível $s \in F$ no espaço de elementos decodificados, deve existir pelo menos uma condição inicial no espaço de elementos codificados (representação compacta) de modo que o decodificador seja capaz de produzi-lo.

Como será evidenciado na seção 4, o algoritmo de expansão de código, que fará o papel do decodificador, não necessita da propriedade de completitude para operar convenientemente, pois existe uma função-objetivo a ser otimizada. Com isso, já durante o processo de expansão de código (e não apenas

ao final da decodificação), parte dos elementos factíveis será descartada por representar um distanciamento da condição de otimalidade. Além disso, ao término da expansão de código, procedimentos de busca local são empregados, de modo que vai haver uma exploração local da região factível, podendo levar a elementos factíveis que melhor atendam aos objetivos, mesmo que estes não sejam diretamente reprodutíveis a partir da decodificação.

O processo de expansão de código pode ser visto como uma busca em uma árvore de decisão, com etapas de *backtracking*. Como a tomada de decisão vai depender da saída de um gerador de números pseudo-aleatórios com a propriedade de repetitividade, então a completitude do decodificador estaria associada à possibilidade de recorrer a todos os possíveis geradores de números pseudo-aleatórios, ou seja, a todas as possíveis seqüências aleatórias, de modo que qualquer seqüência de ramos da árvore de decisão pudesse ser adotada.

Algoritmos de reparação O algoritmo de expansão de código emprega procedimentos de reparação durante o processo de decodificação, e não ao seu final. Portanto, a reparação está incluída no processo de expansão de código. Além disso, procedimentos de busca local são também aplicados imediatamente após o término da etapa de decodificação.

Neste caso, o efeito Baldwin pode representar um papel importante no processo de obtenção da solução, pois é possível optar por condições iniciais ou então geradores de números pseudo-aleatórios que reduzam o custo das etapas de reparação e de busca local.

3 FORMULAÇÃO DO PROBLEMA

A principal contribuição deste trabalho está na solução do problema de representação e na produção de soluções factíveis para o caso específico de geração de tabelas completas de jogos em torneios de competição, baseados em disputas entre dois participantes adversários (Concilio & Von Zuben, 2000a; Concilio & Von Zuben, 2000b), tais como campeonatos esportivos, maratonas escolares, competições culturais, etc. As principais restrições a serem atendidas são:

- cada participante deve enfrentar todos os seus adversários uma única vez durante o torneio;
- todos os n participantes, a menos no caso de constituírem um número ímpar, devem jogar a cada rodada.

Portanto, o objetivo é elaborar uma tabela de jogos completa, formando um turno com $n-1$ rodadas, sendo que uma população de turnos será evoluída geração a geração pela aplicação de operadores genéticos devidamente adaptados ao contexto. O problema apresentado acima mostra-se compatível com o tipo de solução a ser proposta, uma vez que é fácil perceber a explosão combinatória existente. Em outras palavras, existirá uma grande quantidade de combinações para a formação dos turnos, sendo que este valor será função do número de participantes. Dessa maneira, o algoritmo a ser proposto realizará uma busca entre os possíveis candidatos à solução ótima (ou aproximadamente ótima).

A Tabela 1 mostra o número de combinações possíveis para os jogos, de modo a compor um turno completo. Nesse caso, supõe-se haver distinção de mando de jogo. Caso o torneio tenha turno e returno, a solução não muda, pois basta produzir o turno, já que o returno seria dado pela inversão simples de mando. O número de turnos completos factíveis para n participantes é dado pela seguinte expressão (Concilio, 2000):

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

Tabela 1: Número de combinações possíveis para a composição de um torneio completo factível.

Número de participantes	Número de combinações
2	2
4	384
6	$2{,}36\times10^{7}$
8	$9{,}7410\times10^{14}$
10	$4{,}6331\times10^{25}$
12	$3{,}8785\times10^{39}$
14	$8{,}1039\times10^{56}$
16	$5{,}6893\times10^{77}$
18	$1{,}7383\times10^{102}$
20	$2{,}9062\times10^{130}$

O problema de representação compacta foi resolvido a partir da proposição de uma codificação genética original, que utiliza um algoritmo de geração de todas as rodadas (atendendo às duas restrições citadas) apenas a partir da primeira rodada, conforme será detalhado na seção 4. Portanto, a codificação genética vai envolver apenas a descrição da primeira rodada. Dessa maneira, a Tabela 2 mostra os valores calculados para a definição do número de candidatos factíveis para essa representação genética compacta. O que importa na primeira rodada são os participantes que a comporão e qual a ordem dos competidores nos pares para a definição dos jogos, indicando quem está no seu domínio ou fora dele. Outro ponto a ser considerado é que não importa qual a seqüência de jogos dos pares, tanto na primeira rodada quanto nas demais. No exemplo a seguir para n = 6, existirão 6! (lê-se: seis fatorial) permutações possíveis, sendo 3! rodadas equivalentes para

cada tripla de jogos. Especificamente para o caso dos jogos envolverem os competidores 2 4, 1 6 e 5 3, todas as seguintes rodadas serão equivalentes:

2 4 1 6 5 3, 2 4 5 3 1 6, 1 6 2 4 5 3,
1 6 5 3 2 4, 5 3 2 4 1 6 e 5 3 1 6 2 4

Essas considerações levam à conclusão de que o número de diferentes primeiras rodadas factíveis vai ser dado por: $n!/(n/2)!$, sendo esta a cardinalidade do espaço de busca na representação compacta. O *fitness* será calculado pela

Tabela 2: Número de combinações possíveis para os jogos no caso da representação compacta.

Número de participantes	Número de combinações
2	2
4	12
6	120
8	1680
10	30240
12	665280
14	17297280
16	518918400
18	$1{,}7643\times10^{10}$
20	$6{,}7044\times10^{11}$

soma ponderada de dois termos. O primeiro termo da função (FC_1) contabiliza o número de participantes do torneio que realizam r jogos consecutivos dentro (ou fora) de seu domínio. O valor de r é função de n, sendo que nenhum participante pode realizar mais do que r jogos consecutivos dentro (ou fora) de seu domínio, pois existe uma restrição que impede esta ocorrência. O segundo termo da função-objetivo (FC_2) é uma medida que leva em conta a diferença entre as distâncias máxima e mínima percorridas pelos participantes do torneio, considerando todas as rodadas. É desejável que, para completar o torneio, todos os participantes tenham percorrido uma distância parecida, de modo a não privilegiar este ou aquele participante em termos de custo e tempo de deslocamento. Vale lembrar que cada jogo sempre ocorre no domínio de um dos dois participantes.

Dessa maneira, o problema de otimização apresentado pode ser definido como a seguir, sendo dados os valores de n e r, e supondo que x representa um possível conjunto de $n-1$ rodadas, representando um turno factível:

$$\max_{x} \; w_1 \cdot \frac{1}{FC_1(x)} + w_2 \cdot \frac{1}{FC_2(x)}$$

sujeito a:

- $w_1, w_2 > 0$ e $w_1 + w_2 = 1$;
- cada um dos participantes joga contra todos os outros;
- cada participante joga uma única vez em cada rodada;
- para cada participante, a diferença do número de jogos em seu domínio e fora dele é no máximo um;
- número máximo de jogos consecutivos no domínio do participante (ou fora) não é superior a r.

3.1 Exemplo ilustrativo

Considere o seguinte exemplo para cálculo de FC_1 e FC_2, adotando n = 8, r = 3 e $w_1 = w_2$ = 0,5. A matriz D contém as distâncias entre o domínio de cada participante em relação ao domínio dos demais.

$$D = \begin{bmatrix} 0 & 0 & 0 & 435 & 435 & 1123 & 1123 & 2783 \\ 0 & 0 & 0 & 435 & 435 & 1123 & 1123 & 2783 \\ 0 & 0 & 0 & 435 & 435 & 1123 & 1123 & 2783 \\ 435 & 435 & 435 & 0 & 0 & 1558 & 1558 & 2491 \\ 435 & 435 & 435 & 0 & 0 & 1558 & 1558 & 2491 \\ 1123 & 1123 & 1123 & 1558 & 1558 & 0 & 0 & 3906 \\ 1123 & 1123 & 1123 & 1558 & 1558 & 0 & 0 & 3906 \\ 2783 & 2783 & 2783 & 2491 & 2491 & 3906 & 3906 & 0 \end{bmatrix}$$

Rodada 1: 4 7 2 6 5 8 1 3
Rodada 2: 6 4 8 3 5 1 7 2
Rodada 3: 2 1 8 6 7 5 3 4
Rodada 4: 4 2 3 7 6 5 1 8
Rodada 5: 8 2 7 6 5 3 1 4
Rodada 6: 3 6 4 8 7 1 2 5
Rodada 7: 6 1 8 7 5 4 2 3

Figura 3: Exemplo de candidato factível à solução.

Tabela 3: Análise do candidato factível apresentado na Figura 3 em relação ao *fitness*.

Participante	Penalidade em relação a *r*	Distância percorrida
1	0	2681
2	0	4341
3	0	3218
4	0	2428
5	0	3551
6	0	6152
7	0	6587
8	0	7765
	Somatória= 0	Menor= 2428 Maior= 7765

O *fitness* é então obtido na forma:

$$\frac{1}{FC_1} = \frac{1}{\text{Somatória} + 1} = 1 \qquad \frac{1}{FC_2} = \frac{1}{\frac{\text{Maior}}{\text{Menor}}} = 0,313$$

$$\text{Fitness} = w_1 \frac{1}{FC_1} + w_2 \frac{1}{FC_2}$$

$$\text{Fitness} = 0,5 + 0,157 = 0,657$$

A penalidade adotada para r será sempre acrescida de uma unidade (no candidato factível à solução) para qualquer participante que jogar três vezes consecutivas dentro ou fora do seu domínio. No candidato à solução apresentado na Figura 3 esse fato não ocorre, portanto a penalidade em relação a r será zero (conforme mostra a Tabela 3).

4 ESTRATÉGIA DE SOLUÇÃO

A estratégia de solução permite a elaboração de um turno completo com qualquer número de participantes. Outra característica é o fato de que a quantidade efetiva de participantes deve ser sempre um número par. Obviamente, se o número de participantes for ímpar, define-se um participante fictício, de modo que seu adversário será o participante do torneio que irá folgar naquela rodada.

Para o cálculo do número de rodadas necessárias, utiliza-se a combinação de participantes dois a dois. Dessa maneira, aplica-se:

$$C_{n,p} = \frac{n!}{(n-2)!\ 2!}$$

Para exemplificar, tomando o número de participantes igual a 20, resulta:

$$C_{20,2} = \frac{20!}{(20-2)!\ 2!} = 190 \text{ jogos}$$

Como, por hipótese, todos os participantes jogam em todas as $n-1$ rodadas, cada uma delas terá $\frac{n}{2}$ jogos. Para o caso de 20 participantes, serão necessárias 19 rodadas com 10 jogos cada, totalizando 190 jogos.

4.1 Processo de Codificação Genética: uma Abordagem Compacta

A codificação genética utilizada para a composição de um cromossomo (primeira rodada), é uma combinação de números entre 1 e n. Cada um desses números (genes) representa um participante dentro da tabela gerada e são obtidos por meio de sorteio. Ressalta-se que essa formação cromossômica deve seguir criteriosamente os quesitos já apresentados.

Para a geração da segunda rodada em diante, escolhe-se uma semente do processo de geração aleatória, de modo que todos os outros pares de genes (jogos) sejam produzidos a partir dela. Percebe-se, portanto, que a cada primeira rodada gerada está atrelada uma semente, a partir da qual serão codificadas as outras rodadas. Assim, dada uma primeira rodada e uma semente, o algoritmo de geração do turno completo vai sempre produzir um mesmo resultado. É por isso que dizemos que a representação genética é compacta, pois a cada conjunto de primeira rodada e semente, existe sempre um turno completo que vai representar uma solução candidata factível.

Por exemplo, tomando n = 16, começa-se com uma primeira rodada arbitrária, tal como:

16 1 9 11 7 15 4 2 6 10 3 14 12 13 5 8

que dará origem ao seguinte código genético:

16	1	9	11	7	15	4	2	6	10	3	14	12	13	5	8

Esta lista de 16 genes, cada um com um valor distinto, representa um cromossomo, o qual será a entrada para o procedimento de expansão de código, responsável pela geração das demais 14 rodadas. Para tanto, o procedimento de expansão de código irá empregar um gerador de números pseudo-aleatórios que apresente a propriedade de repetitividade, e cada cromossomo contará com um gene adicional que representará a semente da geração (Figura 4). Para um mesmo cromossomo e uma mesma semente, sempre irá resultar uma mesma seqüência de rodadas. Após a elaboração da primeira rodada, a produção da seqüência de jogos que comporão as $n-2$ rodadas restantes será realizada com o auxílio de um algoritmo de reparação e outro de busca local. As rotinas de reparação serão responsáveis pela aceitação ou rejeição de cada um dos números aleatórios gerados, correspondente a cada gene (participante do torneio).

Figura 4: Representação genética compacta (genes + semente).

16	1	9	11	7	15	4	2	6	10	3	14	12	13	5	8	semente

Toma-se como exemplo n = 16 e supõe-se que a primeira rodada é a apresentada anteriormente, se o gerador de números pseudo-aleatórios produzir 11 como uma primeira saída, a conseqüência será uma segunda rodada com a seguinte configuração inicial:

11 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

Nesse caso, o único valor que não poderá ser aceito como

próxima saída do gerador será 9, pois na primeira rodada já existe o jogo entre os participantes 11 e 9, embora com o domínio invertido.

Com o aumento do número de jogos já definidos, a determinação de uma rodada válida pode se tornar mais custosa computacionalmente, pois aumenta a probabilidade de que a saída do gerador pseudo-aleatório não seja aceita. Este processo de montagem do turno completo a partir da primeira rodada, atendendo a múltiplas restrições, pode ser visto como um processo de busca em uma árvore de decisão, sendo necessário retornar a ramos anteriores dessa árvore (*backtracking*) caso não exista possibilidade de progresso a partir das condições existentes. Além dessas rotinas de reparação, mecanismos de busca local são aplicados imediatamente após a conclusão do processo de expansão de código, visando aumentar o *fitness* do indivíduo gerado.

4.2 Algoritmo de Expansão de Código para Produção de Soluções Factíveis

Conforme citado no item anterior, a partir do conjunto de primeira rodada e semente, será elaborado o restante das rodadas pelo emprego de um algoritmo de expansão de código. Salienta-se, novamente, que todas as rodadas devem sempre atender às restrições apresentadas para a sua formação. Se não forem observadas essas características, certamente será gerada uma solução infactível. Dessa maneira, lança-se mão de um algoritmo de expansão de código, basicamente composto dos seguintes passos:

- verificar se o jogo (par de participantes) sorteado ou o seu inverso, já não faz parte de alguma rodada do turno;
- se não fizer parte, validar esse par e proceder com o sorteio de outro jogo. Caso já esteja presente em alguma rodada, rejeitar o par e realizar outro sorteio para ocupar essa mesma posição.
- este processo se repete até que sejam atendidas as restrições. Caso se esgotem as possibilidades sem levar ao atendimento das restrições, todos os jogos dessa rodada serão invalidados e uma nova seqüência será sorteada;
- outra situação verificada pelo algoritmo de reparação é o número de vezes que uma mesma rodada é invalidada. Se esse valor exceder um determinado número, todas as rodadas (a menos da primeira) são rejeitadas e todo o processo se repete a partir da segunda rodada.

Como a árvore de decisão possui um número finito de nós, é certo que o algoritmo de reparação vai chegar a uma solução factível, embora não se tenha controle sobre o custo computacional específico associado a cada vez que o algoritmo é aplicado. A convergência ocorreu em todos os casos simulados, para várias instâncias de número total n de participantes.

O algoritmo de expansão de código representa uma etapa fundamental na geração de soluções factíveis a partir de uma representação compacta, sendo aplicado toda vez que a representação compacta sofrer qualquer tipo de modificação.

4.3 Algoritmo de Buscal Local

O processo de busca local é aplicado a todos os indivíduos, sejam eles gerados aleatoriamente para a composição da população ou após a aplicação dos operadores genéticos.

A busca local permite a geração de candidatos factíveis e a produção de ótimos locais. Em outras palavras, desde que não haja infactibilidade da solução e para a melhoria do valor do *fitness* de um dado conjunto de $n-1$ rodadas, será possível fazer uma inversão do domínio de cada uma das disputas envolvidas. Outra possibilidade existente é a reordenação das rodadas já definidas. No caso da representação compacta, todas as vezes em que o procedimento de expansão do código for aplicado em conjunto com o algoritmo de busca local, sua execução a partir de um mesmo código genético compacto levará sempre a um mesmo fenótipo.

O custo computacional para a geração da população aumenta proporcionalmente aos custos associados ao algoritmo de busca local. Entretanto, resultados muito superiores para o *fitness*, foram alcançados com essa nova metodologia (quando comparados aos valores obtidos sem sua aplicação), por requererem um número reduzido de gerações junto ao processo evolutivo.

4.4 Operação do Algoritmo Genético

Em linhas gerais, o programa calcula o *fitness* de cada indivíduo de uma população inicial gerada. Essa população é ordenada levando-se em conta o *fitness* obtido para, a seguir, receber a aplicação do algoritmo de *Roulette Wheel* e em seqüência os operadores genéticos. Novamente a população é ordenada para receber uma seleção elitista.

Após a escolha dos melhores cromossomos (codificação genética das soluções candidatas), o restante da população é complementada pela produção de novos cromossomos. Dessa maneira, a próxima geração está pronta para sofrer os passos já descritos. Ao término de todas as gerações será apresentada a solução que alcançar o melhor resultado de *fitness* segundo os critérios especificados. Os módulos que compõem o pacote de software que foi implementado são detalhados em Concilio (2000).

5 RESULTADOS E DISCUSSÃO

Para efeito de comparação com a solução fornecida por um especialista em um problema do mundo real, são apresentados a seguir os resultados obtidos pelo programa desenvolvido neste estudo e a tabela utilizada no Campeonato Paulista de Futebol de 1997, em sua Divisão A1. Utilizou-se a tabela deste ano, porque de 1998 em diante o modo de disputa foi alterado, não atendendo mais à restrição de que cada participante deveria jogar contra todos os demais adversários.

Nota-se que, ao comparar o *fitness* da Tabela 5, elaborada pelo programa, com o da Tabela 4, utilizada no Campeonato Paulista de 1997, resultados superiores foram obtidos a partir da aplicação da estratégia proposta neste trabalho. Vale ressaltar que não há conhecimento por parte dos autores deste trabalho acerca do procedimento adotado pelo especialista na produção da Tabela 4. Pode ter havido apenas a preocupação da geração de uma solução factível, seguida ou não de etapas de refinamento em busca do atendimento de objetivos, não necessariamente os mesmos adotados neste trabalho. Sendo assim, as duas 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, independente dos detalhes de implementação desta última.

Tabela 4: Solução fornecida pelo especialista (n=16). *Fitness* 1/FC$_1$= 0,333 *Fitness* 1/FC$_2$= 0,233 *Fitness* total= 0,283 (w$_1$=w$_2$=0,5)
Rodada 1⇒ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Rodada 2⇒ 15 14 4 5 2 8 1 7 6 3 10 11 16 13 12 9
Rodada 3⇒ 6 4 5 7 8 1 3 2 11 13 14 16 9 15 12 10
Rodada 4⇒ 4 8 2 5 1 6 7 3 10 14 15 11 16 12 13 9
Rodada 5⇒ 4 7 5 1 8 3 10 13 11 16 14 9 12 15 6 2
Rodada 6⇒ 15 10 2 4 5 8 3 1 6 7 11 14 12 13 9 16
Rodada 7⇒ 3 5 8 6 7 2 1 4 16 10 9 11 14 12 13 15
Rodada 8⇒ 7 14 2 10 3 12 5 13 4 9 6 15 8 16 1 11
Rodada 9⇒ 12 2 13 3 9 5 14 4 15 7 16 6 11 8 10 1
Rodada 10⇒ 2 13 3 9 5 14 7 16 6 11 8 10 1 12 4 15
Rodada 11⇒ 9 2 14 3 15 5 16 4 11 7 10 6 12 8 13 1
Rodada 12⇒ 7 10 2 14 3 15 5 16 4 11 6 12 8 13 1 9
Rodada 13⇒ 15 2 16 3 11 5 10 4 12 7 13 6 9 8 14 1
Rodada 14⇒ 2 16 3 11 5 10 7 13 6 9 8 14 1 15 4 12
Rodada 15⇒ 11 2 10 3 12 5 13 4 9 7 14 6 15 8 16 1

Existe uma sensível diferença em relação aos dois valores de penalidade para r calculados para as Tabelas 4 e 5. Para o caso da Tabela 5, $\frac{1}{FC_1} = 1,000$, indicando que nenhum participante do torneio jogou três ou mais vezes consecutivas dentro ou fora do seu domínio. Já para a Tabela 4, $\frac{1}{FC_1} = 0,333$, o que mostra que houve a participação de

Tabela 5: Solução fornecida pelo algoritmo genético, com representação compacta e busca local (n=16). Número de gerações=20, tamanho da população=40, taxa de crossover= 65%, taxa de mutação= 2%, tempo de execução=1h 08min. *Fitness* 1/FC$_1$= 1,000 *Fitness* 1/FC$_2$= 0,333. *Fitness* total= 0,666 (w$_1$=w$_2$=0,5)
Melhor tabela (solução factível)
Rodada 1⇒ 16 1 9 11 7 15 4 2 6 10 3 14 12 3 5 8
Rodada 2⇒ 15 13 10 2 5 9 11 7 6 1 4 16 8 14 3 12
Rodada 3⇒ 1 3 10 15 9 4 8 16 14 7 12 11 13 6 2 5
Rodada 4⇒ 6 4 7 8 12 9 11 3 13 14 16 10 15 5 1 2
Rodada 5⇒ 7 2 14 1 10 13 3 16 8 11 4 12 9 15 5 6
Rodada 6⇒ 15 8 3 7 5 13 11 16 9 1 4 10 2 14 6 12
Rodada 7⇒ 11 6 12 5 15 3 13 9 8 2 7 10 16 14 1 4
Rodada 8⇒ 10 9 13 4 7 6 1 12 16 5 8 3 14 11 2 15
Rodada 9⇒ 12 2 4 7 9 8 6 16 11 13 5 14 15 1 3 10
Rodada 10⇒ 7 13 14 9 2 3 4 11 10 12 16 15 5 1 6 8
Rodada 11⇒ 12 7 1 10 16 9 11 5 14 15 8 4 13 2 3 6
Rodada 12⇒ 1 8 4 14 9 3 10 11 15 12 2 6 7 5 13 16
Rodada 13⇒ 3 13 5 4 2 9 12 8 11 1 16 7 14 10 6 15
Rodada 14⇒ 13 1 5 3 11 2 8 10 7 9 16 12 14 6 4 15
Rodada 15⇒ 13 8 3 4 12 14 10 5 9 6 15 11 1 7 2 16

alguns competidores em pelo menos três rodadas consecutivas dentro ou fora do seu domínio. Em uma análise mais cuidadosa, pode-se verificar que o participante 3 jogou três vezes consecutivas no seu domínio nas rodadas seis, sete e oito. O participante 10, também nessas mesmas rodadas, fez três disputas consecutivas fora do seu domínio. Essa situação implica que o somatório em relação à r será dois, portanto $\frac{1}{FC_1} = \frac{1}{2+1} = 0,333$. Caso se procure eliminar diretamente esta seqüência de três jogos consecutivos para os participantes 3 e 10, no domínio e fora dele respectivamente, pela simples troca de mando de jogo, por exemplo, na rodada oito (substituição do jogo 2 10 por 10 2 e do jogo 3 12 por 12 3) resolve-se o problema imediato para os participantes 3 e 10, mas cria-se outros problemas, como:

- a violação da restrição de que cada participante deva ter no máximo a diferença de um entre jogos dentro e fora do domínio;
- com estas trocas, o participante 2 passou a ter três jogos consecutivos fora do seu domínio.

Conclui-se, portanto, que a diferença de desempenho das duas soluções comparadas não pode ser eliminada de forma elementar, fato que conduz à conclusão de que há uma grande diferença qualitativa entre as duas soluções sob comparação.

Analisando-se $1/FC_2$, verifica-se que quanto maior for esta razão, a diferença entre a menor distância percorrida e a

maior será um valor menor. Isso implica em dizer que o torneio está mais equilibrado em relação à distância percorrida pelos participantes.

Para o caso da Tabela 5, a Figura 5 mostra a situação do melhor *fitness* e do *fitness* médio ao longo das gerações. Nota-se na figura apresentada que já para a primeira geração o melhor *fitness* é maior que o obtido pelo especialista (Tabela 4), entretanto este valor ainda será evoluído ao longo das gerações. Este melhor *fitness* obtido já a partir da primeira rodada deve-se à adoção de mecanismos de busca local, fazendo com que cada indivíduo que compõe a primeira geração já represente um ótimo local. A distância existente entre o *fitness* médio e aquele produzido pelo melhor indivíduo demonstra a existência de um nível significativo de diversidade na população ao longo das gerações.

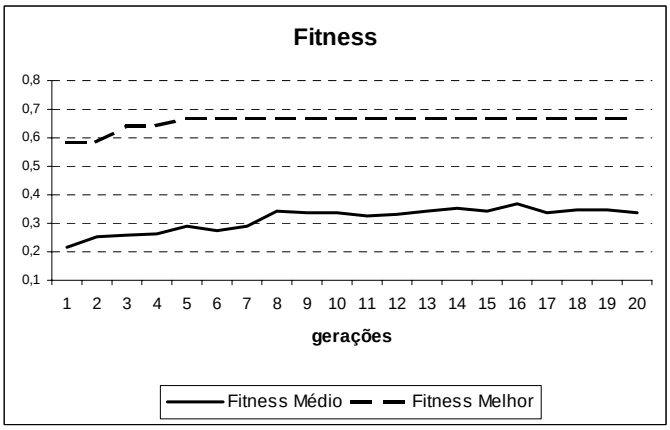

Figura 5: Evolução dos *fitness* da média da população e do melhor indivíduo a cada geração, resultando na solução apresentada na Tabela 5.

5.1 Comparação entre codificação compacta e expandida

A busca de solução para problemas *COP* também pode ser implementada empregando codificação expandida, a qual se dá pelos mecanismos usuais de obtenção de indivíduos factíveis via técnicas de otimização baseada em restrições e pela aplicação de operadores evolutivos diretamente sobre o código expandido, seguida da aplicação de procedimentos de reparação e busca local.

São etapas idênticas ao caso anterior, envolvendo expansão de código:

- a técnica de otimização baseada em restrições. No entanto, aqui esta técnica é aplicada uma única vez para cada indivíduo, não necessitando ser reaplicada toda vez que o código genético é alterado, como feito no caso anterior.
- os procedimentos de busca local.

Por outro lado, as diferenças são as seguintes:

- os operadores genéticos aqui só se aplicam ao código expandido, podendo gerar indivíduos infactíveis, enquanto que no caso anterior estavam restritos ao código compacto, gerando sempre indivíduos factíveis. Sendo assim, o genótipo aqui equivale ao fenótipo do caso anterior.
- os procedimentos de reparação aqui só se aplicam sob o código expandido, enquanto no caso anterior eles entram em ação durante a expansão de código.

A Tabela 6 mostra a média de cinco resultados obtidos em relação ao *fitness* para a codificação genética expandida quando comparada com a compacta. Ressaltando que o termo codificação genética expandida faz referência ao cromossomo que possui no seu genoma todos os genes representativos do torneio. Veja o seguinte exemplo, considerando n=10:

- para a versão compacta, o cromossomo (código genético) terá um tamanho de 10 genes e o seu fenótipo será composto por 90 genes, ou seja, $n \times (n-1)$;
- para a versão expandida, o cromossomo será composto por 90 genes, sendo que toda esta carga genética será evoluída geração a geração.

Tabela 6: Testes comparativos envolvendo a codificação genética compacta e expandida com número de gerações igual a 20 e tamanho da população de 30.

Número de Participantes	**Codificação Expandida**	**Codificação Compacta**
	***Fitness* médio alcançado**	***Fitness* médio alcançado**
8	0,598	0,654
10	0,615	0,637
12	0,599	0,607
14	0,612	0,629
16	0,611	0,615
18	0,578	0,600

Analisando-se a Tabela 6, verifica-se que o uso de codificação expandida produz técnicas de busca menos eficientes, quando comparadas à técnica de busca baseada em

codificação compacta e expansão de código. A razão para tal é simplesmente o fato de não haver uma evolução de condições iniciais para o algoritmo de otimização baseado em restrições, e pelo fato do processo evolutivo estar operando em um espaço com elementos factíveis e infactíveis, o que impede a aceitação de boa parte das operações genéticas propostas, reduzindo em muito a taxa média de progresso junto ao processo evolutivo.

A Tabela 7 mostra o tempo de execução do programa computacional desenvolvido para os mesmos casos da Tabela 6. Os tempos medidos foram obtidos para o processamento em um microcomputador *Pentium* III 650MHz com 128Mbytes de memória *RAM*. O programa computacional foi desenvolvido no ambiente de programação *Delphi*. Nota-se que o tempo de execução para a codificação genética compacta é maior que o da expandida em todos os testes realizados. Justifica-se esse fato pela presença do algoritmo de expansão de código, que faz a geração das demais rodadas do torneio tomando como ponto de partida a primeira rodada e mais uma semente.

Tabela 7: Testes comparativos em relação ao tempo de execução do programa com número de gerações igual a 20 e tamanho da população de 30.

Número de participantes	**8**	**10**	**12**	**14**	**16**	**18**
Codificação Compacta	14s	59s	6min 45s	20min 55s	29min 15s	1h 50min 20s
Codificação Expandida	4s	22s	2min 11s	9min 20s	15min 22s	46min 40s

6 COMPARAÇÃO COM OUTRAS META-HEURÍSTICAS

Problemas de escalonamento, assim como outras formulações envolvendo otimização combinatória com restrições, já vêm sendo tratados na literatura a partir de metodologias alternativas (Voss, 2001; Grötschel & Lovász, 1993) àquelas empregadas neste trabalho. Como exemplo, podemos citar o uso de meta-heurísticas como busca tabu (Widmer, 1991; Porto & Ribeiro, 1995; Taillard *et al*., 1997), GRASP (*greedy randomized adaptive search*) (Kontoravdis & Bard, 1995; Arguello *et al*., 1997; Binato *et al*., 2000) e *simulated annealing* (Johnson *et al*., 1989).

Além disso, é de amplo conhecimento que busca tabu (BT) (Glover & Laguna, 1997), GRASP (Resende, 1998) e *simulated annealing* (SA) (Kirkpatrick *et al*., 1983), principalmente em suas versões mais elaboradas (BT: Dammeyer & Voss, 1993; GRASP: Hansen & Mladenovic, 1999; SA: Ingber, 1996), apresentam desempenho superior àquele produzido pelo emprego de algoritmos genéticos sem busca local e outros refinamentos. Por exemplo, no tratamento do problema da mochila (*knapsack problem*), Battiti & Tecchiolli (1995) obtiveram melhores resultados com busca tabu, quando comparada com algoritmos genéticos.

No entanto, versões mais elaboradas de algoritmos genéticos, seguindo as mesmas linhas de atuação adotadas no presente trabalho, vêm apresentando resultados superiores às demais meta-heurísticas, como em Burke *et al*. (1996), em Thangiah *et al*. (1994) e em Merz e Freisleben (1999).

Mesmo quando os resultados são equivalentes, as abordagens populacionais, como as versões mais elaboradas de algoritmos meméticos, as quais também incluem a abordagem adotada no presente trabalho, apresentam a vantagem de fornecer múltiplas soluções de boa qualidade (Colorni *et al.*, 1998).

Frente à multiplicidade de problemas com uma grande variedade de requisitos, a natureza do espaço de busca resultante em cada caso impõe dificuldades a certos algoritmos de solução e não a outros, e a situação pode se inverter em outros casos. Moscato (2001) procurou investigar aspectos característicos e genéricos dos problemas, e de seus correspondentes espaços de busca, que influenciam no desempenho de uma variada classe de meta-heurísticas, mas para se chegar a algo como uma teoria de complexidade computacional para meta-heurísticas é ainda necessário alcançar avanços na análise de desempenho e na formalização das abordagens de solução.

Logo, restrito a um problema de otimização combinatória específico, nenhum resultado conclusivo e genérico pode ser obtido para uma dada meta-heurística, quando comparada com abordagens alternativas de solução. No entanto, resultados indicativos do potencial desta meta-heurística podem ser derivados desta comparação.

Esta é a razão para que resultados comparativos façam parte das perspectivas futuras deste trabalho. Eles só não foram incluídos no presente trabalho pelo fato de que todas as propostas alternativas presentes na literatura não são imediatamente implementáveis ou mesmo diretamente comparáveis, ou por não serem completamente reprodutíveis a partir dos trabalhos publicados ou por incluírem peculiaridades de implementação ausentes na presente formulação e decisivas na implementação de um processo comparativo. Esforços têm sido realizados no sentido de viabilizar estas comparações, incluindo contato pessoal com os autores. Os principais trabalhos selecionados da literatura são os seguintes: Hamiez & Hao (2001), Schaerf (1999), Nemhauser & Trick (1998), Terril & Willis (1994) e Wright (1994). Não se trata de uma lista exaustiva de contribuições na área, mas já indicam que o problema de geração automática de turnos completos em torneios e suas generalizações e

particularizações correspondem a um "campo de prova" importante para todos aqueles ocupados em contribuir junto à área de otimização combinatória com restrições.

7 COMENTÁRIOS FINAIS

A aplicação conjunta de técnicas de computação evolutiva, busca local e otimização baseada em restrições tem por objetivo produzir uma solução de compromisso para problemas de otimização caracterizados por apresentarem um grande número de variáveis, múltiplos objetivos e múltiplas restrições.

Como principais contribuições deste trabalho, é possível mencionar:

- a própria iniciativa de aplicação conjunta de técnicas de computação evolutiva, busca local e otimização baseada em restrições, com a distribuição adequada de "papéis" entre cada abordagem, procurando explorar a natureza complementar de cada uma ao longo da busca pela solução ótima;
- a formalização da proposta em termos de conceitos bem definidos no campo da computação evolutiva, como decodificadores, algoritmos de reparação e algoritmos de busca local;
- a elaboração e a implementação computacional de um algoritmo de expansão de código para o problema de geração de turnos completos em torneios, o qual pode ser visto como uma caminhada em uma árvore de decisão;
- comparação de desempenho entre codificação expandida e codificação compacta para o problema de geração de turnos completos em torneios.

Os resultados obtidos ao aplicar a técnica proposta permitem concluir que é possível se beneficiar com o uso de uma representação compacta em conjunto com um algoritmo de expansão de código, quando comparado com o emprego direto de codificação expandida. As razões levantadas para explicar o sucesso desta abordagem são:

- processo evolutivo somente tem lugar no espaço compacto, onde não há infactibilidade ou onde a factibilidade pode ser facilmente conquistada. Com isso, a existência de restrições praticamente não afeta a taxa de evolução no espaço compacto.
- algoritmo de expansão de código, responsável pelo atendimento das restrições, é um método de otimização baseado em restrições, de modo que a representação expandida resultante será sempre factível;
- as duas possíveis limitações do algoritmo de expansão de código, ou seja, a impossibilidade de garantir que qualquer representação expandida factível tenha uma representação compacta associada e a existência de ótimos locais como resultado do processo de expansão, são amenizadas respectivamente pela presença de busca local e pela evolução de uma população de representações compactas.

A abordagem proposta, como qualquer outra estratégia de busca implementada em computador, também apresenta suas desvantagens:

- necessidade de definição de uma codificação compacta para cada problema de aplicação, mais especificamente de um espaço de busca compacto, sendo que sua dimensão mostrou-se decisiva na eficiência do processo evolutivo;
- necessidade de implementação de um algoritmo de expansão de código para cada problema de aplicação, levando em conta todas as particularidades do problema;
- custo computacional vinculado ao processo de busca.

Além de superar a abordagem baseada no código expandido, a comparação de desempenho junto a soluções propostas por especialistas, adotadas para problemas do mundo real, permite concluir que a adoção de uma codificação compacta, seguida da expansão de código com reparação e busca local, é eficaz na procura de soluções de alta qualidade, superando de forma evidente soluções que vêm sendo adotadas na prática. Infelizmente, por motivos já enunciados na seção 6, comparações com abordagens alternativas já propostas na literatura ainda não foram concluídas.

Como perspectivas futuras para este trabalho são sugeridos os seguintes tópicos:

- verificar a possibilidade de incorporação de restrições adicionais e estudar o desempenho do algoritmo quando se varia a dificuldade no atendimento das restrições;
- verificar a possibilidade de incorporação de penalidades na função de *fitness* para o caso do processo de expansão, juntamente com a busca local e os algoritmos de reparação, requerer recursos computacionais excessivos (por exemplo, número elevado de ocorrências de *backtracking*);
- permitir que um mesmo processo de expansão de código possa operar com diferentes geradores de números pseudo-aleatórios, todos apresentando a propriedade de repetitividade;

- no caso da aplicação da estratégia de solução proposta neste trabalho a problemas que admitem o emprego de mais de um algoritmo de otimização baseada em restrições, criar condições para que o algoritmo de expansão de código possa optar por um ou outro dentre os algoritmos disponíveis para expansão de código;

- extensão da estratégia de solução proposta a outros problemas de escalonamento e possivelmente a outros problemas de otimização com restrições;

- comparação criteriosa de desempenho com abordagens alternativas já propostas na literatura.

A APÊNDICE - GERADOR DE NÚMEROS PSEUDO-ALEATÓRIOS COM DISTRIBUIÇÃO UNIFORME E REPETITIVIDADE

Conforme definido por L'Ecuyer (1994), computadores digitais só podem gerar números pseudo-aleatórios, por se tratarem de máquinas totalmente determinísticas. No entanto, desde que um gerador de números pseudo-aleatórios seja aprovado em testes de aleatoriedade, sua aplicação vai levar a comportamentos equivalentes aos produzidos por geradores puramente aleatórios.

Um gerador de números pseudo-aleatórios em computadores digitais tem um estado que evolui em um espaço discreto S. Esse espaço é composto por um número finito de estados e a repetitividade é garantida a partir de uma recorrência na forma: $s_n = f(s_{n-1}),\ n \geq 1$, sendo que $s_0 \in S$ será denominado a semente e $f : S \rightarrow S$ será a função determinística de transição. A cada passo n, a função de saída do gerador será dada por $u_n = g(s_n)$ com $g : S \rightarrow [0, 1]$ (essa saída poderia ser mais geral, entretanto está sendo considerado o intervalo [0,1]). Observe que a seqüência de saída do gerador será um conjunto de valores representado por $\{u_n, n \geq 0\}$. Como o espaço S é finito, a seqüência $\{u_n, n \geq 0\}$ deverá ser periódica (possivelmente após um transitório inicial). Em outras palavras, todas as vezes em que a semente s_0 for a mesma, a seqüência aleatória gerada será repetida. Em situações em que é necessário aumentar a periodicidade do gerador, ou seja quando a quantidade de números aleatórios a serem gerados é muito grande, será desejável fazer com que esse período seja o mais próximo possível da cardinalidade do espaço S.

Geralmente, os modelos fornecidos pelos sistemas computacionais são os geradores lineares apresentando uma relação de recorrência $I_{j+1} = (aI_j + c) \bmod m,\ j = 1, 2, ...$, responsável pela geração de uma seqüência I_1, I_2, I_3, ... de inteiros entre 0 e $m-1$, sendo m o módulo, a e c inteiros positivos denominados multiplicador e incremento. A recorrência vai certamente produzir, para algum $j = p \leq m$, $I_j = I_k\ (k < j)$, ou seja, ela terá um período $p \leq m$. Se o período for $p = m$, todo inteiro entre 0 e $m-1$ vai ocorrer em alguma das próximas $m-1$ iterações, fazendo com que a escolha do valor inicial I_0 da recorrência (semente da geração pseudo-aleatória) não influa de forma significativa no resultado estatístico associado a seqüências longas. Os geradores lineares têm a vantagem de serem rápidos, de simples implementação e repetitivos para uma mesma máquina.

REFERÊNCIAS

Arguello, M.F., Bard, J.F. & Yu, G. "A GRASP for Aircraft Routing in Response to Groundings and Delays", Journal on Combinatorial Optimization, 5: 211-228, 1997.

Bäck, T., Fogel, D.B. & Michalewicz, Z. (eds.) "Handbook of Evolutionary Computation", Oxford University Press, 1997.

Battiti, R. & Tecchiolli, G. "Local search with memory: Benchmarking rts", Operations Research Spektrum, 17(2/3): 67-86, 1995.

Binato, S., Hery, W.J., Loewenstern, D.M. & Resende, M.G.C. "A GRASP for Job Shop Scheduling", AT&T Labs Research Technical Report: 00.6.1, 2000.

Burke, E.K., Newall, J.P. & Weare, R.F. "A Memetic Algorithm for University Exam Timetabling", *in* Burke, E.K. & Ross, P. The Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science, Springer, 1153: 241-250, 1996.

Coello, C. A. C. "An Updated Survey of Evolutionary Multiobjective Optimization Techniques: State of the Art and Future Trends", Congress on Evolutionary Computation, vol. 1, pp. 3-13, 1999.

Colorni A., Dorigo, M. & Maniezzo, V. "Metaheuristics for High-School Timetabling", Computational Optimization and Applications, 9(3): 277-298, 1998.

Concilio, R. "Contribuições à Solução de Problemas de Escalonamento pela Aplicação Conjunta de Computação Evolutiva e Otimização com Restrições", Dissertação de Mestrado, Faculdade de Engenharia Elétrica e de Computação, Unicamp, 2000.

Concilio, R. & Von Zuben, F. J. "Geração Automática de Tabela de Jogos em Torneios: Uma Abordagem Evolutiva com Busca Local e Representação Genética Compacta", Anais do XIII Congresso Brasileiro de Automática (CBA 2000), pp. 1247-1252, Florianópolis 2000(a).

Concilio, R. & Von Zuben, F. J. "Evolutionary Design of Schedules in Championships with Compact Genetic Codification and Local Search", Workshop on Memetic Algorithms (WOMA) at the 2000 Genetic and Evolutionary Computation Conference (GECCO-2000), pp. 109-113, Las Vegas, Nevada, USA, July, 2000(b).

Costa, M. F. N. "Computação Evolutiva para a Minimização de Perdas Resistivas em Sistemas de Distribuição de Energia Elétrica", Dissertação de Mestrado, Faculdade de Engenharia Elétrica e de Computação, Unicamp, 1999.

Dammeyer, F. & Voss, S. "Dynamic Tabu List Management using the Reverse Elimination Method", Annals of Operations Research, 41:31-46, 1993.

Eiben, A. E. & Ruttkay, Z. "Constraint-Handling Techniques", *in* Bäck, T., Fogel, D.B. & Michalewicz, Z. (eds.) Handbook of Evolutionary Computation, Oxford University Press, 1997.

Glover, F. & Laguna, M. "Tabu Search", Kluwer, 1997.

Goldberg, D. E. "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison-Wesley, 1989.

Grötschel, M. & Lovász, L. "Combinatorial Optimization: A Survey", DIMACS Technical Report 93-29, Princeton University, 1993.

Hamiez, J.-P. & Hao, J.-K. "Solving the Sports League Scheduling Problem with Tabu Search", *in* Nareyek, A. (ed.) Local Search for Planing and Scheduling, Lecture Notes on Artificial Intelligence, Springer, 2148: 24-36, 2001.

Hansen, P. & Mladenovic, N. "An introduction to variable neighborhood search", *in* Voss, S., Martello, S., Osman, I.H. & Roucairol, C. (eds.) Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, pp. 433-458, Kluwer, 1999.

Ingber, L. "Adaptive Simulated Annealing (ASA): Lessons learned", Control and Cybernetics, 25: 33-54, 1996.

Johnson, D.S., Aragon, C.R., McGeoch, L.A. & Schevon, C. "Optimization by simulated annealing: An experimental evaluation; Part I: Graph Partitioning", Operations Research, 37: 865-892, 1989.

Kirkpatrick, S., Gelatt Jr., C.D. & Vecchi, M.P. "Optimization by simulated annealing", Science, 220: 671-680, 1983.

Kontoravdis, G. & Bard, J.F. "A GRASP for the Vehicle Routing Problem with Time Windows", ORSA Journal on Computing, 7(1): 10-23, 1995.

L'Ecuyer, P. "Testing random number generation", Annals of Operations Research, 53: 77-120, 1994.

Merz, P. & Freisleben, B. "A Comparison of Memetic Algorithms, Tabu Search, and Ant Colonies for the Quadratic Assignment Problem", Proceedings of the 1999 International Congress of Evolutionary Computation (CEC'99), IEEE Press, pp. 2063-2070, 1999.

Michalewicz, Z. "Constraint-Handling Techniques", *in* Bäck, T., Fogel, D.B. & Michalewicz, Z. (eds.) Handbook of Evolutionary Computation, Oxford University Press, 1997.

Moscato, P.A. "On Evolution, Search, Optimizaton, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms", Tech. Rep. Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989.

Moscato, P.A. "Problemas de Otimização NP, Aproximabilidade e Computação Evolutiva: Da Prática à Teoria", Tese de Doutorado, Faculdade de Engenharia Elétrica e de Computação, Unicamp, 2001.

Nemhauser, G.L. & Trick, M.A. "Scheduling a Major College Basketball Conference", Operations Research, 46(1): 1-8, 1998.

Papadimitriou, C.H. & Steiglitz, K. "Combinatorial Optimization: Algorithms and Complexity", Prentice-Hall, 1982.

Porto, S.C.S. & Ribeiro, C.C. "Parallel Tabu Search Message-Passing Synchronous Strategies for Task Scheduling under Precedence Constraints", Journal of Heuristics, vol. 1, 1995.

Resende, M.G.C. "Greedy Randomized Adaptive Search Procedures (GRASP)", AT&T Labs Research Technical Report: 98.41.1, 1998.

Schaerf, A. "Scheduling Sport Tournaments Using Constraint Logic Programming", Constraints, 4(1): 43-65, 1999.

Sherali, H. D. Bazaraa, M. S. & Shetty, C. M. "Nonlinear Programming: Theory and Algorithms", 2nd. edition, John Wiley & Sons, 1993.

Smith, A. E. & Coit, D. W. "Constraint-Handling Techniques", *in* Bäck, T., Fogel, D.B. & Michalewicz, Z. (eds.) Handbook of Evolutionary Computation, Oxford University Press, 1997.

Surry, P. D., Radcliffe, N. J. & Boyd, I. D. "A multi-objective approach to constrained optimization of gas supply networks", *in* Fogarty, T. (ed.) AISB-95 Workshop on Evolutionary Computing, Springer, pp. 166-180, 1995.

Taillard, E.D., Badeau, P., Gendreau, M., Guertin, F. & Potvin, J.Y. "A tabu search heuristic for the vehicle routing problem with soft time windows", Transportation Science, 31: 170-186, 1997.

Terril, B.J. & Willis, R.J. "Scheduling the Australian State Cricket Season Using Simulated Annealing", Journal of the Operational Research Society, 45(3): 276-280, 1994.

Thangiah, S.R., Osman, I.H. & Sun, T. "Hybrid Genetic Algorithms, Simulated Annealing and Tabu Search Methods for Vehicle Routing Problems with Time Windows", Technical Report UKC/OR94/4, Institute of Mathematics and Statistics, University of Kent, Canterbury, UK, 1994.

Voss, S. "Meta-heuristics: The State of the Art", *in* Nareyek, A. (ed.) Local Search for Planning and Scheduling, Springer, pp. 1-23, 2001.

Whitley, D., Gordon, V. S. & Mathiask K. "Lamarckian Evolution, the Baldwin Effect and Function Optimization", *in* Davidor, Y., Schwefel, H.-P. & Männer, R. (eds.) Proceedings of the Third Conference on Parallel Problem Solving from Nature, Berlin: Springer, pp. 6-15, October, 1994.

Widmer, A.M. "The Job-shop Scheduling with Tooling Constraints: A Tabu Search Approach", Journal of the Operational Research Society, 42: 75-82, 1991.

Wright, M. "Timetabling County Cricket Fixtures Using a Form of Tabu Search", Journal of the Operational Research Society, 45(7): 758-770, 1994.