META-HEURÍSTICAS PARA O PROBLEMA DE PARTIÇÃO DE NÚMEROS

Silvio Alexandre de Araujo
silvio@din.uem.br

Ademir Aparecido Constantino
ademir@din.uem.br

Candido Ferreira Xavier de Mendonça Neto
xavier@din.uem.br

Departamento de Informática

Universidade Estadual de Maringá

87020-090-Maringá-PR, Brasil

Resumo

Neste trabalho, são implementados vários procedimentos meta-heurísticos para resolução do Problema de Partição de Números, que consiste num dos problemas clássicos de otimização combinatória cujo problema de decisão associado pertence à classe NP-difícil. Além de ter diversas aplicações práticas, este problema é freqüentemente usado como base para demonstrar a NP-dificuldade de outros problemas. Basicamente, os seguintes procedimentos foram implementados: heurísticas de Busca Local Simples e as meta-heurísticas Simulated Annealing, Busca Tabu, Algoritmos Genéticos e Algoritmos Meméticos. O objetivo deste trabalho consiste em fazer uma análise comparativa desses métodos, bem como, investigar alguns aspectos relativos à complexidade do problema de partição de números.

Palavras-chave: problema de partição de números, meta-heurísticas.

Abstract

In this work, we use several metaheuristics procedures to solve the Number Partitioning Problem, that is a classical combinatorial optimisation problem whose associated decision problem is known to be NP-hard. There are several practical applications for this problem and also very often it is used to show NP-hardness of other problems. We report basically the following approaches: Simple Local Search heuristics and the metaheuristics, Simulated Annealing, Tabu Search, Genetic Algorithm and Memetic Algorithm. The main goal of this work is to compare these approaches and to investigate some complexity aspect of the number partitioning problem.

Key words: number partitioning problem, metaheuristics.

1. Introdução

A motivação para este trabalho surgiu após a revisão de alguns estudos que retratam a dificuldade de se aplicar meta-heurísticas para a resolução do problema de partição de números (Johnson *et al.* (1991), Arguello, *et al.* (1996), Diaz *et al.*(1996), Berretta e Moscato, (1999)). Além disso, vários autores utilizam o problema de partição de números para estudar um fenômeno chamado "fase de transição" (Gent e Walsh (1998), Berretta, *et al.*(2003), Mertens, (2003), Stadler *et al.* (2003)) que consiste basicamente na transição de uma "região", onde a maioria dos exemplos de um problema de otimização combinatória tem muitas soluções, para uma outra "região", onde a maioria dos problemas não tem nenhuma solução. Este fenômeno pode aparecer em diversos problemas de otimização combinatória e, na Seção 9 deste trabalho, tem-se alguns comentários adicionais.

O **problema de partição de números** consiste em: dado um conjunto de N números, o objetivo é subdividi-lo em dois subconjuntos (chamados de partições) de tal forma que, a diferença entre os valores das somas dos números dessas duas partições seja a menor possível. Por exemplo, considere o seguinte conjunto com quatro números (23, 20, 56, 48). As partições (20,56) e (23,48) consistem no particionamento ótimo para este conjunto e, seu valor é 5. Apesar da simplicidade do enunciado, este é um problema de otimização combinatória que pertence a classe **NP-difícil**. Observe que, para um conjunto com N números têm-se 2^N possíveis maneiras de subdividi-lo em duas partições.

Na **representação** utilizada neste trabalho, cada partição é representada por um vetor binário, onde o número 0 representa uma partição e o número 1 representa a outra. Assim, a representação da partição acima fica (1,0,0,1). Cabe observar que existem outras formas de representações, por exemplo, representação ternária ou representação utilizando números inteiros.

Para a obtenção dos resultados computacionais utiliza-se um conjunto de 25 **exemplos** (baseados nos exemplos gerados em Berretta e Moscato, (1999)) com tamanhos de 15, 35, 55, 75 e 95 números (cinco exemplos para cada tamanho), sendo que, em todos os exemplos, cada número possui 10 dígitos. A **análise dos resultados** é feita com base na média aritmética dos cinco exemplos e as principais **medidas de desempenho** analisadas são: qualidade da solução, tempo e robustez. Cabe observar que, para este conjunto de exemplos são conhecidos os resultados ótimos, bem como os resultados obtidos por uma boa heurística de construção.

O **recurso computacional** utilizado neste trabalho consiste num PC Compaq MMX Pentium II 300, 128 Mb de RAM, sistema operacional Windows NT, linguagem C e compilador Borland C++ 5.0.

Na seção 2 deste trabalho, descreve-se algumas heurísticas de construção e, na seção 3, algumas heurísticas de busca são apresentadas. Nas seções 4, 5, 6 e 7 são descritas, de forma resumida, as várias meta-heurísticas utilizadas bem como alguns resultados computacionais obtidos durante o ajuste de parâmetros. Finalmente na seção 8 têm-se os resultados computacionais e na seção 9 os comentários finais e propostas de trabalhos futuros.

2. Heurísticas de Construção (HC)

Este tipo de heurística consiste em, passo a passo, construir uma solução para um determinado problema. As soluções obtidas pelas heurísticas de construção serão utilizadas como soluções iniciais para as heurísticas de busca, descritas na Seção 3.

Nos experimentos computacionais foram utilizadas três heurísticas construtivas, descritas a seguir:

- **HC1:** Consiste em construir uma solução aleatoriamente, ou seja, dado um determinado número, escolhe-se aleatoriamente (com uma determinada probabilidade) qual o conjunto em que este número deverá ficar.
- **HC2:** Inicialmente ordena-se os números em ordem decrescente. Os números ordenados são separados um para cada conjunto de maneira sucessiva, ou seja, se o primeiro número for para um determinado conjunto, o segundo deverá ir para o outro e assim sucessivamente.
- **HC3:** Segue a mesma idéia da **HC2**, entretanto na **HC2** ao fazer a ordenação, muda-se também a ordem dos números no arquivo de entrada de dados e isto faz com que os índices relativos a cada valor fiquem em ordem decrescente, o que estava influenciando em alguns resultados posteriores, por exemplo, no algoritmo genético. Ao fazer o cruzamento de "1-point" a influência do pai que passava a parte esquerda do vetor para o filho era bem maior do que a do outro pai. Diante disso,

fez-se na **HC3** uma ordenação por índices sem alterar a ordem do arquivo de entrada de dados. Assim, o valor da solução obtida pelas heurísticas **HC2** e **HC3**, são sempre iguais, entretanto a solução em si, geralmente é diferente.

Na **HC1**, as soluções obtidas são pouco robustas pois são obtidas de maneira aleatória. Entretanto, esta heurística é importante pois, através dela, pode-se testar o desempenho das meta-heurísticas partindo-se de uma solução qualquer. Além disso, a construção de soluções aleatórias é necessária em alguns casos. Por exemplo, para os algoritmos genéticos e meméticos descritos na Seção 7 pois permitem a obtenção de soluções com bastante diversidade. Na **HC2** e **HC3**, obtém-se uma solução relativamente boa. Com isso, pretende-se aplicar as meta-heurísticas a partir de uma boa solução inicial e analisar a influência da solução inicial no desempenho das mesmas.

3. Heurísticas de Busca (HB)

As Heurísticas de busca consistem basicamente em: a partir de uma solução inicial obtida, por exemplo, através de uma heurística de construção, realizar movimentos obtendo-se outras soluções.

Foram feitos alguns testes iniciais com várias heurísticas baseadas nos movimentos de **transferência** e **troca,** tais como: **transferir** um ou mais elementos de um conjunto para outro e **trocar** um ou mais elementos entre os conjuntos. Concluiu-se que as heurísticas que utilizavam somente transferência entre conjuntos davam resultados piores que as que utilizavam somente troca ou uma mistura de transferência e troca. Além disso, observou-se que quando utilizou-se mais que um elemento para troca ou transferência, os resultados também eram bons, entretanto, intuiu-se que isto poderia dificultar o ajuste dos parâmetros.

Diante disso, quatro diferentes heurísticas de busca foram utilizadas::

- **HB1:** Nesta heurística, cada movimento consiste em **trocar** dois elementos, um de cada conjunto. Estes elementos são escolhidos **aleatoriamente** nos dois conjuntos.
- **HB2:** São considerados dois tipos de movimento. Escolhe-se **aleatoriamente** um elemento do conjunto maior (conjunto no qual a soma dos números é maior). Caso a diferença entre os dois conjuntos seja maior do que este elemento, o movimento caracteriza-se por **transferir** este elemento para o conjunto menor e, caso a diferença seja menor do que o elemento, o movimento consiste em **trocar** este elemento do conjunto maior com um elemento escolhido **aleatoriamente** do conjunto menor.
- **HB3:** Consiste em **trocar** dois elementos, um de cada conjunto. Escolhe-se o elemento do conjunto maior **aleatoriamente** e escolhe-se o **menor** elemento do conjunto menor. Para tanto, é feita uma variação nos índices, começando do índice de menor valor, sendo escolhido para troca o primeiro que pertença ao conjunto menor .
- **HB4:** Esta heurística é semelhante a **HB2**. A única diferença é que, no movimento de **troca** de dois elementos, a escolha no conjunto menor não é aleatória, ou seja, escolhe-se o **menor** elemento.

A idéia básica das duas primeiras heurísticas de busca é realizar movimentos aleatórios. Com isso, tem-se uma busca pouco "inteligente" no sentido de encontrar boas soluções. Entretanto, esta busca é eficiente quando se analisa a possibilidade da heurística entrar em ciclagem. As heurísticas **HB3** e **HB4**, são mais "inteligentes" pois, ao escolher o menor elemento do menor conjunto, tentam balancear os conjuntos. Por outro lado, nestas heurísticas, a possibilidade de entrar em ciclagem é maior.

Outro comentário que se pode fazer diz respeito aos movimentos realizados pelas heurísticas de busca. Nas heurísticas **HB1** e **HB3**, a única possibilidade de movimento consiste na troca de elementos entre os conjuntos. No entanto, ao utilizar estas heurísticas considerando que a solução inicial seja obtida por alguma das heurísticas de construção, observa-se que o número de elementos que contém cada conjunto fica predeterminado e, com isso, perde-se a possibilidade de se obter uma quantidade muito grande de soluções. Diante disso, foram implementadas as heurísticas **HB2** e **HB4** onde se tem a possibilidade de transferir um elemento de um conjunto para o outro, permitindo assim, que o número de elementos de cada conjunto seja mutável.

3.1 *Hill-Clibing* (HIC) e *Randon Generated Test* (RGT)

A estratégia de busca local *hill-clibing* só aceita um movimento se ele melhorar o valor da solução incumbente. Enquanto na estratégia *randon generated test* qualquer solução é aceita e no final escolhe-se a melhor.

Na Tabela 1 a seguir, tem-se uma comparação das quatro heurísticas de busca analisando o valor médio (para os cinco exemplos de cada tamanho) da solução em função do tempo (para todos os testes computacionais realizados e tempo computacional foi limitado a 2 minutos). A heurística de construção utilizada foi a **HC1**, devido a sua aleatoriedade.

Tabela 1: Hill-Clibing e Randon generated test (valores médios para os cinco exemplos de cada tamanho).

Tam	Método	HIC1	HIC2	HIC3	HIC4	RGT1	RGT2	RGT3	RGT4	RGT5
15	Solução	**78626419**	116542843	2790533886	1231450370	**1048865**	**1048865**	1110498591	**1048865**	**1048865**
	Melhor Iteração	88,4	53	25,2	14	5517,6	4321,2	45845,8	11121,4	16946
	Melhor Tempo	0	0	0	0	0	0	0,2	0	0
	Total de Iterações	12955222	12329299	16176968,2	15958204,6	13449688	12629326	16474751,8	16514827	5659613
35	Solução	**28950865**	49074822	4770748178	248763744	1901	283	20024085	**252,2**	15221
	Melhor Iteração	292,8	232,2	41,8	42,2	3248235	6452686	4523007	6246745	1248738
	Melhor Tempo	0	0	0	0	28,8	60,8	33,2	45,4	53,2
	Total de Iterações	12955155	12652390	16356344	16258087	13491949	12721603	16410441,8	16436058	2804799
55	Solução	**6113929**	6168531,4	234986698	112893245	2743,8	229,4	5691262223	**218,6**	31780,6
	Melhor Iteração	551,2	471,2	78,8	48	6112118	9277518	5580957	8595421	783721
	Melhor Tempo	0	0	0	0	54	87,6	41,4	62,8	50,6
	Total de Iterações	12987125	12725241	16518619	16783762,6	13522633	12762532	16231142,6	16471818	1858222
75	Solução	2984107	**2002506,4**	8113758034	194414065	1658,8	692	9738228792	**312,4**	58685,6
	Melhor Iteração	1487,2	585,8	52,2	35,2	7335878	5991873	6882706	10785355	675170
	Melhor Tempo	0	0	0	0	65,4	56,4	51	78,8	57,8
	Total de Iterações	12976673	12696289	16363394,2	16255370,4	13487129	12765115	16081582,8	16450626	1397457
95	Solução	3300751	**2174180,8**	6996863375	68570813,2	2912	**186,8**	11387104739	386,8	76538,8
	Melhor Iteração	6663,4	1410	89,6	55,6	6729429	6030738	3667770,2	10191587	541996
	Melhor Tempo	0	0	0	0	59,8	56,6	27,6	74,4	58,4
	Total de Iterações	12993163	12766777	16214519	16470085	13480141	12772257	15770986,6	16460689	1115821

Pode-se observar que, na estratégia de *hill-clibing* as heurísticas de busca param num ótimo local muito rapidamente chegando a soluções muito ruins. Entretanto, na estratégia *randon generated test* as heurísticas conseguem escapar de ótimos locais e as soluções são muito boas (a **RGT5** representa a utilização da **HC1** várias vezes). Esta estratégia será comparada com as meta-heurísticas utilizadas.

Comparando as heurísticas de busca na estratégia **RGT**, pode-se observar que a **HB3** obtém os piores resultados. Isto ocorreu devido ao fato de que para alguns exemplos esta heurística chega a resultados bons. Entretanto, para vários exemplos obtém-se resultados muito ruins devido a facilidade de entrar em ciclagem, parando em ótimos locais muito rapidamente, fazendo com que a média dos exemplos seja alta. A **HB1** demora mais para chegar a um ótimo local, entretanto, devido ao fato de

restringir o número de elementos de cada partição os resultados são apenas razoáveis. As melhores heurísticas de busca são a **HB2** e **HB4,** pois são as únicas que não restringem o número de elementos de cada partição.

4. *Simulated Annealing* (SA)

Simulated Annealing originou-se a partir da observação de um processo físico chamando *annealing* (Metropolis *et al.*, 1953), sendo que os conceitos físicos deste processo foram introduzidos para resolução de problemas de otimização combinatória por Kirkpatrick *et al.* (1983) e Cerny (1985). A idéia de SA é aceitar todas as soluções que melhoram o valor da função objetivo e, aceitar com uma certa probabilidade, soluções que piorem o valor da função objetivo, tentando assim, fugir de um ótimo local.

Explicações detalhadas com respeito a aplicação de **SA** em problemas de otimização combinatória podem ser encontradas em Reeves (1993) e Dias *et al.* (1996). Serão feitos, a seguir, alguns comentários com respeito a variação de cada um dos parâmetros.

- Temperatura Inicial

Inicialmente, testou-se vários valores fixos para a temperatura inicial (1000, 10000, 100000, dentre outros). No entanto, com a utilização de valores fixos para a temperatura os resultados obtidos estavam sendo pouco robustos. Acredita-se que isto ocorreu devido ao fato de que a temperatura inicial deve estar relacionada com o valor da função objetivo, pois, nos critérios de aceitação (ou não) de soluções ruins descritos a seguir estas duas grandezas estão diretamente relacionadas.

Diante disso, passou-se a utilizar uma temperatura inicial baseada em Dias *et al.* (1996) que consiste na seguinte fórmula: $T_{inicial} = \frac{\mu}{-\ln(\phi)} C$

Onde: C é o custo da solução atual obtida pela heurística de construção.
ϕ é a probabilidade de que uma solução que seja μ% pior que a atual seja aceita.

Observa-se que, utilizando esta função, consegue-se uma relação da temperatura inicial com a solução atual. Tem-se então dois parâmetros que podem ser variados para determinar a temperatura inicial. Foram feitas várias combinações entre os parâmetros ϕ e μ até que se conseguisse determinar quais eram as melhores (μ=0.8 e ϕ=0.9). Portanto, tem-se uma temperatura inicial bastante alta.

- Regra de Redução da Temperatura

A temperatura é reduzida por um fator $\alpha \in [0.8,0.99]$ sendo que, α=0.99 deu melhores resultados, ou seja, a temperatura deve ser reduzida lentamente.

- Temperatura Final (Critério de Parada)

Optou-se por utilizar o tempo de 2 minutos como critério de parada.

- Critério para Aceitação

Foram utilizadas duas diferentes funções de aceitação de uma solução ruim. A primeira consiste na função padrão de SA, onde, dado um número gerado aleatoriamente (com distribuição uniforme) entre 0 e 1, a solução só é aceita se este número é menor que o valor de $\exp^{(-\Delta custo/T)}$. A segunda consiste numa função determinística baseada em Dias *et al.* (1996) onde rejeita-se apenas as soluções que sejam muito ruins (Δcusto>T). Assim, observa-se que, para ambas as funções, a probabilidade de aceitação de uma solução ruim decresce de acordo com o aumento da variação do custo (Δcusto) e com a diminuição da temperatura (T).

Função	Descrição
F1	Rejeita soluções quando um número gerado entre 0 e 1 é maior que $\exp^{(\Delta custo/T)}$
F2	Rejeita soluções quando (Δcusto>T)

A Tabela 2 faz uma comparação entre estas duas funções de aceitação. Os parâmetros utilizados são: **HC3**, **HB2**, μ=0.8, ϕ=0.9, α=0.99 e o critério de parada é 2 minutos. Tem-se que **% Melhor** indica a porcentagem média de aceitação quando a melhor solução foi obtida e **% Final** indica a porcentagem média de aceitação no final dos 2 minutos.

Tabela 2: Comparação das funções de aceitação (valores médios para os cinco exemplos de cada tamanho).

Tamanho	15		35		55		75		95	
Função	F1	F2	F1	F2	F1	F2	F1	F2	F1	F2
Solução	1048865	1048865	**114,6**	165,8	234,6	**231,8**	**347,2**	356	**135,2**	164,8
Melhor Iteração	2748	3999,2	9281832	12998867	12465484	10562195	6311783	8545054	13940688	8073311
Melhor Tempo	0	0	54,6	73,8	74,6	59,2	36,8	48,6	83,6	45
Total de Iterações	19353134	21294424	19662039	21229030	19665627	21552413	19722050	21445311	19750803	21783584
%Melhor	100	43,2	83,2	80	74	86,4	86,2	89,8	72,8	94,4
%Final	57,2	54,8	61	61,8	58,6	59	59,6	60,4	60,8	60,6
Temp. Inicial	3,36E+10	3,36E+10	3,78E+10	3,78E+10	3,41E+10	3,41E+10	3,63E+10	3,63E+10	3,8E+10	3,8E+10
Temp. Final	6,86E+08	4,72E+08	7,48E+08	5,46E+08	6,7E+08	4,63E+08	7,07E+08	5,01E+08	7,47E+08	4,94E+08

Observa-se que a **F1** tem um desempenho um pouco melhor para este conjunto de parâmetros. Para as duas funções a porcentagem de aceitação quando a melhor solução é obtida é bastante alta e o tempo é baixo (em relação ao tempo final de 2 minutos).

5. Busca Tabu Curto Prazo (TCP)

Caracteriza-se pela utilização de uma memória de curto prazo para a orientação da busca, de forma a restringir a busca, impedindo que uma solução que foi visitada num passado recente seja visitada novamente. Os parâmetros utilizados são: atributos, regra de ativação tabu, tempo de permanência tabu e critério de aspiração. Maiores informações com respeito a cada um destes parâmetros podem ser obtidas em Laguna (1995), Glover e Laguna (1997), Reeves (1993) e Dias *et al.* (1996). A seguir, serão relatados os efeitos da variação de cada um destes parâmetros para o problema de partição de números.

- Atributos e Regra de Ativação Tabu

Foram utilizados dois atributos diferentes, dependendo da heurística de busca que esteja sendo utilizada.

	Atributo	Regra de ativação tabu
A1	(i,j)	Trocar i com j
A2	i	Transferir i de um conjunto para o outro

Como é possível observar na Seção 3, as heurísticas de busca que utilizam a combinação dos dois atributos deram melhores resultados. Cabe observar que o atributo A2 é mais restritivo que o A1.

- Tempo de Permanência Tabu

	Tempo tabu	Observação
TT1	X	Onde X é um número fixo
TT2	X*(N)	Onde N é a Quantidade de números a serem particionados
TT3	[X1*N,X2*N]	Onde X1 e X2 são números quaisquer

Inicialmente, foi testado um número fixo sem relação nenhuma com o tamanho do problema (TT1) e, observou-se que o algoritmo não era robusto, ou seja, obtinha resultados bons para alguns exemplos e ruins para outros. Posteriormente, testou-se um número dependente do tamanho do problema (TT2), obtendo-se resultados melhores. Por fim, testou-se a utilização de um intervalo dependente do tamanho do problema (TT3), o qual foi difícil de ajustar. Portanto utilizou-se nos testes finais um tempo tabu fixo de acordo com o tamanho do problema.

- Critério de Aspiração

Foi utilizado apenas um critério de aspiração que consiste na melhoria da solução incumbente, ou seja, se um movimento que melhora a solução incumbente estava tabu, ele é liberado de sua condição tabu.

- Critério de Parada

O critério de parada utilizado foi o tempo de execução de 2 minutos, pois, como os programas estavam sendo executados sempre no mesmo ambiente computacional (totalmente dedicado a execução do programa) este era um critério mais justo para comparação dos métodos.

- Variações do Método

Foram feitas duas variações na aplicação da busca tabu curto prazo:

Método	Descrição
TCP	Aplicação da busca tabu curto prazo simples
TCPR	Aplicação da busca tabu curto prazo com a utilização de *restart*

A primeira (**TCP**) consiste simplesmente na aplicação dos critérios descritos nas seções anteriores, até que o critério de parada fosse satisfeito.

A segunda (**TCPR**) consiste na aplicação da primeira até que algum critério de *restart* ou o critério de parada seja satisfeito. Quando um critério de r*estart* é satisfeito, todos os movimentos que estão proibidos são liberados e a busca é reiniciada a partir da solução incumbente.

Foram utilizados vários critérios para a aplicação de *restart*. Por exemplo, de acordo com o tempo de execução ou número de iterações, sendo que, este tempo/iterações eram contados somente a partir da não melhoria da solução incumbente. Outro critério utilizado foi determinar o *restart* a partir da porcentagem de movimentos que estavam sendo proibidos. Entretando, em geral o **TCP** obteve melhores resultados (ver Tabela 3).

A partir dos dados, observa-se que a **TCPR** não colaborou para um melhor desempenho da meta-heurística. Em geral, para os parâmetros utilizados, não compensa fazer *restart*. Entretanto, observou-se que o desempenho da **TCPR** melhora (em relação a **TCP**) à medida que o tempo tabu aumenta e que o critério de parada é mais longo. Diante disso, conclui-se que a aplicação de *restart* é vantajosa quando a quantidade de movimentos que estão proibidos é muito grande, mas esta não é a situação encontrada nos conjuntos de parâmetros que obtiveram melhores soluções.

6. Busca Tabu Longo Prazo (TLP)

Os métodos de **TLP** implementados consistem basicamente na aplicação da **TCP** por algum tempo e, posteriormente, aplica-se estratégias de intensificação e diversificação.

- Soluções de elite

Inicialmente, testou-se a estratégia de deixar a busca tabu de curto prazo rodando por algum tempo, sendo que neste tempo, todas as vezes que se atualizava a solução incumbente, guardava-se estas informações numa matriz de residência e, a partir de um certo tempo, passava-se a usar estas informações para uma estratégia de intensificação, ou seja, de busca de soluções parecidas com as que tinham sido armazenadas e que supostamente eram boas soluções, ou ainda, soluções de elite. Entretanto, percebeu-se que essas soluções de elite não eram tão boas, pois muitas soluções ruins obtidas no início da busca estavam no conjunto de soluções de elite.

Num segundo passo, foi implementada uma estratégia onde o conjunto de soluções de elite foi fixado num determinado tamanho e, após rodar a busca tabu de curto prazo durante um determinado tempo, obtém-se neste conjunto de elite as melhores soluções sendo que a quantidade de soluções de elite obtida é dependente do tamanho predeterminado.

Foram utilizados vários tamanhos diferentes para determinar a quantidade de soluções de elite. Inicialmente, utilizou-se tamanhos pequenos (10, 20, 50) com a esperança de que seria interessante ter soluções de elite realmente boas para aplicar a intensificação. Como os resultados não estavam sendo bons, foram utilizados alguns conjuntos de soluções de elite grandes (100, 1000, 10000) e, a partir destes conjuntos, aplicou-se estratégias de intensificação e diversificação, entretanto os resultados também não foram bons.

- Estratégias de Intensificação e Diversificação

A partir das informações obtidas do conjunto de soluções de elite, foi feita uma busca visando a melhoria da solução. Foram utilizadas três estratégias diferentes:

- **TLP1:** A partir das informações da matriz de residência, obtida a partir da utilização da **TCP** por um determinado período de tempo foi construída uma solução inicial para aplicação das

estratégias de intensificação ou diversificação. Estas estratégias consistem em aplicar uma heurística de busca aleatória onde se dá um peso maior para o valor que mais interessa (dependendo da estratégia que se esteja adotando).

- **TLP2:** A única diferença em relação a **TLP1** é a utilização da solução incumbente como solução inicial para as estratégias de intensificação e diversificação.
- **TLPR:** Semelhante **TCPR** descrita anteriormente. Entretanto, após o *restart*, utiliza-se uma solução atual obtida a partir das informações de residência, sendo que, na **TCPR** utilizava-se a solução incumbente após o *restart*.

Os dois primeiros métodos foram ruins sendo que a melhor solução foi encontrada enquanto se estava aplicando a busca tabu curto prazo. O terceiro método obteve os melhores resultados, mas, ainda assim, estes resultados não foram muito bons. Este fato pode ser observado na Tabela 3, onde se tem uma comparação dos dois métodos de busca tabu curto prazo com o melhor método longo prazo. Os parâmetros utilizados foram: **HC1**, **HB2**, **TT2**, critério de *restart* de acordo com a porcentagem de movimentos que estavam sendo proibidos e o critério de parada igual a 2 minutos.

Tabela 3: Comparação entre as estratégias de busca tabu (valores médios para os cinco exemplos de cada tamanho).

Tamanho	15			35			55		
Estratégia	TCP	TCPR	TLP	TCP	TCPR	TLP	TCP	TCPR	TLP
Solução	1048865,2	1048865,2	1048865	**340,2**	711	414,6	**101**	259,4	479
Melhor Iteração	7611	7611	7611	5991930,4	6872819	3191488	6359522	6909774	3313178
Melhor Tempo	0	0	0	60,6	63	32,2	64,2	63,2	33,8
Total de Iterações	11808685,8	13334148	12324063	11866981	13639185	12383682	11885738	13446139	11997148

Tamanho	75			95		
Estratégia	TCP	TCPR	TLP	TCP	TCPR	TLP
Solução	**336**	558,4	264,8	610,8	**364,4**	574
Melhor Iteração	7811945	6970619,2	5183549	2310090,6	8889229	3985538
Melhor Tempo	79,4	63,6	52,6	23	81,6	39,8
Total de Iterações	11815306,6	13450513	12065958	11830702	13276098	11923510

7. Algoritmos Genéticos e Meméticos

A principal diferença destas meta-heurísticas para as outras das seções anteriores, é que elas trabalham com uma população de soluções e não com apenas uma solução de cada vez. Sendo que, nos algoritmos genéticos a evolução não tem memória enquanto que nos algoritmos meméticos permite-se uma evolução cultural da população.

7.1 Operadores genéticos

- Operador de Reprodução (*crossover*)

Foram utilizados três tipos de operadores de reprodução: ***one-point crossover***, ***two-point crossover***, e ***crossover uniforme***. Estes operadores foram utilizados individualmente ou em combinações, sendo que, os melhores resultados foram obtidos quando eram utilizadas combinações dos três operadores.

- Operador de Mutação

Foram utilizados dois operadores de mutação diferentes:

Mutação	Descrição
M1	Transferir uma porcentagem de elementos de um conjunto escolhido aleatoriamente para outro.
M2	Transferir uma porcentagem de elementos de um conjunto de maior valor para o de menor valor.

A porcentagem escolhida para a mutação variou de 10% a 30% do tamanho do exemplo. Para **M1**, a variação desta porcentagem não influenciou muito nos resultados. Entretanto, para **M2**, quanto maior a porcentagem de mutação melhor foi a solução. Este resultado era esperado, pois, **M2** consiste numa mutação onde é feita uma espécie de busca local que acaba balanceando o indivíduo.

- Escolha dos pais

A escolha dos pais foi feita de inúmeras maneiras: dois melhores, melhor com um aleatório, dois aleatórios, dentre outras. Foi observado que, quando um e, principalmente quando os dois melhores eram fixos, o algoritmo entrava em crise de diversidade muito rapidamente. Os melhores resultados foram obtidos quando um dos pais era escolhido aleatoriamente e o outro, algumas vezes era escolhido aleatoriamente e outras vezes escolhia-se o melhor indivíduo. Com isso, tentou-se fazer um balanço entre manter a diversidade da população, e tentar melhorar a qualidade das soluções.

7.2 População

- Representação da População

A representação utilizada para a população consiste numa matriz, onde cada linha representa um indivíduo e as colunas representam um gene de cada indivíduo. Cabe lembrar que foi utilizada a representação binária para cada indivíduo. Assim, o número de linhas da matriz representa o tamanho da população e o número de colunas representa o tamanho do exemplo.

- Tamanho da População

Inicialmente, foram testadas algumas populações iniciais com tamanhos fixos (10, 20, 40, 80). Entretanto, os resultados não estavam sendo satisfatórios. Na etapa seguinte, foram utilizados tamanhos de população que variavam de acordo com o tamanho do exemplo (N, 2N....), os quais, também não deram bons resultados. Observou-se que variar o tamanho da população de forma proporcional ao tamanho do exemplo não era uma boa idéia, pois, exemplos grandes tinham as maiores populações e isto tornava o processo de geração de população muito lento. Diante disso, foram testados (sem sucesso) alguns exemplos cujo tamanho da população era inversamente proporcional ao tamanho do exemplo (1000/N, 500/N).

- Geração da População Inicial

A população inicial é gerada de forma totalmente aleatória utilizando a heurística **HC1**.

7.3 Variações no Algoritmo

Dado um determinado algoritmo que havia sido implementado, após muitas tentativas de ajustar os parâmetros sem conseguir obter resultados satisfatórios (em relação aos resultados que já haviam sido obtidos pelas outras meta-heurísticas), decidiu-se várias vezes mudar parcialmente (ou até totalmente) o algoritmo. A seguir descreve-se três variantes do algoritmo:

- **Primeira versão:** dada uma população inicial com tamanho fixo, os dois melhores (menor valor da função objetivo) indivíduos eram escolhidos e passava para a população seguinte por elitismo. Dois elementos da população eram escolhidos e eram cruzados. O filho sofria mutação e era incluído na população no lugar do pior elemento. Testou-se também incluir no lugar de um elemento qualquer, ou só incluí-lo na população se ele fosse melhor que algum indivíduo. Este procedimento era feito até que se atingisse algum critério de *restart*, por exemplo, tempo, número de iterações, crise de diversidade (medida pela comparação entre os valores da função objetivo). Foram feitos inúmeros testes utilizando diferentes combinações de parâmetros sem chegar a resultados satisfatórios (em relação aos resultados obtidos pelas outras meta-heurísticas).
- **Segunda versão:** a diferença em relação à primeira versão é a utilização duas matrizes, uma para a população que estava sofrendo os cruzamentos e outra para a população que estava sendo gerada. Com isso, tentou-se evitar que indivíduos resultantes de cruzamentos recentes fossem eliminados da população muito rapidamente. Os resultados foram um pouco melhores do que os da primeira versão, porém continuam insatisfatórios.
- **Terceira versão:** concluiu-se que o problema dos dois primeiros métodos estava na forma como era feito o *restart*, pois, apenas os dois melhores indivíduos da população eram mantidos após o *restart* e todo o resto da população era gerado novamente. Com isso, toda a evolução que se tinha

conseguido após algumas gerações de população era perdida. Diante disso, o algoritmo foi totalmente modificado, ou seja, implementou-se um algoritmo onde não era feito *restart*. A população inicial era gerada aleatoriamente, os dois melhores passavam para a população seguinte por elitismo. O procedimento utilizado para a geração de uma nova população consiste em gerar uma porcentagem de indivíduos com cada um dos três operadores de reprodução e uma porcentagem é gerada aleatoriamente pela **HC1**. Com isso, pretende-se diversificar um pouco a população. Esta versão apresentou os melhores resultados que são mostrados na Tabela 4.

7.4 Algoritmos Meméticos

Os algoritmos meméticos implementados consistem simplesmente em aplicar o algoritmo genético com um procedimento de busca local (utilizamos a **HB4**) no indivíduo antes de inseri-lo na população. Entretanto, para as duas primeiras versões de algoritmo genético, a melhoria obtida com a aplicação da busca local era perdida ao fazer o *restart*.

Para a terceira versão do algoritmo genético, a aplicação de busca local obteve melhorias surpreendentes (ver Tabela 4). O procedimento de aplicação de busca local consiste no seguinte: ao invés de fazer a mutação do indivíduo, como se faz no algoritmo genético, foi feita uma aplicação de busca local por um número predeterminado de iterações.

Tabela 4: Comparação entre algoritmos genéticos e meméticos (valores médios para os cinco exemplos de cada tamanho).

Tamanho	15		35		55		75		95	
Método	GA	MA	GA	MA	GA	MA	GA	MA	GA	MA
Solução	1048865,2	1048865,2	3629	**53,8**	3963,4	**227,4**	4816,8	**212,4**	4886,4	**108,8**
Melhor Iteração	5211,2	128	1763354	156504,8	905092,4	222500	814484	154157,6	531124,4	199895,6
Melhor Tempo	0	0	80,6	54,2	63,4	79,6	77	57,4	63,4	78,6
Total de Iterações	5748132,8	368030	2640119,6	344414	1717180,4	333275,6	1269387	321482	1006868	305616,8

8. Comparação das Meta-heurísticas

O objetivo desta seção é fazer uma comparação entre as várias meta-heurísticas implementadas, bem como, as estratégias de busca descritas na Seção 3. Foram escolhidas as melhores combinações de parâmetros encontradas para cada método e, para que possa ser feita uma comparação justa, o **critério de parada** utilizado em todos os métodos é um tempo de 2 minutos. Além disso, vale lembrar que o ambiente computacional utilizado é sempre o mesmo.

A primeira comparação a ser feita é quanto ao **nível de dificuldade de implementação**. Todas as meta-heurísticas foram relativamente fáceis de implementar. A facilidade de implementação do problema de partição de números ajudou para que isto ocorresse. A título de comparação pode-se utilizar a seguinte ordem crescente de dificuldade: **RGT, SA, BTC, BTL, GA/ MA.**

Em se tratando de meta-heurísticas, uma medida importante é quanto à **dificuldade de ajuste dos parâmetros**. Em geral, esta é a maior dificuldade das meta-heurísticas. Neste trabalho, observou-se a seguinte ordem crescente de dificuldade: **RGT, SA, BTC, BTL, GA/ MA.**

Antes de analisar a qualidade das soluções obtidas pelas meta-heurísticas, apresenta-se a Tabela 5, onde se tem: o valor da solução obtida por uma heurística de construção chamada KK desenvolvida por Karmarkar e Karp (1982); o valor da solução ótima obtida pelo método "*Complete Karmarkar Karp*" (CKK) desenvolvido por Korf (Korf (1995) e Korf (1998)); o número de nós para se obter a solução ótima pelo método CKK.

Tabela 5: Resultados: heurística KK e Solução Ótima (valores médios para os cinco exemplos de cada tamanho).

Tamnho	15	35	55	75	95
KK	64389124	298682,2	117049,4	28164,4	3542,4
Solução Ótima	1048865,2	3,8	0,2	0,8	0,4
Número de Nós	750,6	86928089	6185744	408653,8	908339

A seguir, tem-se a Tabela 6 onde analisa-se a qualidade da solução, o tempo gasto (M. tempo) e o número de iterações (M. iteração) para obter a **melhor solução**. Pode-se observar que, enquanto a heurística de busca **HIC** obteve resultados ruins, a heurística **RGT** obteve bons resultados chegando, em alguns casos, a superar os procedimentos meta-heurísticos. Com exceção da meta-heurística **GA**, os procedimentos meta-heurísticos obtiveram resultados bons e robustos. Vale ressaltar a grande diferença obtida entre o **MA** e **GA** usando-se o mesmo conjunto de parâmetros, com a diferença que, em **MA** aplica-se a busca local **HB4** no indivíduo antes de inseri-lo na população. Observa-se ainda que, na maioria dos exemplos, o tempo computacional gasto para obter a melhor solução foi baixo em relação ao tempo total de 2 minutos.

Tabela 6: Comparação dos resultados valores médios para os cinco exemplos de cada tamanho).

Tam	Método	HIC	RGT	SA	BTCP	GA	MA
15	M. Iteração	13470994	16514827,4	19353134	11808686	5748132,8	368030
	Sol. Inicial	4424976790	11193038813	4424976790	11193038813	4424976790	4424976790
	Sol. Obtida	89453562	**1048865,2**	**1048865,2**	**1048865,2**	**1048865,2**	**1048865,2**
	M. Tempo	0	0	0	0	0	0
35	M. Iteração	13551285	16436058,4	19662038,8	11866981	2640119,6	344414
	Sol. Inicial	4975792561	18688598250	4975792561	18688598250	4975792561	4975792561
	Sol. Obtida	32722712	252,2	114,6	340,2	3629	**53,8**
	M. Tempo	0	45,4	54,6	60,6	80,6	54,2
55	M. Iteração	13549141	16471818	19665626,8	11885738	1717180,4	333275,6
	Sol. Inicial	4493399123	31299250681	4493399123	31299250681	4493399123	4493399123
	Sol. Obtida	7834749	218,6	234,6	**101**	3963,4	227,4
	M. Tempo	0	62,8	74,6	64,2	63,4	79,6
75	M. Iteração	13551165	16450626	19722049,8	11815307	1269387,2	321482
	Sol. Inicial	4779481853	28924677909	4779481853	28924677909	4779481853	4779481853
	Sol. Obtida	3328397	312,4	347,2	336	4816,8	**212,4**
	M. Tempo	0	78,8	36,8	79,4	77	57,4
95	M.Iteração	13516878	16460689,2	19750803,2	11830702	1006868	305616,8
	Sol. Inicial	5010086708	31583915872	5010086708	31583915872	5010086708	5010086708
	Sol. Obtida	1585882	386,8	135,2	610,8	4886,4	**108,8**
	M. Tempo	0	74,4	83,6	23	63,4	78,6

9. Comentários Finais e Trabalhos Futuros

Neste trabalho, várias meta-heurísticas foram aplicadas ao problema de partição de números onde foi feita uma análise de cada método mostrando os vários passos necessários para se chegar ao conjunto de parâmetros final, facilitando assim que outros estudos sejam desenvolvidos com base no processo descrito aqui.

Escolheu-se o problema de partição de números, pois, de acordo com vários outros trabalhos da literatura, este é um problema desafiante para a aplicação de meta-heurísticas. Além disso, é um problema que torna fácil a caracterização do fenômeno de "fase de transição". Na Tabela 5 é possível observar que existe uma grande dificuldade de resolução para exemplos com cerca de 35 números, o que mostra claramente a "fase de transição" para esse problema, que neste caso, é uma fase do tipo "fácil-difícil-fácil", ou seja, para exemplos com 15 números o problema é relativamente fácil e, à medida que aumenta o tamanho do problema (após 35 números), o problema se torna relativamente fácil novamente.

Uma explicação intuitiva para este fenômeno é dada em Berretta, *et al.* (2003), e consiste no seguinte: suponha que temos N pedras com diferentes pesos e que o peso médio seja P. Deve-se separar estas pedras em dois grupos com o mesmo peso. Para um número muito grande de pedras N (assumindo que se mantenha a média P) tem-se um valor muito pequeno para P/N, como se cada pedra fosse tão pequena, que pudesse ser vista como grãos de areia. Isso torna muito mais fácil a tarefa de separar em dois grupos com mesmo peso, pois existe uma grande quantidade de soluções ótimas e sub-ótimas. Para um número pequeno de pedras N, embora o número de soluções ótimas seja

reduzido, o problema pode ser resolvido facilmente por um algoritmo exato. Portanto, para caracterizar a dificuldade de resolver este problema, torna-se clara a importância da relação entre as grandezas N e P (que no caso do problema de partição de números seriam a quantidade de números e o valor médio desses números, respectivamente).

Em trabalhos futuros deve-se aprofundar os estudos com relação a métodos de solução, mais especificamente sobre o fenômeno de "fase de transição" aplicados ao problema de partição de números e também a outros problemas de otimização combinatória.

Agradecimentos: Os autores agradecem as contribuições feitas pelos árbitros anônimos.

BIBLIOGRAFIA

ARGUELLO, M. F., FEO, T. A. e GOLDSCHIDT, O. (1996), Randomized Methods for the Number Partitioning Problem. *Computers and Operations Research v. 23, n. 2, p. 103-111.*

BERRETTA, R.E. e MOSCATO, P (1999), *The Number Partitioning Problem, An Open Challenge For Evolutionary Computation?*, Chapter 17 of New Ideas In Optimisation, D. Corne, M. Dorigo, and F. Glover (eds.), p. 261-278, McGraw-Hill, UK.

BERRETTA, R., COTTA, C. e MOSCATO, P. (2003), *Enhancing the Performance of Memetic Algorithms by Using a Matching-based Recombination Algorithm: results on the number partitioning problem*. Meta-heuristics: Computer Decision-Making (M. G. C. Resende and J. Souza, editors), Kluwer

CERNY, V. (1985) Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulatetd Algorithm, *Journal of Optimization Theory and Applications, Vol 45, pp 41-45*.

DIAZ, A., GLOVER, F., GHAZIRI, H. M., GONZÁLEZ, J. L., LAGUNA, M. MOSCATO, P. e TSENG, F. T. (1996), *Optimización Heurística y Redes Neurolales*, Editorial Paraninfo, Espanha.

GEN, M. e CHENG, R. (1997) *Genetic Algorithms & Engineering Design*, Wiley.

GLOVER, F. e LAGUNA, M. (1997), *Tabu Search*, Kluwer Academic Publishers, USA.

JOHNSON, D. S., ARAGON, C. R., McGEOCH, L. A. e SCHEVON, C., (1991), Optimization by Simulated Annealing: An experimental evaluation; Part II: Graph coloring and Number Partitioning. *Operations Research. 39, n. 3, p. 378-406*.

KARMARKAR, N. e KARP, R. (1982), The Differencing Method of Sett Patitioning, *Technical Report UCB/CSD 82/113, Computer Science Division, University of California, Berkeley*.

KIRKPATRICK, S., C. D. GELATT Jr, M. P. VECCHI (1983) Optimization by Simulated Annealing, *Sience, Vol 220, pp 671-680*.

KORF, R. E. (1995), From Approximate to Optimal Solutions: a case study of number partitioning. *C. S. Mellish editor, Proceedings of 14th Joint Conference on Artificial Intelligence, (IJCAI-95), 266-272, Montreal Canada, Morgan Kauffman.*

KORF, R. E. (1998), A Complete Anytime Algorithm for Number Partitioning. *Artificial Intelligence, v. 106, n. 2, p. 181-203.*

LAGUNA, M. (1995), *Tabu Search Tutorial*, II Escuela de Verano Latino Americana de Investigación Operativa.

MERTENS, S. (2004), The Esiest Hard Problem: Number Partitioning, *to appear in Computational Complexity and Statistical Physics (Oxford University Press, New York)* http://arxiv.org/ftp/cond-mat/papers/0310/0310317.pdf.

METROPOLIS, N., A. W. ROSENBLUTH, M. N. ROSENBLUTH, A. H. TELLER, E. TELLER (1953) Equation of State Calculations by Fast Computing Machines, *Journal of Chemical Physics, Vol 21, pp 1087-1092.*

MICHALEWICZ, Z. (1992) *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag.

RAYWARD-SMITH, V. J., OSMAN, I. H., REEVES, C. R. E SMITH, G. D. (1996) *Modern Heuristic Search Methods*, Wiley, Inglaterra.

REEVES, C.R. (1993), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell.

STADLER, P. F., HORDIJK, W. e FONTANARI, J. F. (2003), Phase Transition and Landscape Statistics of the Number Partitioning Problem. *Santa Fe Institute, Working Paper 03-02-006* (http://www.santafe.edu/sfi/publications/Working-Papers/03-02-006.pdf).