UNIVERSIDADE FEDERAL FLUMINENSE

FABRÍCIO DA SILVA PERES

Heurísticas GRASP para o Problema de Formação de Células de Manufatura

NITERÓI

2006

UNIVERSIDADE FEDERAL FLUMINENSE

FABRÍCIO DA SILVA PERES

Heurísticas GRASP para o Problema de Formação de Células de Manufatura

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Otimização Combinatória e Inteligência Artificial.

Orientador:
Luiz Satoru Ochi

NITERÓI

2006

Heurísticas GRASP para o Problema de Formação de Células de Manufatura

Fabrício da Silva Peres

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre.

Aprovada por:

Prof. Luiz Satoru Ochi / IC-UFF (Presidente)

Prof. Carlile Campos Lavor / UNICAMP

Profa. Loana Tito Nogueira / IC-UFF

Niterói, 13 de Novembro de 2006.

Agradeço a Deus por mais esta conquista.

Agradecimentos

Agradeço primeiramente a Deus por ter me dado capacidade e auxílio para concluir este mestrado.

Agradeço tambem a minha família pelo incentivo e por ter me dado toda a base necessária para conseguir chegar até aqui.

E finalmente agradeço a minha noiva, que já será minha esposa no dia da defesa, pelo apoio, motivação e compreensão dos momentos que precisei ficar ausente para concluir este curso.

Resumo

O objetivo deste trabalho é a de analisar o desempenho da heurística GRASP na solução do problema de formação de células de manufatura(PFCM). Neste contexto, são efetuadas análises experimentais através de resultados empíricos comparando com resultados obtidos por outros algoritmos da literatura.

O desenvolvimento dos algoritmos propostos foi realizado através de uma evolução, inserindo novas técnicas em cada nova versão. Foram utilizadas técnicas como filtro na fase de construção, mineração de dados e reconexão por caminhos.

Entretanto, após os testes realizados e os resultados obtidos, os algoritmos aqui propostos não conseguiram superar os melhores resultados conhecidos na literatura. Os resultados colocaram o melhor algoritmo proposto em terceiro lugar no comparativo geral com os algoritmos da literatura. Apesar disto este algoritmo conseguiu superar os resultados de cinco outros algoritmos da literatura.

Abstract

The main reason of this research is to analyze how the GRASP heuristic works in solving problems of manufacturing cell formation . In this context, there are made so many experimental analysis, comparing empiric results to results obtained from other kind of algorithm mentioned in literature.

The design of all proposal algorithms was made by an evolution, always inserting new techniques to each step. We can mention some of these techniques used as filter, in construction phase, data mining and path relinking.

However, after finishing tests, the results obtained with proposal algorithms couldn't get better than the best algorithms found in literature. The best proposal algorithm was ranking in third in general comparison, that places it better than other five algorithms found in literature.

Keywords: Metaheuristic, GRASP, Cluster, Cell formation problem, Data Mining, Path Relinking

Palavras-chave

1. Metaheurística
2. GRASP
3. Clusterização
4. Problema de Formação de Células de manufatura
5. Mineração de dados
6. Reconexão por caminhos

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A utilização da metaheurística GRASP tem se destacado na literatura de métodos heurísticos devido à sua simplicidade e pelo seu bom desempenho na solução de diferentes problemas de otimização combinatória com elevado grau de complexidade (problemas NP-Completo e NP-Difícil).

O problema de formação de células de manufatura é um problema de clusterização com elevado grau de complexidade onde é necessário agrupar máquinas em células, agrupar peças em famílias e unir as células e famílias em clusters.

As máquinas pertencentes a uma mesma célula devem ter em comum o fato de serem utilizadas por uma mesma família de peças. Da mesma forma, as peças pertencentes há uma mesma família devem ter em comum o fato de serem produzidas pela mesma célula de máquinas.

Neste contexto, a meta principal deste trabalho é desenvolver diferentes versões da heurística GRASP para resolver o problema de formação de células de manufatura e efetuar uma análise experimental através de resultados empíricos comparando-os com resultados obtidos por outros algoritmos da literatura.

No total serão apresentadas cinco versões, onde cada versão incluirá uma nova técnica na versão anterior para buscar melhores resultados.

O primeiro algoritmo a ser desenvolvido utilizará os conceitos básicos da heurística GRASP, ou seja, será um algoritmo iterativo onde cada iteração é constituída de duas fases: construção e busca local. A fase de construção é responsável por criar uma solução inicial viável e a fase de busca local percorre a vizinhança desta solução inicial tentando melhorá-la. A melhor solução encontrada entre todas as iterações é fornecida como resultado.

O segundo algoritmo a ser apresentado utiliza a técnica de filtro na fase de construção que consiste em gerar várias soluções a cada iteração da fase de construção antes de passar para a fase de busca local.

O terceiro algoritmo utiliza a técnica de mineração de dados a cada n iterações. A mineração retornará uma nova solução através de uma "combinação"das melhores soluções encontradas até aquele ponto.

O quarto algoritmo utiliza a técnica de reconexão por caminhos. Com esta técnica uma nova solução é gerada alterando uma solução base, elemento a elemento, até chegar a uma solução

alvo.

E o quinto e último algoritmo a ser apresentado é uma versão híbrida utilizando todas as técnicas anteriores.

Este trabalho está divido em sete capítulos. O segundo capítulo define e explica o problema de formação de células de manufatura e mostra algumas medidas de similaridade e de desempenho utilizadas para avaliar os resultados; o terceiro capítulo apresenta algumas técnicas de resolução do problema de formação de células de manufatura; o quarto capítulo contém uma explicação sobre a heurística GRASP; o quinto capítulo contém o funcionamento dos algoritmos propostos neste trabalho; o sexto capítulo contem os resultados dos algoritmos propostos, comparações entre eles e os algoritmos da literatura; e o sétimo capítulo contém as conclusões deste trabalho.

Capítulo 2

Problema de Formação de Células de Manufatura (PFCM)

2.1 Introdução

Para que uma indústria consiga obter bons resultados é necessário gerenciar com qualidade e eficiência seu sistema de manufatura, de modo que economize os custos e aumente a velocidade da produção sem que se perca a qualidade dos produtos produzidos.

A forma de gerenciamento de um sistema de manufatura depende, entre outras coisas, da variedade e volume da produção. Indústrias dedicadas a produzir uma pequena variedade de produtos e em grande volume utilizam **"sistema de manufatura orientado ao produto"** através de **"linhas de produção"** como forma de gerenciamento.

No sistema orientado ao produto, cada linha de produção é dedicada à produção de um único produto. Para isto todas as máquinas necessárias na criação de um determinado produto são alocadas no mesmo local. Isto é feito para acelerar a produção que será realizada em grande escala.

Este sistema é caracterizado por baixa manipulação de material, facilidade de controle e gerenciamento do sistema, e baixo tempo de produção. Porém não são flexíveis [42].

No caso de indústrias dedicadas na criação de uma grande variedade de produtos em escala média fica inviável a utilização do gerenciamento por linha de produção, já que seria necessário criar uma linha de produção para cada um dos diversos produtos produzidos. E como estes não seriam produzidos em grande escala o gasto com as máquinas de cada linha de produção não seria compensado. Nestes casos é necessário utilizar outra forma de gerenciar o sistema de manufatura.

Uma forma existente para gerenciar sistemas de manufatura deste tipo é o **"sistema de manufatura orientado por processo"** que consiste em agrupar máquinas com funções semelhantes em um mesmo local formando o que chamamos de clusters de máquinas especializadas em funções específicas, ou departamentos especializados em determinadas funções. Desta forma, cada peça de um dos produtos da indústria passaria pelos departamentos necessários à sua manufatura, ou seja, pelos departamentos que possuem as máquinas necessárias.

Com a formação de clusters de máquinas, os diversos produtos produzidos na indústria compartilham as máquinas disponíveis ao invés de utilizarem máquinas únicas na sua manufatura, como no caso de linhas de produção.

Este sistema é caracterizado por sua grande flexibilidade devido à alta reutilização das máquinas, porém necessita de muita manipulação de material, possui dificuldade no controle e tempo alto de produção [42].

Outra forma existente de gerenciar sistemas de manufatura de indústrias com grande variedade de produtos e volume médio é a utilização da **"manufatura celular"**.

A manufatura celular une as vantagens dos sistemas orientado ao produto e orientado por processo e é voltado para indústrias com grande variedade de produtos e volume médio de produção.

Na manufatura celular, máquinas com diferentes funções utilizadas para produzirem peças em comum são agrupadas em *células*; e peças que necessitam de máquinas em comum são agrupas em *famílias*.

Cada célula de máquinas é dedicada a produzir uma família de peças. Desta forma, criam-se clusters, onde cada cluster é formado por uma célula de máquinas e uma família de peças.

O **sistema de manufatura celular perfeito** é obtido ao criar clusters tais que todas as peças de um cluster sejam produzidas apenas pelas máquinas existentes neste mesmo cluster e todas as máquinas sejam utilizadas na manufatura destas peças. Com esta formação as peças não precisam ser deslocadas para outros clusters (locais) para que sejam concluídas, e as máquinas não são subutilizadas. Desta forma ganha-se em tempo, manuseio das peças e aproveitamento das máquinas.

Apesar desta formação ser a ideal, uma solução desta forma é difícil de ser encontrada na prática.

A manufatura celular apresenta vantagens como: facilidade no controle e gerenciamento do sistema de manufatura, redução do transporte e manuseio de peças, velocidade da produção e maior segurança no trabalho.

Em um nível conceitual, a modelagem das células de manufatura desconsidera vários fatores do processo de manufatura como a capacidade de produção das máquinas, tempo de duração de cada operação, custo de cada operação e etc. E considera apenas as operações das máquinas sobre as peças, ou seja, considera apenas a informação de qual máquina é utilizada na produção de qual peça [29].

Utilizando este conceito, um sistema de manufatura pode ser representado por uma matriz binária de ordem m x p utilizada para relacionar as m máquinas e as p peças. Nesta matriz um elemento $a_{i,j}$ = 1 indica que a máquina i realiza uma operação sobre a peça j e um elemento $a_{i,j}$ = 0 indica o contrário.

A tabela 2.1 ilustra um sistema de manufatura com m = 8 máquinas e p = 12 peças, representado por uma matriz binária relacionando máquinas e peças (instância Seiffodini Wolfe - 1986). As máquinas estão representadas pelas linhas da matriz numeradas de 0 a 7 e as peças estão representadas pelas colunas da matriz numeradas de 0 a 11.

	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
M0	1	1	1	1	-	-	-	-	-	-	-	-
M1	1	-	1	1	1	1	1	-	-	1	-	-
M2	-	-	1	1	1	1	1	1	1	-	-	-
M3	-	-	-	-	-	1	1	1	1	1	-	-
M4	-	-	-	-	-	-	1	1	1	1	-	-
M5	-	-	-	-	-	-	1	1	1	-	1	-
M6	-	-	-	-	-	-	-	-	-	-	1	1
M7	-	-	-	-	-	-	-	-	-	-	1	1

Tabela 2.1: Exemplo de matriz binária representando um sistema de manufaturas com 8 máquinas e 12 peças (Seiffodini Wolfe).

A tabela 2.1 mostra que a máquina 0 realiza operações apenas sobre as peças 0, 1, 2 e 3 ($a_{0,0}$ = 1, $a_{0,1}$ = 1, $a_{0,2}$ = 1, $a_{0,3}$ = 1); a máquina 1 realiza operações sobre as peças 0, 2, 3, 4, 5, 6 e 9 e assim por diante para as demais máquinas.

Formar uma célula de manufatura para o sistema representado pela tabela 2.1 consiste em formar cluster com células de máquinas que atendam famílias de peças. A representação de uma possível solução para este sistema está ilustrado na tabela 2.2.

	P0	P1	P2	P3	P4	P10	P11	P5	P6	P7	P8	P9
M0	**1**	**1**	**1**	**1**	**-**	-	-	-	-	-	-	-
M1	**1**	**-**	**1**	**1**	**1**	-	-	1	1	-	-	1
M6	-	-	-	-	-	**1**	**1**	-	-	-	-	-
M7	-	-	-	-	-	**1**	**1**	-	-	-	-	-
M2	-	-	1	1	1	-	-	**1**	**1**	**1**	**1**	**-**
M3	-	-	-	-	-	-	-	**1**	**1**	**1**	**1**	**1**
M4	-	-	-	-	-	-	-	**-**	**1**	**1**	**1**	**1**
M5	-	-	-	-	-	1	-	**-**	**1**	**1**	**1**	**-**

Tabela 2.2: Possível solução para o exemplo de sistema de manufatura.

A solução encontrada para o sistema de manufatura da tabela 2.1 e ilustrada pela tabela 2.2 contem 3 clusters:

O cluster 1 é formado pela célula de máquinas M0 e M1 e pela família de peças P0, P1, P2, P3 e P4; O cluster 2 é formado pela célula de máquinas M6 e M7 e pela família de peças P10 e P11; O cluster 3 é formado pela célula de máquinas M2, M3, M4 e M5 e pela família de peças P5, P6, P7, P8 e P9;

Podemos observar pela tabela 2.2 que a solução ideal não foi encontrada. A solução ideal deveria atender duas condições:

- Todas as peças serem atendidas apenas pelas máquinas do seu cluster. Pode-se observar na solução que este item não foi atendido pela quantidade de elementos inter-clusters (elementos iguais a 1 fora dos clusters). Os elementos $a_{1,5}$, $a_{1,6}$, $a_{1,9}$, $a_{2,2}$, $a_{2,3}$, $a_{2,4}$ e $a_{5,10}$ são elementos inter-clusters por não pertencerem a nenhum cluster. Isto na prática significa que uma peça localizada em um cluster precisa passar por uma máquina de outro cluster para ser produzida.

- Todas as máquinas do cluster serem utilizadas por todas as peças. Pode-se ver que existem elementos iguais a zero dentro dos clusters, significando que algumas máquinas não são utilizadas por algumas peças do seu cluster. Os elementos da figura 2.2 nestas condições são os elementos $a_{1,1}$, $a_{2,9}$, $a_{4,5}$, $a_{5,5}$ e $a_{5,9}$.

Como é raro formar clusters de manufatura que atendam as condições acima, existem pesquisas para resolver o problema de dependência entre clusters como, por exemplo, através de duplicação de máquinas.

Utilizando duplicação de máquinas na solução mostrada na tabela 2.2 poderíamos duplicar a máquina 1 no cluster 3, a máquina 2 no cluster 1 e a máquina 5 no cluster 2. Porém duplicar as máquinas significa investimento na compra de novas máquinas, o que pode não ser interessante para o gestor do sistema de manufatura, além disto estas máquinas duplicadas não seriam utilizadas pelas outras peças do cluster, causando desperdício das mesmas.

2.2 Particularidades da solução

Uma solução para o PFCM consiste no agrupamento de máquinas e peças em clusters da melhor forma possível. E o número de clusters que a solução gerada conterá é um fator determinante para a qualidade da solução e para o tempo de sua obtenção. Pois quanto maior o número de clusters maior é a variedade de possíveis soluções e maior o tempo para encontrar a melhor solução.

Por ser um fator tão crítico, a quantidade ideal de clusters da solução deve ser escolhido com cuidado. Porém como é difícil para o gestor do sistema de manufatura definir previamente o melhor valor desta variável, muitos algoritmos realizam esta tarefa ao gerar uma solução para o PFCM.

Outro item importante a se observar é o fato de que uma solução que possua algum cluster contendo apenas uma máquina ou apenas uma peça é considerada uma solução inválida. Isto

porque a resolução do PFCM consiste em agrupar máquinas e peças em clusters para agilizar a fabricação das peças. Neste sentido a literatura tem como regra proibir este tipo de solução.

Também não são consideradas soluções válidas aquelas que possuem cluster em que haja alguma máquina que não atenda nenhuma peça do seu cluster ou alguma peça que não seja atendida por nenhuma máquina do seu cluster.

2.3 Medidas de Similaridade e de Desempenho para o PFCM

Uma solução do PFCM consiste de clusters contendo células de máquinas e famílias de peças agrupadas de maneira tal que se possa obter melhor desempenho pelo sistema de manufatura. O sistema ideal consiste de clusters tais que todas as peças do cluster sejam produzidas apenas pelas máquinas existentes neste mesmo cluster e todas as máquinas sejam utilizadas na manufatura destas peças.

As células devem conter as máquinas mais similares entre si e da mesma forma as famílias devem conter as peças mais similares entre si. Para isto existem medidas de similaridade para calcular a similaridade entre duas máquinas e entre duas peças. Algumas medidas de similaridade serão apresentadas nesta seção.

Esta seção será utilizada também, para apresentar medidas de desempenho utilizadas na literatura para avaliar a qualidade das soluções geradas para o PFCM. As mais usadas são a Eficiência de Agrupamento e a Eficácia de Agrupamento.

As medidas de similaridade e de desempenho apresentadas nesta seção são destinadas à avaliação de soluções obtidas com base apenas em matriz binária de entrada do PFCM.

2.3.1 Medidas de Similaridade

Existem muitas medidas de similaridade diferentes baseadas em matriz binária. Neste capítulo serão apresentas as medidas Similaridade de Jaccard, Similaridade Euclidiana e Similaridade baseada em comparação de vetores.

2.3.1.1 Similaridade de Jaccard

A fórmula que define a similaridade de Jaccard é a seguinte:

$$J_{ij} = \frac{Y_{ij}}{(Y_i + Y_j - Y_{ij})} \tag{2.1}$$

onde
Y_{ij} = Número de peças (máquinas) em comum atendidas (usadas) pelas máquinas (peças) i e j.
Y_i = Número de peças (máquinas) atendidas (usadas) pela máquina (peça) i.
Y_j = Número de peças (máquinas) atendidas (usadas) pela máquina (peça) j.

A similaridade entre duas máquinas será máxima (igual a 1) quando todas as peças atendidas por elas forem as mesmas, ou seja, quando $Y_{ij} = Y_i = Y_j$ e a similaridade será igual a zero quando as duas máquinas não atenderem nenhuma máquina em comum, ou seja, $Y_{ij} = 0$. O mesmo acontece para a similaridade entre duas peças.

Esta medida tem um problema apresentado por Vakharia e Wemmerlov em [39] que ocorre quando Y_j » Y_i (Y_j é muito maior do que Y_j). Por exemplo, seja $Y_i = 10$, $Y_j = 100$, $Y_{ij} = 10$, $Y_k = 10$ e $Y_{ik} = 5$. Se aplicarmos a fórmula, então $J_{i,j}$ = 10 / (10 + 100 -10) = 0,1 e $J_{i,k}$ = 5 / (10 + 10 - 5) = 0,33 , ou seja, segundo a fórmula as máquinas i e k são mais similares do que i e j apesar de j atender todas as peças que i atende.

Neste mesmo trabalho, Vakharia e Wemmerlov propuseram a seguinte modificação nesta medida de similaridade para contornar esta falha.

$$S_{ij}(\alpha) = \alpha \frac{y_{ij}}{y_i} + (1 - \alpha) \frac{y_{ij}}{y_j} \tag{2.2}$$

onde
$y_i \leq y_j$
$0 \leq S_{ij}(\alpha) \leq 1$

Com esta alteração os valores máximo (1) e mínimo (0) são encontrados de acordo com as mesmas condições da similaridade de jaccard original. Porém com esta alteração, o projetista do sistema pode dar pesos diferentes para as relações y_{ij}, y_i (primeira parte da equação) e y_{ij}, y_j (segunda parte da equação). Desta forma, em casos em que Y_j » Y_i, o projetista pode dar um peso maior para a relação entre y_{ij} e y_i.

2.3.1.2 Similaridade Euclidiana

A fórmula que define a similaridade Euclidiana é a seguinte:

$$EUC_{ij} = \sqrt{\sum_{k=1}^{m} x_{ijk}} \tag{2.3}$$

onde
m = número de máquinas (peças)
x_{ijk} = 0 se apenas uma das peças (máquinas) i ou j utilizarem (atender) a máquina (peça) k; 1 caso contrário.

A similaridade entre duas peças será máxima quando todas as máquinas usadas por elas forem as mesmas, ou seja, quando x_{ijk} = 1 para todo k. O mesmo acontece para a similaridade entre máquinas.

2.3.1.3 Similaridade baseada em comparação de vetores

Esta medida foi proposta em [28] e é calculada através da seguinte fórmula:

$$S_{ij} = \frac{y_{ij}(y_{ij} + y_0)}{(y_{ij}\sqrt{(y_{ij} + y_0)} + y_i + y_j)\sqrt{y_{ij} + y_0}} \tag{2.4}$$

onde
y_{ij} = número de máquinas que atendem as peças i e j.
y_i = número de máquinas que atendem apenas a peça i.
y_j = número de máquinas que atendem apenas a peça j.
y_0 = número de máquinas que não atendem nem a peça i nem a peça j.

S_{ij} será igual a 0 se nenhuma máquina atender as peças i e j (y_{ij} = 0) e será igual a 1 se y_{ij} for diferente de 0 e todos os outros parâmetros iguais a 0.

2.3.2 Medidas de Desempenho

Existe uma grande variedade de medidas para avaliar a qualidade de uma solução. As medidas mais conhecidas são a Eficiência de Agrupamento e a Eficácia de Agrupamento. Este tópico irá descrever estas e outras medidas.

2.3.2.1 Eficiência de Agrupamento

Segundo [24], esta medida foi proposta por Chandrasekharan e Rajagopalan em 1986 e é calculada da seguinte forma:

$$EA = qr_1 + (1 - q) \times r_2 \tag{2.5}$$

onde
r_1 = razão entre o número de operações (1´s) dentro dos clusters e o número de elementos dentro dos clusters.
r_2 = razão entre o número de lacunas (0´s) fora dos clusters e o número total de elementos fora dos cluster´s.
q = fator de peso ($0 \leq q \leq 1$)
$0 \leq r \leq 1$

O valor de r_1 será máximo (1) quando todos os elementos dos clusters forem iguais a um e será mínimo (0) quando todos os elementos dos clusters forem iguais a zero.

O valor de r_2 será máximo (1) quando todos os elementos inter-clusters (fora dos clusters) forem iguais a zero e será mínimo (0) quando todos os elementos inter-clusters forem iguais a um.

Esta medida possui duas falhas apontadas por [24] e [33]. A forte dependência do parâmetro q e a tendência da medida ficar sempre entre 0,75 e 1 nos testes apresentados na literatura. Desta forma, mesmo soluções ruins obtêm uma medida perto de 0,75.

2.3.2.2 Eficácia de Agrupamento

Segundo [24], esta medida foi proposta por Kumar e Chandrasekharan e é calculada da seguinte forma:

$$T = \frac{e - e_0}{e + e_v} \tag{2.6}$$

onde
e = número de operações (1‘s) na matriz
e_0 = número de elementos inter-clusters (1´s fora dos clusters)
e_v = número de elementos iguais a zero dentro dos clusters
$0 \leq T \leq 1$

A eficácia de agrupamento encontra o valor máximo quando o número de elementos inter-clusters (e_0) e o número de elementos iguais a zero dentro dos clusters (e_v) forem nulos.

Quanto maior for o valor de (e_0) e (e_v) menor será o resultado da eficácia de agrupamento.

Esta medida é utilizada por vários algoritmos apresentados na literatura e será a medida utilizada neste trabalho para avaliar as soluções geradas pelos algoritmos.

Uma falha apresentada por Sarker em [33] é que a eficácia de agrupamento não é sensível a soluções com clusters que contenham linhas ou colunas vazias. Desta forma o algoritmo que utiliza esta medida precisa eliminar estas soluções porque estas não são soluções válidas e a medida de desempenho não é capaz de identificar isto.

2.3.2.3 Índice de Capacidade de Agrupamento

Segundo [24], esta medida foi proposta por Hsu e é calculada da seguinte forma:

$$GCI = 1 - \frac{e_0}{e} \tag{2.7}$$

onde
e = número de operações (1‘s) na matriz
e_0 = número de elementos inter-clusters (1´s fora dos clusters)
$0 \leq GCI \leq 1$

O GCI encontra o valor máximo (1) quando o número de elementos inter-clusters for igual a zero e encontra o valor mínimo (0) quando todas as operações(1´s) forem elementos inter-clusters.

O GCI possui a falha de não levar em consideração a quantidade de elementos iguais a zero dentro dos clusters. Desta forma soluções com clusters que contenham vários elementos iguais a zero podem ser consideradas boas soluções por esta medida.

2.3.2.4 Eficiência Global

Segundo [24], esta medida foi proposta por Harhalakis e é calculada da seguinte forma:

$$GLE = \frac{e_1}{e} \tag{2.8}$$

onde
e_1= número de elementos iguais a 1 dentro dos clusters
e = número de operações (1‘s) da matriz
$0 \leq \text{GLE} \leq 1$

O GLE encontra o valor máximo (1) quando todas as operações da matriz estiverem dentro dos clusters e o valor mínimo (0) quando todas as operações forem inter-clusters.

Assim como GCI, esta medida não leva em consideração a quantidade de lacunas dentro dos clusters. Desta forma soluções com clusters que contenham vários elementos iguais a zero podem ser consideradas boas soluções através desta medida.

2.3.2.5 Medida Primária

Segundo [16] e [33], esta medida foi proposta por Zhang e é utilizada por Mak em [21]. Esta medida é calculada da seguinte forma:

$$ng = \frac{e_1}{(e_1 + e_v)} - \frac{e_i}{e_0} \tag{2.9}$$

onde e1 = número de operações executadas dentro dos clusters
e_v = número de lacunas nos clusters
e_0 = número de operações inter-clusters
e = número de operações da matriz
$-1 \leq \text{ng} \leq 1$

A medida primária alcança o valor máximo quando o número de operações inter-clusters e o número de lacunas dentro dos clusters forem iguais a zero e todas as operações forem realizadas dentro dos clusters.

O valor mínimo (-1) é encontrado quando todas as operações da matriz forem operações inter-clusters.

2.3.2.6 Medida de eficiência de Agrupamento Duplamente Ponderada

Esta medida foi proposta por Sarker em [33] e é calculada da seguinte forma:

$$nQ = (\frac{q_1 e_1 + (1 - q_1)e_v}{e_1 + e_v}) \times (\frac{q_2 e_1 + (1 - q_2)e_0}{e_1 + e_0}) \tag{2.10}$$

onde

q_1 e q_2 são fatores de peso, tais que $0 \leq q_1, q_2 \leq 1$
e_1 = número de operações executadas dentro dos clusters
e_v = número de lacunas nos clusters
e_0 = número de operações inters-clusters
$0 \leq nQ \leq 1$

Sarker utilizou dois pesos para que o projetista do sistema de manufatura pudesse atribuir pesos diferentes para a eficiência intra-cluster (primeira parte da equação) e para a eficiência inter-cluster (segunda parte da equação).

A medida encontra o valor máximo quando o número de operações inter-clusters e o número de lacunas forem iguais a zero e os pesos q_1 e q_2 forem iguais a um. Neste caso a medida se resume à nQ = $(q_1 e_1/e_1) \times (q_2 e_1/e_1)$.

Esta medida apresenta falhas em duas situações. Se os pesos forem iguais a zero e não houver nenhuma operação dentro dos clusters a medida encontrará o valor máximo, porém esta solução deve ser considerada inválida pelo fato de todos os clusters estarem vazios. E se todas as operações da matriz estiverem dentro dos clusters e não houver nenhuma lacuna, a medida encontrará valor zero se os pesos q_1 e q_2 forem nulos, apesar de ter avaliado uma solução ótima.

2.3.2.7 Índice de Célula

Proposta por Mukattash em [24]] esta medida é calculada da seguinte forma:

$$CI = \frac{1}{n} \times \sum_{j=1}^{p} \frac{k_j \times n_j}{k_j + v_j + e_j} \tag{2.11}$$

onde
n = número total de máquinas
n_j = número total de máquinas no j-ésimo cluster
v_j = número de lacunas no j-ésimo cluster
e_j = número de elementos inter-cluster no j-ésimo cluster
k_j = número de operações no j-ésimo cluster
p = número de clusters
$0 \leq CI \leq 1$

O índice de célula encontra o valor máximo quando o número de lacunas (v_j) nos clusters e o número de elementos inter-clusters (e_j) forem nulos. E encontra o valor mínimo (0) quando o número de operações (k_j) de todos os clusters for igual a zero.

O índice de célula permite ao projetista do sistema avaliar a participação de cada cluster no resultado final.

2.4 Formulação Matemática do PFCM

O PFCM é um caso especial dos problemas de clusterização automática e portanto classificado como um problema NP-Completo [17]. Desta forma, o uso exclusivo de métodos exatos de

otimização fica restrito á instâncias de pequenas dimensões.

2.4.1 Número de Soluções

A obtenção de células de manufatura de forma otimizada é um trabalho difícil, pois um sistema de manufatura pode ter um elevado número de soluções possíveis, e encontrar a melhor solução dentre elas leva muito tempo se for necessário testar cada uma delas.

Como a formação de células de manufatura pode ser classificado como um problema de clusterização é possível calcular o número de soluções possíveis através do número de Stirling dado pela equação 2.12, que calcula o número de maneiras de se agrupar n dados em c clusters mutuamente exclusivos e não vazios segundo [41]

$$\sum_{i=1}^{c}(-1)^{c-i} \times \frac{i^n}{[i! \times (c-i)!]} \tag{2.12}$$

Na formação de células de manufatura, pode-se dizer que as peças ou as máquinas são os dados e as células e famílias são os clusters. Desta forma o número de maneiras de agrupar n peças ou máquinas em c clusters é dada pela equação 2.12 . Porém o problema de formação de células de manufatura precisa agrupar as máquinas em células e as peças em famílias. E cada célula pode ser associada a qualquer família para formação de clusters. Sendo assim o número de possíveis soluções para p peças, m máquinas e c clusters é dado pela equação 2.13.

$$(\sum_{i=1}^{c}(-1)^{c-i} \times \frac{i^p}{[i! \times (c-i)!]}) \times (\sum_{i=1}^{c}(-1)^{c-i} \times \frac{i^m}{[i! \times (c-i)!]}) \tag{2.13}$$

Aplicando a equação 2.13 para a formação de células de manufatura com 3 clusters, variando o número de peças de 15 a 100 e de máquinas de 10 a 70 obtém-se os seguintes resultados:

Número de Peças	Número de máquinas	Número de soluções
15	10	$2{,}2159683 \times 10^{10}$
60	40	$1{,}431603835 \times 10^{46}$
100	70	$3{,}583372838 \times 10^{79}$

Tabela 2.3: Exemplo de número de soluções para um PFCM.

Observando na tabela 2.3 a grande quantidade de soluções geradas , chega-se à conclusão de que é impraticável testar cada uma das possíveis soluções para encontrar a melhor solução no caso de instâncias de elevadas dimensões. Esta fato tem estimulado o desenvolvimento de heurísticas e metaheurísticas para resolver este problema de forma aproximada.

2.4.2 Representação matemática do agrupamento de elementos em clusters

Ainda de acordo com [41] podemos modelar matematicamente a alocação de elementos em clusters através do modelo descrito por 2.14 a 2.16. Esta formulação pode ser usada para o agrupamento de peças em cluster ou de máquinas em cluster, porém não contempla a clusterização de ambas simultaneamente.

Maximizar:

$$\sum_{k=1}^{c}\sum_{i=1}^{n}\sum_{j<i} s_{ij} \times x_{ik} \times x_{jk} \tag{2.14}$$

sujeito a:

$$\sum_{k=1}^{c} x_{ik} = 1, i \in 1, 2, ..., n \tag{2.15}$$

$$\sum_{i=1}^{n} x_{ik} < u, k \in 1, 2, ..., c \tag{2.16}$$

onde
$x_{ik} \in 0, 1$
i = 1,2,...,n
k = 1,2,...,c
c = número de clusters a serem formados
n = número de elementos
u = número máximo de elementos por cluster
s_{ij} = índice de similaridade entre os elementos i e j
x_{ik} = 1 se o elemento i pertence ao cluster k e 0 caso contrário.
x_{jk} = 1 se o elemento j pertence ao cluster k e 0 caso contrário.

Para cada par de elementos dentro de um mesmo cluster, o resultado da equação 2.14 aumenta de acordo com a similaridade entre os dois elementos. Sendo assim, quanto maior a similaridade entre os elementos de um mesmo cluster maior será o resultado da equação.

As equações 2.15 garantem que um elemento só pode estar em um cluster e as inequações 2.16 garantem que um cluster não contêm mais elementos que o permitido.

Capítulo 3

Técnicas para resolução do Problema de Formação de Células de Manufatura

3.1 Introdução

A manufatura celular tem se mostrado uma eficiente maneira de se gerenciar o sistema de manufatura de uma indústria, porém obter uma solução eficiente para cada necessidade é uma tarefa difícil.

Desta forma, várias técnicas e algoritmos foram desenvolvidos para se obter a melhor solução possível. Sendo que a melhor solução viável é aquela que atende as restrições do problema e mais se aproxima da solução ideal, onde as peças não precisam ser deslocadas para outros clusters para serem produzidas e as máquinas sejam totalmente utilizadas nos custers a que elas pertencem, ou seja, criar clusters que sejam mais independentes quanto possível.

Os algoritmos mais comumente encontrados na literatura podem ser classificados da seguinte forma: clusterização baseada em array, clusterização hierárquica, clusterização não-hierárquica, algoritmos baseados em programação matemática, algoritmos baseados em teoria dos grafos, algoritmos baseados em inteligência artificial e outras heurísticas segundo [17].

Segue abaixo uma breve descrição de cada um destes métodos segundo [17] e [29].

3.2 Clusterização Baseada em *Array*

Os algoritmos deste tipo utilizam como descrição do problema uma matriz binária de entrada que define quais máquinas realizam tarefas sobre quais peças.

Dada a matriz de entrada, o algoritmo realiza várias movimentações nas linhas e nas colunas a fim de agrupar as máquinas em células, as partes em famílias e finalmente construir os clusters agrupando células e famílias.

O algoritmo *Bond Energy* (BEA) [33] e o algoritmo *rank order clustering* [22] são os dois algoritmos mais antigos e mais conhecidos desta classe.

3.3 Clusterização Hierárquica

Os métodos de clusterização hierárquica diferentemente dos métodos baseados em matriz não conseguem criar as células de máquinas e as famílias de peças ao mesmo tempo. É necessário definir cada uma separadamente.

Os algoritmos desta classe utilizam como dado de entrada um coeficiente de similaridade que deve ser calculado previamente através de dados que compõem o sistema de manufatura. Com este coeficiente o algoritmo determina uma hierarquia de grupos, sendo que em cada nível de hierarquia pode haver um número diferente de grupos com diferente número de membros. No caso do PFCM os grupos são as células ou famílias e os membros são as máquinas ou peças.

Os algoritmos desta classe podem ser divididos em duas categorias: divisivo e aglomerativo. Os algoritmos divisivos iniciam colocando todas as máquinas (ou peças) em uma mesma célula (ou família) e vai iterativamente particionando as células até que cada máquina esteja em uma célula. Os algoritmos aglomerativos, ao contrário, iniciam com cada máquina em uma célula e vai unindo as células até que o número desejado de células seja formado.

Uma vantagem deste método sobre os métodos baseados em matriz é o fato de poder utilizar outras informações além de apenas qual máquina atende qual peça, oferecendo assim a possibilidade de se levar em conta na solução fatores como capacidade de produção de uma máquina, tempo da produção, volume de produção de uma peça e etc.

Algoritmos de clusterização hierárquica são utilizados para resolver o PFCM em [5] na segunda parte do algoritmo proposto e no algoritmo divisivo proposto por Harhalakis em [14].

3.4 Clusterização Não-Hierárquica

Algoritmos desta classe também encontram os clusters através de uma função de similaridade, porém diferentemente da clusterização hierárquica não há a necessidade de computação prévia desta função. Entretanto é necessário informar o número de clusters a ser encontrado, o que é um valor difícil de se obter em diferentes aplicações.

O algoritmo coloca cada máquina (ou peça) em uma célula (ou família) e iterativamente vai adicionando outras máquinas à célula de acordo com uma função de similaridade.

O algoritmo ZODIAC [6] e Srinivasan [35] são exemplos de algoritmos que utilizam clusterização não-hierárquica.

3.5 Algoritmos baseados em Programação Matemática

O agrupamento de máquinas ou peças pode ser tratado como um problema de otimização e, portanto pode-se usar métodos exatos de programação matemática para resolvê-lo.

Os algoritmos que utilizam programação matemática possuem a capacidade de restringir o tamanho de cada célula , restringir o número de células e utilizar diferentes processos em cada uma de suas etapas.

Uma desvantagem dos métodos desta classe é o fato de não conseguir agrupar máquinas e peças simultaneamente.

O modelo das P-Medianas de Kusiak [19], que é um dos modelos mais populares, minimiza o total de distâncias entre cada par de máquinas ou peças. Outros modelos de programação inteira foram desenvolvidos por Boctor [4], Choobineh [7], Gunasingh e Lashkari [13], e Purcheck [26].

Srinivasan [35] desenvolveu um modelo de atribuição para o problema de formação de células enquanto Steudal e Ballakur [36] aplicaram um modelo de programação dinâmica.

A maior parte destes algoritmos envolve modelos de programação inteira linear ou não-linear. Isto os torna muito custosos quando utilizados para instâncias com número elevado de máquinas e peças devido ao tempo computacional necessário para se chegar a uma solução ótima.

3.6 Algoritmos Baseados em teoria dos Grafos

Algoritmos baseados em grafos também são utilizados para resolver o PFCM. Neste caso o problema é representado como um grafo onde os vértices são as máquinas ou peças que são conectadas por uma aresta que representa a relação de similaridade entre elas.

Rajagopalan e Batra [27] representaram o problema de formação de células através de grafo, onde os vértices são as máquinas (peças). Estes vértices são conectados por uma aresta caso as máquinas (peças) sejam utilizadas por peças (máquinas) em comum. Este algoritmo tem a desvantagem de não agrupar as máquinas e as peças simultaneamente.

Vanneli e Kumar [40] desenvolveram em 1986 um modelo de grafo para obter uma estrutura de blocos diagonal perfeita utilizando duplicação de máquinas. Em 1987 Kumar e Vanneli [18] desenvolveram um procedimento com o mesmo objetivo através de terceirização de peças.

Em 1991 Askin [2] propôs um algoritmo de caminho hamiltoniano para o problema de agrupamento. O algoritmo encontrava um caminho hamiltoniano que reorganizasse as linhas da matriz para encontrar uma estrutura diagonal de blocos perfeita.

3.7 Algoritmos baseados em Inteligência Artificial

Existem alguns métodos baseados em conceitos de inteligência artificial elaborados para a resolução do problema de formação de células de manufatura.

Utilizando redes neurais pode-se citar o modelo de Nguema [3] baseado no modelo de Hopfiel que tem como principal vantagem a capacidade de resolver instâncias grandes. Shankar [34] apresenta um modelo transitoriamente caótico e compara os resultados com quatro outros modelos que também utilizam redes neurais.

Um algoritmo que utiliza lógica fuzzy é encontrado em [28], onde a entrada do algoritmo é uma matriz máquinaxpeça de relação fuzzy onde cada elemento representa o grau de relacionamento entre a máquina e a peça.

Como os algoritmos genéticos são uma metaheurística muito utilizada para resolver vários problemas de otimização, existem vários algoritmos que utilizam esta técnica para resolver o PFCM. Em [12] é apresentado uma técnica combinando algoritmo genético com uma heurística de busca local. O algoritmo genético é utilizado para construir células de máquinas e a busca local para agrupar as peças às máquinas. Em [17] é apresentado um algoritmo genético com programação inteira que usa uma representação única para indivíduos (máquinas / peças). Esta formulação reduz o tamanho do problema de formação de células. Em [10] é apresentado um algoritmo genético construtivo baseado no problema das p-medianas.

Outra técnica utilizada para resolver o PFCM é a busca tabu que inicia com uma solução semente e procura uma solução melhor na vizinhança. Soluções para o PFCM através da busca tabu foram propostas em [1], [37] e [8].

Simulated annealing é uma metaheurística de busca local probabilística e soluções para o PFCM utilizando esta técnica são apresentadas em [15] e [42].

Capítulo 4

Metaheurística GRASP

4.1 Introdução

Neste capítulo descreveremos a metaheurística GRASP cujos conceitos serão utilizados nos algoritmos propostos neste trabalho.

As metaheurísticas são procedimentos genéricos para a solução aproximada de problemas computacionalmente difíceis. As metaheurísticas mais conhecidas são os algoritmos genéticos, a busca tabu, *simulated annealing*, GRASP, colônias de formigas e VNS. Cada uma utiliza um mecanismo diferente para fugir de ótimos locais e tentar se aproximar da solução ótima global. A adaptação de uma metaheurística para um certo problema cria uma heurística para este problema.

A metaheurística GRASP (*greedy randomized adaptative search procedure*) proposta por Feo e Resende [9] é um procedimento iterativo onde cada iteração é constituída de duas fases: construção e busca local. A fase de construção é responsável por criar uma solução inicial viável e a fase de busca local percorre a vizinhança desta solução inicial tentando melhorá-la. A melhor solução encontrada entre todas as iterações é fornecida como resultado do GRASP.

As definições do GRASP aqui apresentadas foram baseadas no trabalho de Maurício Resende e Celso Ribeiro em [30].

O algoritmo 1 ilustra o pseudocódigo da metaheurística GRASP.

Algoritmo 1 GRASP

```
1: ler entrada
2: for k = 1 até número_de_iterações do
3:     solução1 = construção(semente)
4:     solução2 = busca local (solução1)
5:     guarda melhor solução ( solução2 , melhor solução)
6: end for
```

A fase de construção consiste em criar uma solução inicial iterativamente elemento a elemento. Cada elemento candidato a fazer parte da solução é avaliado e recebe um valor através de uma função gulosa, a partir disto é criada uma lista LCR (lista de candidatos restrita) com os melhores elementos. Após a criação desta lista o algoritmo escolhe um elemento aleatori-

amente da mesma e o adiciona na solução parcial. Para a escolha do próximo elemento, cada elemento restante é reavaliado pela função gulosa e uma nova lista LCR é criada. Devido a este procedimento, pode-se dizer que o algoritmo de construção é iterativo, guloso, aleatório e adaptativo.

Para definir a inclusão de um elemento na lista LCR em um problema de maximização, considere t um elemento candidato a entrar na lista, C o conjunto de todos os elementos candidatos, g(t) a função gulosa utilizada , tmin e tmax o menor e o maior valor encontrado pela função até o momento e $\alpha \in [0,..,1]$ o fator de aleatoriedade. Desta forma a inclusão de um elemento na lista LCR pode ser determinada por t $\in$ C | g(t) $\geq$ (tmin + (1 - α) (tmax - tmin)) .

O algoritmo 2 ilustra o procedimento de construção para um problema de maximização.

Algoritmo 2 Procedure construção

```
1: Solução = { }
2: while solução imcompleta do
3:    tmin = min ( g(t) | t ∈ C )
4:    tmax = max ( g(t) | t ∈ C )
5:    LCR = { t ∈ C | g(t) >= ( tmin + ( 1 - α ) ( tmax - tmin) ) }
6:    Escolhe um elemento t ∈ LCR aleatoriamente
7:    Solução = solução ∪ { t }
8: end while
9: return Solução
```

onde:
t - um elemento
C - Conjunto de todos os elementos candidatos
g(t) - função gulosa que avalia o elemento
α - fator que define a aleatoriedade da escolha de um elemento na LCR.

Pelo pseudocódigo pode-se notar que o fator α define o nível de aleatoriedade da escolha de um elemento na lista LCR. Se α = 0 as escolhas serão totalmente gulosas (|LCR| = 1) e se α = 1 as escolhas serão totalmente aleatórias (|LCR| = C).

Segundo [25], a maior parte das implementações de GRASP da literatura utilizam uma LCR baseada em alguma função gulosa. As implementações tradicionais utilizam valores fixos para α, porém segundo [23], a fase construtiva de um algoritmo GRASP com α fixo pode nunca gerar uma solução global ótima em alguns casos.

O GRASP pode ser visto como técnica de múltiplos reinícios(*multistart*) onde cada iteração produz uma solução a partir de uma distribuição desconhecida. Desta forma, a média e a variância das soluções obtidas ao longo das iterações dependem da geração da LCR. Por exemplo, se a LCR for restrita a um único elemento (puramente guloso) então a mesma solução será gerada em todas as iterações e a variância da distribuição será nula. Se a LCR possuir vários elementos então a variância das soluções será maior e o valor médio das soluções será pior, porém a melhor solução encontrada tende a ser melhor do que a solução gerada pelo algoritmo puramente guloso.

Foi realizado um estudo em [30] sobre a influência do parâmetro α, que restringe a LCR, na diversidade e qualidade das soluções geradas e o tempo computacional levado pelo algoritmo

GRASP.

Segundo este estudo, quanto maior a variância de soluções encontradas na fase de construção maior é a variância das soluções encontradas pela busca local. E como o algoritmo GRASP é iterativo e gera várias soluções, a chance de encontrar uma solução ótima aumenta com a maior variedade das soluções geradas. Porém com o aumento do número de soluções geradas, aumenta também o tempo computacional gasto na busca local.

Conclui-se então que a escolha do parâmetro α é crítica e fundamental para o bom desempenho do algoritmo GRASP.

Após a geração da solução inicial pela fase de construção, um algoritmo GRASP tradicional realiza uma fase de busca local com base nesta solução para tentar encontrar soluções vizinhas de melhor qualidade. Uma solução vizinha consiste em uma solução que difere da solução inicial em poucos elementos como, por exemplo, no caso do PFCM soluções que contenham uma peça colocada em clusters diferentes são soluções vizinhas.

A busca local examina cada solução vizinha através de uma medida de desempenho f() e verifica se alguma é melhor do que a solução inicial. Após verificar a vizinhança é retornada a melhor solução encontrada.

O algoritmo 3 ilustra o pseudocódigo de uma busca local.

Algoritmo 3 Procedure Busca Local

```
1: Melhor_solução s* = solução_inicial
2: Obtem a vizinhança V
3: while percorre vizinhança V do
4:     Busca solução s' de V
5:     if f(s') > f(s*) then
6:         s* = s'
7:     end if
8: end while
9: return S*
```

4.2 Variações do GRASP

Existem várias técnicas e adaptações para tentar melhorar a performance do GRASP alterando alguma fase do algoritmo ou combinando com técnicas de inteligência artificial e analise combinatória.

Abaixo estão descritas algumas destas técnicas.

4.2.1 GRASP reativo

O GRASP reativo consiste em não utilizar um parâmetro α fixo na fase de construção para todas as iterações. Desta forma haverá uma maior variedade de soluções iniciais geradas na fase de construção.

Para tal procedimento o algoritmo pode por exemplo, escolher um α diferente em cada iteração a partir de um conjunto de possíveis valores. A escolha do melhor α é guiada pelo comportamento do algoritmo em cada iteração passada.

A aplicação desta técnica tende a proporcionar um algoritmo mais robusto do que o GRASP tradicional e com soluções melhores devido à maior diversidade de soluções geradas.

4.2.2 GRASP com Filtro

O GRASP com filtro consiste em simplesmente gerar mais de uma solução na fase de construção de cada iteração GRASP antes de realizar a busca local.

O algoritmo com filtro avalia cada solução encontrada durante n iterações na fase de construção através da medida de desempenho f(). Após isto, retorna a melhor solução encontrada ao final das iterações para ser processada pela fase de busca local.

O algoritmo 4 ilustra o pseudocódigo do filtro na fase de construção

Algoritmo 4 Filtro na Construção

```
1: while n < iterações_no_filtro do
2:     s = construção();
3:     if f(s) > f(s*) then
4:         s* = s
5:     end if
6:     Incrementa n
7: end while
8: return s*
```

A utilização desta técnica permite a geração de soluções iniciais melhores para a fase de busca local do GRASP.

O risco desta técnica é a possibilidade da fase de construção gerar soluções iniciais iguais em cada iteração GRASP. Para que isto não ocorra pode-se guardar as soluções iniciais já utilizadas pelo algoritmo em um conjunto. E as novas soluções iniciais geradas só serão utilizadas caso elas não façam parte deste conjunto.

4.2.3 GRASP com Mineração de dados

O GRASP com mineração de dados consiste em utilizar a técnica de mineração de dados para auxiliar o GRASP a encontrar melhores resultados.

Mineração de dados é uma técnica que consiste em encontrar informações úteis dentro de um banco de dados com várias informações de um determinado problema. No caso da utilização conjunta com o GRASP a mineração de dados pode ser utilizada para gerar uma nova solução a partir de padrões encontrados nas soluções geradas anteriormente. Desta forma, pode-se encontrar uma solução ótima que seja vizinha das melhores soluções encontradas anteriormente e que ainda não tenha sido avaliada.

Para executar este procedimento é necessário guardar as melhores soluções encontradas

durante as iterações GRASP em um conjunto que será "minerado"para encontrar uma nova solução a cada x iterações.

Exemplos da utilização de GRASP com mineraçao de dados podem ser encontrado nos trabalhos [31] e [32] realizados por Santos e Ochi.

O algoritmo 5 ilustra o pseudocódigo da utilização de mineração de dados.

Algoritmo 5 Grasp com Mineração de Dados

```
1: ler entrada
2: for k =1 até número_de_iterações do
3:    solução1 = construção(semente)
4:    solução2 = busca local (solução1)
5:    guarda_melhor_solução ( solução2 , melhor_solução)
6:    armazena_solução ( vetor_de_melhores_soluçoes, melhor_solução)
7:    if k é divisível por x then
8:       Solução3 = mineração_dados(vetor_de_melhores_soluçoes)
9:       guarda melhor solução ( solução3 , melhor_solução)
10:      armazena solução ( vetor_de_melhores_soluçoes, melhor_solução)
11:   end if
12: end for
```

4.2.4 GRASP com reconexão por caminhos

O GRASP com reconexão por caminhos é outra técnica utilizada para tentar melhorar os resultados do GRASP tradicional.

Esta técnica foi proposta originalmente por Glover em [11]. O uso dela em um procedimento GRASP foi proposto primeiramente por Laguna e Martí [20] e posteriormente houve diversas melhorias e extensões.

As duas estratégias básicas de utilização de reconexão por caminhos no GRASP são:

- Reconexão por caminhos aplicada como uma estratégia de pós-otimização entre todas as soluções do conjunto elite. Onde conjunto elite é um conjunto das k melhores soluções geradas até o momento pelo algoritmo.
- Reconexão por caminhos aplicada como uma estratégia de intensificação a cada ótimo local obtido após a fase de busca local.

O algoritmo de reconexão por caminhos consiste em escolher um par de soluções (solução base, solução alvo) dentro do conjunto elite e realizar uma série de movimentos para transformar a solução base na solução alvo. Cada solução encontrada durante esta trajetória é avaliada e incluída no conjunto elite e/ou marcada como melhor solução global dependendo do seu valor.

Além das estratégias básicas existem outras sugestões utilizadas para tentar melhorar o desempenho do algoritmo. Algumas delas são:

- Não aplicar a reconexão por caminhos em todas iterações GRASP, mas apenas periodicamente.

- Explorar duas estratégias potencialmente diferentes, fazer com que a solução base chegue até a solução alvo e o caminho contrário, ou seja, gerar soluções variando a solução alvo até chegar na solução base.
- Não percorrer a trajetória completa da solução base até a solução alvo, mas apenas parte dela (reconexão por caminhos truncada).

Ao se utilizar estas alternativas é necessário avaliar a relação entre o tempo computacional gasto e a qualidade da solução encontrada.

O algoritmo 6 ilustra o paseudocódigo do grasp com reconexão por caminhos.

Algoritmo 6 Grasp com reconexão por Caminhos

```
1:  ler entrada
2:  for k =1 até número_de_iterações do
3:     solução1 = construção(semente)
4:     solução2 = busca local (solução1)
5:     guarda_melhor_solução ( solução2 , melhor_solução)
6:     armazena_solução ( vetor_de_melhores_soluçoes, melhor_solução)
7:     solução_base := escolhe_solução_base( vetor_de_melhores_soluçoes)
8:     solução_alvo := escolhe_solução_alvo( vetor_de_melhores_soluçoes)
9:     Solução3 = reconexao_por_caminhos(solução_base,solução_alvo)
10:    guarda melhor solução ( solução3 , melhor_solução)
11:    armazena solução ( vetor_de_melhores_soluçoes, melhor_solução)
12: end for
```

Capítulo 5

Implementações GRASP para o PFCM

5.1 Introdução

Este capítulo apresenta cinco versões do algoritmo GRASP para resolver o problema de formação de células de manufatura. Cada versão inclui uma nova técnica para buscar melhores resultados em relação a um algoritmo GRASP tradicional.

O primeiro algoritmo a ser desenvolvido utilizará os conceitos básicos da metaheurística GRASP, ou seja, é um algoritmo iterativo onde cada iteração é constituída de duas fases: construção e busca local. A fase de construção é responsável por criar uma solução inicial viável e a fase de busca local percorre uma vizinhança desta solução inicial tentando melhorá-la. A melhor solução encontrada entre todas as iterações é fornecida como resultado do GRASP.

A quantidade de iterações que o algoritmo irá executar é um parâmetro do algoritmo GRASP.

O segundo algoritmo a ser apresentado utilizará a técnica de filtro na fase de construção que consiste em gerar várias soluções na fase de construção antes de passar para a fase de busca local da mesma iteração.

O terceiro algoritmo utilizará conceitos de mineração de dados a cada k iterações. A mineração irá retornar uma nova solução através de uma "combinação"das melhores soluções encontradas até aquele ponto.

O quarto algoritmo utilizará a técnica de reconexão por caminhos. Com esta técnica novas soluções intermediárias serão geradas alterando uma solução base, elemento a elemento, até chegar a uma solução alvo.

E o quinto e último algoritmo proposto utiliza todas as técnicas anteriores para tentar obter melhores resultados.

Todos os algoritmos propostos não utilizam valores fixos para o número de clusters que uma solução conterá e para os fatores de aleatoriedade da LCR do GRASP. Estes valores serão variados dinamicamente e aleatoriamente.

Nos tópicos a seguir são descritas todas as particularidades dos algoritmos propostos e a descrição dos mesmos.

5.2 Informação de Entrada sobre o PFCM

Todos os algoritmos recebem como única informação do problema, uma matriz binária de ordem M X P utilizada para relacionar as máquinas e as peças. Onde M representa o número de máquinas e P o número de peças. Nesta matriz um elemento $a_{m,p}$ = 1 indica que a máquina m realiza uma operação sobre a peça p e um elemento $a_{m,p}$ = 0 indica o contrário.

A tabela 5.1 ilustra um sistema de manufatura com 8 máquinas e 12 peças representado por uma matriz binária relacionando máquinas e peças. As máquinas estão representadas pelas linhas da matriz numeradas de 0 a 7 e as peças estão representadas pelas colunas da matriz numeradas de 0 a 11.

	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
M0	1	1	1	1	-	-	-	-	-	-	-	-
M1	1	-	1	1	1	1	1	-	-	1	-	-
M2	-	-	1	1	1	1	1	1	1	-	-	-
M3	-	-	-	-	-	1	1	1	1	1	-	-
M4	-	-	-	-	-	-	1	1	1	1	-	-
M5	-	-	-	-	-	-	1	1	1	-	1	-
M6	-	-	-	-	-	-	-	-	-	-	1	1
M7	-	-	-	-	-	-	-	-	-	-	1	1

Tabela 5.1: Exemplo de matriz binária de entrada para o PFCM

5.3 Formato da Solução

A solução do problema de formação de células de manufatura obtida pelos algoritmos propostos consiste em determinar quantos clusters terá a solução gerada, em quais clusters as máquinas e peças do problema serão alocadas e qual a qualidade desta solução (desempenho). Sendo que o objetivo do algoritmo é encontrar uma solução com o maior desempenho(problema de maximização).

A solução será representada por uma estrutura que conterá o número de clusters que formam a solução, o valor de desempenho e a associação de máquinas e peças em clusters. Esta associção é representada por um vetor onde as primeiras posições representam as máquinas, as últimas posições representam as peças e o conteúdo do vetor identifica em qual cluster esta máquina ou peça foi alocada. A tabela 5.2 ilustra este vetor.

máquinas				peças					
M0	M1	M2	M3	P0	P1	P2	P3	P4	P5
Ck	Ck	Ck	Ck	Ck	Ck	Ck	Ck	Ck	Ck

Tabela 5.2: Formato da Solução

5.4 Definição do Número de Clusters

A quantidade de clusters de uma solução é solicitada como entrada em muitos algoritmos, porém é muito difícil para o projetista do sistema de manufatura descobrir previamente qual é o melhor valor a ser utilizado. Por este motivo os algoritmos apresentados neste trabalho variam o número de clusters dinamicamente durante as iterações para que o próprio algoritmo descubra qual é o melhor valor a ser utilizado.

O número de clusters de uma solução possui um limite inferior e um limite superior. Uma solução válida deve possuir pelo menos dois clusters porque uma solução com apenas um cluster consistiria em agrupar todas as máquinas e peças em um mesmo cluster. E para evitar que sejam geradas soluções com clusters unitários, ou seja, com apenas uma máquina ou uma peça o limite superior para o número de cluster é igual à metade do número de máquinas, já que o mesmo é sempre menor do que o número de peças.

Com conhecimento dos limites inferior e superior, o algoritmo varia o número de clusters a ser utilizado durante a execução de maneira dinâmica e aleatória. O algoritmo realiza esta escolha da seguinte forma:

- Monta um vetor que contêm as possibilidades de escolha para o número de clusters, limitado pelos limites inferior e superior.
- Escolhe um valor do vetor aleatoriamente.
- Marca este valor como utilizado.
- Utiliza este valor como número de clusters até que seja utilizado em combinação com todos os fatores de aleatoriedade da LCR.
- Escolhe outro valor do vetor e repete os passos anteriores.
- Quando todos os valores do vetor forem utilizados, o vetor é reinicializado e todos os passos repetidos até o fim do algoritmo.

5.5 Medida de Desempenho Utilizada

Os algoritmos utilizaram a medida de desempenho Eficácia de Agrupamento definida no capítulo 2.3 para avaliar cada solução gerada. A razão para esta escolha, é que os algoritmos da literatura que servirão para comparação também utilizam esta medida.

Esta medida é calculada pela equação 5.1.

$$T = \frac{e - e_0}{e + e_v} \tag{5.1}$$

onde
e = número de operações (1‘s) na matriz
e_0 = número de elementos inter-clusters (1´s fora dos clusters)
e_v = número de elementos iguais a zero dentro dos clusters
0 <= T <= 1

5.6 Validade da Solução

Algumas das soluções que podem ser geradas pelo algoritmo são consideradas inválidas e são descartadas por não resolverem o PFCM.

As soluções inválidas são as seguintes:

- São consideradas inválidas as soluções que contiverem clusters que possuam apenas uma máquina ou uma peça. Isto porque o objetivo do algoritmo é agrupar máquinas e peças. Se uma máquina está sozinha significa que ela não atende nenhuma peça em comum com outra máquina e se uma peça está sozinha significa que as máquinas que a atendem não atendem nenhuma outra. Em ambos os casos, as máquinas e peças envolvidas não deveriam fazer parte da entrada do PFCM por não poderem ser agrupadas.
- São consideradas inválidas as soluções que contiverem clusters em que uma máquina não atenda nenhuma peça do seu cluster. Um cluster que contenha uma máquina que não atenda nenhuma peça significa que este cluster possui uma máquina sem utilidade, ou seja, a máquina foi colocada em um local onde não será utilizada. Em uma solução eficiente cada máquina deve ser colocada nos clusters onde atendam o maior número de peças possível, otimizando assim a utilização destas máquinas.
- São consideradas inválidas as soluções que contiverem clusters que contenha uma peça que não é atendida por nenhuma máquina do seu cluster. Pois uma peça colocada em um cluster em que não exista nenhuma máquina que a atenda não poderá ser produzida neste cluster. Uma solução eficiente deve conter clusters onde as peças sejam colocadas junto com as máquinas que a produzem para otimizar a fabricação da mesma.

5.7 Montagem da LCR

Como descrito no Capítulo 4, a lista LCR é utilizada na fase de construção do algorimo GRASP para auxiliar na escolha do próximo elemento a fazer parte da solução inicial. No caso do PFCM, um elemento da solução representa o número (identificação) do cluster que a máquina ou peça será alocada, desta forma a LCR conterá clusters candidatos.

A fase de construção do GRASP para o PFCM é realizada em duas etapas. A primeira etapa aloca as máquinas nos clusters e a segunda etapa aloca as peças nos clusters. Como os clusters são escolhidos para as máquinas e para as peças em momentos diferentes, são criadas duas listas LCR diferentes. E para possibilitar maior liberdade ao projetista do sistema de manufatura foram utilizados dois fatores de aleatoriedade diferentes. O fator α para escolher os clusters para as peças e o fator β para escolher os clusters para as máquinas.

A lista LCR utilizada na escolha de clusters para as peças é montada através da equação:
LCR = { t $\in$ C | g(t) >= (tmin + (1 - α) (tmax - tmin)) }

A lista LCR utilizada na escolha de clusters para as máquinas é montada através da equação:
LCR = { t $\in$ C | g(t) >= (tmin + (1 - β) (tmax - tmin)) }
onde
t - um elemento (um cluster) do conjunto C.

C - Conjunto de todos os elementos (clusters) candidatos.
g(t) - função gulosa que avalia o elemento (cluster).
α ou β - fator que define a aleatoriedade da escolha de um elemento na LCR, $\alpha \in$ [0,1], $\beta \in$ [0,1]
tmin - menor valor encontrado pela função gulosa.
tmax - maior valor encontrado pela função gulosa.

A função gulosa define o quanto a solução sendo criada irá melhorar(aumentar seu valor) com a escolha de um determinado cluster. E esta função é calculada de maneira diferente para a escolha do cluster para as máquinas e para as peças.

A função gulosa utilizada para avaliar um cluster t para uma máquina i é a seguinte:

$$g(t) = \frac{Pit}{Pi + Pt - Pit} \tag{5.2}$$

onde
Pit - número de peças atendidas em comum entre a máquina i e todas as máquinas já pertencentes ao cluster t.
Pi - número de peças atendidas pela máquina i
Pt - número de peças atendidas por todas as máquinas já pertencentes ao cluster t

A função gulosa utilizada para avaliar um cluster t para uma peça i é o seguinte:

$$g(t) = \frac{Mit}{Mt} \tag{5.3}$$

onde
Mit - número de máquinas pertencentes ao cluster t que atendem a peça i.
Mt - número de máquinas pertencentes ao cluster t

5.8 Variação dos fatores de aleatoriedade

A escolha dos parâmetros α e β é crítica e fundamental para o bom desempenho do algoritmo GRASP como já mencionado no Capítulo 4. Por este motivo os algoritmos propostos não utilizam valores fixos para estes parâmetros, ao invés disto os algoritmos variam os valores dos mesmos durante a execução para encontrar os melhores parâmetros, ou seja, os que produzem soluções de melhor qualidade.

Em [9], Tom Feo e Maurício Resende mostraram que os melhores valores para α varia entre 0,5 e 0,9 para um problema de minimização e os melhores resultados obtidos são com α = 0,8. Porém como o PFCM é um problema de maximização, os valores de α e β utilizados foram entre 0,1 e 0,5 (escolha baseada em testes preliminares).

Os algoritmos alteram o valor de α e de β da seguinte maneira:

- O valor de α e de β são utilizados primeiramente como 0,1.

- Em cada iteração o valor de α ou de β é incrementado em 0,1 (até chegar em 0,5) formando combinações ainda não utilizadas.
- Se a solução encontrada na iteração com estes valores for a melhor solução até o momento, os valores de α e de β são guardados.
- Quando todas as combinações forem utilizadas, os valores de α e de β voltam para 0,1 e os outros passos são repetidos.
- Estes passos são executados até que sejam executadas x % do número total de iterações GRASP.
- Após este limite de iterações o algoritmo utiliza os valores que forneceram as melhores soluções.

5.9 Algoritmos propostos

Abaixo segue a descrição de cada um dos algoritmos propostos neste trabalho.

5.9.1 GRASP1 (Tradicional)

A heurística GRASP1 foi implementada utilizando os métodos tradicionais de um algoritmo GRASP.

As demais heurísticas possuem a mesma estrutura deste, porém elas possuem módulos adicionais para melhorar o desempenho do mesmo.

Abaixo segue a descrição de cada etapa do algoritmo GRASP1.

5.9.1.1 Parâmetros de Entrada

Este algoritmo recebe por parâmetro o nome do arquivo que contém a matriz binária com informações do PFCM e o nome do arquivo de configuração com as seguintes informações:

- Número máximo de iterações que o algoritmo irá executar;
- Valor que indica o percentual (0 a 1) de iterações que serve como limite para começar a usar os melhores valores encontrados até o momento para os parâmetros alfa, beta e número de clusters. Se este valor for igual a 1 o algoritmo irá variar estes parâmetros até o fim, se for igual a 0 o algoritmo irá utilizar os melhores valores a partir da segunda iteração e se for igual a 0,3 por exemplo o algoritmo irá utilizar os melhores valores a partir da iteração igual a 30 % das iterações totais mais uma.

5.9.1.2 Estrutura do Programa Principal

O programa principal do GRASP1 é ilutrado através do algoritmo 7.

Algoritmo 7 Main GRASP1

```
1: Recebe parâmetros de entrada
2: Inicializa estruturas e variáveis
3: k := 1
4: while k < número_de_iterações do
5:    Varia os valores de α, β e número_de_clusters
6:    solução1 = construção
7:    valida_solucao(solução1)
8:    if solução1 inválida then
9:       volta para início do while
10:   end if
11:   avalia_solucao(solução1)
12:   if solução1.desempenho > melhor_desempenho then
13:      melhor_solução := solução1
14:      melhor_desempenho := solução1.desempenho
15:   end if
16:   solução2 = busca_local(solução1)
17:   avalia_solucao(solução2)
18:   if solução2.desempenho > melhor_desempenho then
19:      melhor_solução := solução2
20:      melhor_desempenho := solução2.desempenho
21:   end if
22:   Incrementa k
23: end while
24: return melhor_solução
```

5.9.1.3 Fase de Construção

A fase de construção cria a solução inicial a ser utilizada pelo algoritmo na iteração corrente. Esta solução é criada através das seguintes etapas:

Criação do vetor de similaridades para auxiliar na construção

O algoritmo utiliza similaridade de Jaccard para atribuir um valor de similaridade para cada par de máquinas do problema e cria um vetor ordenado crescentemente com estes pares.

Formação dos clusters iniciais contendo as primeiras máquinas.

Um par de máquinas é escolhido aleatoriamente entre os 30 % pares menos similares do vetor de similaridades. Uma máquina do par é alocada no cluster 0 e a outra máquina no cluster 1.

O algoritmo aloca uma máquina para cada cluster faltante verificando a similaridade entre cada máquina candidata e as máquinas já alocadas. A máquina que possuir a menor similaridade iniciará o novo cluster e este procedimento é realizado até que todos os clusters sejam preenchidos.

Cada cluster recebe mais uma máquina para evitar a criação de soluções com apenas uma máquina que são consideradas inválidas. A máquina que possuir a maior similaridade com a máquina já alocada no cluster é a escolhida para ser alocada ao mesmo.

Alocação inicial das peças

O algoritmo aloca duas peças em cada cluster para que não seja gerada nenhuma solução com um cluster que contenha apenas uma peça, que seria uma solução inválida.

O algoritmo escolhe para os clusters duas peças que sejam atendidas pelas duas máquinas contidas nos mesmos.

Alocação das máquinas restantes

As máquinas restantes são alocadas em clusters uma a uma utilizando a lista LCR. Para cada máquina a lista LCR é preenchida com os melhores clusters segundo a função gulosa. Após a criação da LCR, um cluster é escolhido aleatoriamente dentro desta lista para que a máquina seja alocada.

Alocação das peças restantes

Assim como na alocação das máquinas as peças restantes são alocadas uma a uma utilizando a lista LCR. Para cada peça a lista LCR é preenchida com os melhores clusters segundo a função gulosa. Após a criação da LCR, um cluster é escolhido aleatoriamente dentro desta lista para que a peça seja alocada.

Fim da construção

Ao final destas etapas a construção está finalizada e a solução inicial criada.

O algoritmo 8 ilustra o pseudocódigo da fase de construção.

Algoritmo 8 Procedure Construção

```
1: cria_vetor_similaridades(vet_similiar)
2: calcula_similaridades(vet_similiar)
3: ordena_vetor_similaridades(vet_similiar)
4: forma_clusters_iniciais(maquinas,vet_similiar,solucao)
5: aloca_pecas_iniciais(maquinas,vet_similiar,solucao)
6: for M=0 até número de máquinas do
7:    monta_LCR_maquina(LCR_maq)
8:    cluster_escolhido := Escolhe_cluster_aleatoriamente(LCR_maq)
9:    solucao[M] := cluster_escolhido
10: end for
11: for P=0 até número de peças do
12:    monta_LCR_pecas(LCR_peca)
13:    cluster_escolhido := Escolhe_cluster_aleatoriamente(LCR_peca)
14:    solucao[P] := cluster_escolhido
15: end for
16: return solucao
```

5.9.1.4 Fase de Busca Local

A fase de busca local consiste em tentar encontrar uma solução melhor do que a solução inicial gerada na fase de construção. Para isto são geradas soluções vizinhas da solução inicial, isto é, soluções baseadas na solução inicial.

O algoritmo primeiramente realoca as peças da solução inicial em novos clusters, se a nova formação conseguir uma solução melhor do que a inicial, esta solução é armazenada, senão a busca local é terminada.

A solução gerada pela realocação das peças é alterada através da realocação das máquinas em novos clusters, se a nova formação conseguir uma solução melhor, esta solução é armazenada, senão a busca local é terminada.

Estas alterações continuam, sempre com base na solução anterior, até que a busca local não consiga gerar uma solução melhor que a anterior. Neste momento a busca local termina e retorna a melhor solução encontrada. Caso a busca local não gere nenhuma solução melhor do que a solução inicial, ela mesma é retornada como melhor solução encontrada.

A realocação das peças é feita calculando um coeficiente para saber em qual cluster a peça produz um resultado melhor.

O coeficiente é calculado através da fórmula 5.4.

$$Coef = Um\_dentro - Um\_fora - Zeros \tag{5.4}$$

onde
Um_dentro - quantidade de operações (1‘s) dentro do cluster com a presença desta peça.
Um_fora - quantidade de operações (1‘s) fora do cluster com a presença desta peça.
Zeros - quantidade de falta de operações (0´s) dentro do cluster com a presença desta peça.

Prioriza-se a alteração que fornecer o maior valor de coeficiente. Se houver dois clusters com mesmo coeficiente o algoritmo utiliza o coeficiente de desempate para escolher o cluster. Este coeficiente de desempate é calculado através da fórmula 5.5

$$Coef\_desempate = \frac{Um\_dentro}{nMaquinas} \tag{5.5}$$

onde
Um_dentro - quantidade de operações (1‘s) dentro do cluster com a presença desta peça.
nMaquinas - quantidade de máquinas no cluster.

Com estes coeficientes calculados, o algoritmo aloca a peça no cluster com maior coeficiente.

A realocação das máquinas é realizada da mesma forma que a realocação de peças, ou seja, o algoritmo calcula um coeficiente principal e um coeficiente de desempate para saber em qual cluster a máquina produz um resultado melhor e a aloca neste cluster.

O coeficiente principal é calculado da mesma forma que o calculado para as peças e o coeficiente de desempate é calculado pela equação 5.6.

$$lCoef_desempate = lUm\_dentro/nPecas \tag{5.6}$$

onde:
lUm_dentro - quantidade de operações (1‘s) dentro do cluster com a presença desta máquina.
nPecas - quantidade de peças no cluster.

O algoritmo 9 ilustra o pseudocódigo da busca local.

Algoritmo 9 Procedure Busca Local

```
1: solucao_base := solucao_inicial
2: while melhores soluções do
3:    solucao_temp := realoca_pecas(solucao_base)
4:    avalia_solucao(solucao_temp)
5:    if solucao_temp.desempenho > melhor_desempenho then
6:       melhor_desempenho := solucao_temp.desempenho
7:       solucao_base := solucao_temp
8:    else
9:       return solucao_base
10:   end if
11:   solucao_temp := realoca_maquinas(solucao_base)
12:   avalia_solucao(solucao_temp)
13:   if solucao_temp.desempenho > melhor_desempenho then
14:      melhor_desempenho := solucao_temp.desempenho
15:      solucao_base := solucao_temp
16:   else
17:      return solucao_base
18:   end if
19: end while
20: return solucao_base
```

5.9.2 GRASP_FILTRO

O algoritmo GRASP_FILTRO foi implementado utilizando o algoritmo GRASP1 como base. Nele foi adicionada a técnica de filtro que consiste em gerar várias soluções na fase de construção antes de passar para a fase de busca local da mesma iteração. Diferente do GRASP tradicional que gera apenas uma solução na fase de construção, a cada iteração GRASP.

Abaixo segue a descrição detalhada do algoritmo.

5.9.2.1 Parâmetros de Entrada

Este algoritmo recebe por parâmetro o nome do arquivo que contém a matriz binária com informações do PFCM e o nome do arquivo de configuração com as seguintes informações:

- Número de iterações que o algoritmo irá executar;
- Valor que indica o percentual (de 0 a 1) de iterações que serve como limite para começar a usar os melhores valores encontrados até o momento dos parâmetros alfa, beta e número de clusters;
- Número de soluções gerada na fase de construção antes de retornar a melhor;
- Tamanho do vetor de soluções iniciais;

- Número de repetições de solução inicial toleradas antes de terminar o algoritmo;

5.9.2.2 Funcionamento do Filtro

O algoritmo utiliza um *loop* no programa principal para executar a fase de construção n vezes para encontrar a melhor solução inicial.

Como a fase de construção irá retornar sempre a melhor solução encontrada, a chance desta solução se repetir durante as iterações é muito grande e uma solução inicial repetida dificilmente irá gerar alguma solução melhor do que outra já gerada. Para evitar este problema o algoritmo realiza um controle de soluções iniciais através de um vetor para guardar as soluções iniciais já utilizadas e não repeti-las.

Se não for encontrada nenhuma solução nova na fase de construção, o algoritmo executa a fase de construção novamente. O algoritmo executa estas repetições até alcançar o limite tolerado passado por parâmetro.

A quantidade de soluções iniciais guardadas no vetor é passada por parâmetro. Quando uma solução vai ser inserida, o algoritmo verifica se ela já existe e executa o seguinte procedimento: Caso já exista não faz nada. Caso contrário, coloca no fim do vetor se ele não estiver cheio; ou coloca no lugar da solução mais antiga (inserida primeiro) se o vetor estiver cheio.

O algoritmo 10 ilustra o pseudocódigo do filtro.

Algoritmo 10 Procedure Filtro

```
1: Recebe parâmetros
2: filtro := 0;
3: while filtro < parâmetro_repeticões_filtro do
4:    solução1 = construção
5:    valida_solucao(solução1)
6:    if solução1 inválida then
7:       volta para início do while
8:    end if
9:    avalia_solucao(solução1)
10:   verifica_vetor_solucoes_iniciais(solução1,vetor_solucoes_iniciais)
11:   if encontrar a solução then
12:      Não utiliza
13:   else
14:      if solução1.desempenho > melhor_desempenho then
15:         melhor_solucao := solução1
16:         melhor_desempenho := solução1.desempenho
17:      end if
18:   end if
19:   Incrementa filtro
20: end while
21: if não encontrou nenhuma solução nova then
22:    if tentativas > tolerância_repetições then
23:       Finaliza o algoritmo
24:    else
25:       Incrementa as tentativas
26:       Passa para a próxima iteração (a iteração não é incrementada)
27:    end if
28: end if
29: preenche_vetor_solucoes_iniciais(solução1,vetor_solucoes_iniciais)
```

5.9.3 GRASP_DM

O algoritmo GRASP_DM foi implementado utilizando o algoritmo GRASP_FILTRO como base. Nele foi adicionada a técnica de mineração de dados (*data mining*) para tentar alcançar melhores resultados.

A mineração irá retornar uma nova solução através de uma "combinação"das melhores soluções encontradas até aquele ponto.

A mineração é executada a cada k iterações, onde k é um parâmetro de entrada. Após cada execução da mineração o algoritmo executa a busca local em cima do resultado obtido pela mineração. Após todas as iterações, o algoritmo ainda executa a mineração e a busca local mais uma vez.

Uma descrição mais detalhada do algoritmo é apresentada mais adiante.

5.9.3.1 Parâmetros de Entrada

Este algoritmo recebe por parâmetro o nome do arquivo que contém a matriz binária com informações do PFCM e o nome do arquivo de configuração com as seguintes informações:

- Número de iterações que o algoritmo irá executar;
- Intervalo (número de iterações) de execução da mineração de dados;
- Valor que indica o percentual (de 0 a 1) de iterações que serve como limite para começar a usar os melhores valores encontrados até o momento dos parâmetros alfa, beta e número de clusters;
- Freqüência mínima para um cluster ser alocado em uma máquina ou peça durante a mineração;
- Percentual mínimo para uma solução ser inserida no vetor de melhores soluções;
- Número de soluções gerada na fase de construção antes de retornar a melhor;
- Número de repetições de mineração para um grupo de soluções;
- Tamanho do vetor de melhores soluções;
- Parâmetro que determina se a mineração é ou não gulosa;
- Parâmetro que define se a mineração será realizada;
- Tamanho do vetor de soluções iniciais;
- Número de repetições de solução inicial toleradas antes de terminar o algoritmo.

5.9.3.2 Funcionamento da Mineração de Dados

A busca através de mineração de dados é realizada a cada k iterações, onde o valor de k é passado como parâmetro de entrada para o algoritmo.

Toda vez que o algoritmo executar k iterações ele executará a mineração de dados sobre as melhores soluções encontradas até o momento, porém a mineração não será realizada se o vetor de melhores soluções não tiver sido modificado desde a última mineração. Uma solução é inserida no vetor de melhores soluções apenas se ela for maior ou igual a um percentual da melhor solução já inserida no mesmo.

A mineração será executada nas melhores soluções agrupadas pela quantidade de clusters de cada solução, porém só serão usados os grupos em que o número de soluções pertencentes a ele for maior ou igual a 30 % das melhores soluções. Por exemplo, se o vetor de melhores soluções possuir 21 soluções onde 10 utilizam 4 clusters, 2 utilizam 3 clusters e 9 utilizam 2 clusters então o algoritmo irá minerar apenas o grupo de 4 clusters(10 soluções) e o de 2 clusters(9 clusters) porque possuem mais soluções do que o mínimo de 30 % de 21 soluções (7 soluções).

Para cada grupo de soluções selecionado o algoritmo irá montar uma nova solução. Esta solução será a melhor entre as criadas dentro de um *loop* de n repetições cujo valor foi passado por parâmetro.

Para iniciar uma solução o algoritmo cria uma matriz de máquina/peça X cluster que contém a quantidade de vezes que cada máquina/peça aparece nas soluções utilizando um determinado cluster.

Com base nesta matriz o algoritmo coloca duas máquinas e uma ou duas peças em cada cluster. As duas máquinas escolhidas serão as máquinas que aparecem alocadas neste cluster no maior número de soluções; e a peça selecionada será uma peça atendida por estas máquinas e também alocada neste cluster no maior número de soluções. Se não houver uma peça atendida pelas duas máquinas então serão escolhidas duas peças (uma atendida por cada máquina) para serem alocadas no cluster.

A forma de escolha do cluster que as outras máquinas e peças serão alocadas dependerá se a mineração é gulosa (determinado por parâmetro de entrada) ou não. Se for gulosa, o cluster escolhido será o cluster que foi utilizado mais vezes nas soluções do grupo. Se não for, o algoritmo utiliza a técnica de roleta para escolher o cluster.

Com a técnica de roleta, a escolha de um cluster é realizada aleatoriamente dando maior vantagem para os clusters mais utilizados. Porém a máquina/peça será alocada no cluster escolhido apenas se a percentagem de vezes que este cluster é utilizado nas soluções nesta máquina/peça for maior ou igual a freqüência mínima necessária (passada por parâmetro). Se não for, a máquina ou peça não será alocada em nenhum cluster.

Cada máquina/peça que não foi alocada, será alocada no cluster que tiver maior similaridade. Cada máquina restante será alocada no cluster que possuir a maior quantidade de máquinas que atendam as mesmas peças que ela e cada peça será alocada no cluster que possuir o maior número de máquinas que a atendem.

Após gerar a solução para cada grupo de soluções, a mineração retorna para o algoritmo a melhor solução encontrada entre elas .

O algoritmo 11 ilustra o pseudocódigo do funcionamento da mineração de dados.

Algoritmo 11 Procedure GRASP_DM

```
1: cria_vetor_qtd_solucoes_cluster(vetor_qtd_solucoes_cluster)
2: for grupos:=0 até qtd_grupo_clusters do
3:    if  vetor_qtd_solucoes_cluster[grupos] < 30 % melhores_soluções then
4:       passa para o próximo grupo de clusters
5:    end if
6:    rep := 0
7:    while  rep < repeticoes_mineração do
8:       inicia_nova_solução(grupos,vetor_qtd_solucoes_cluster[grupos],novas_soluções[grupos])
9:       completa_nova_solução(novas_soluções[grupos])
10:      avalia_nova_solução(novas_soluções[grupos])
11:      if  novas_soluções[grupos].desempenho > melhor_desempenho  then
12:         solução_escolhida_cluster := novas_soluções[grupos]
13:         melhor_desempenho := novas_soluções[grupos].desempenho
14:      end if
15:      rep := rep + 1
16:   end while
17:   novas_soluções[grupos] := solução_escolhida_cluster
18: end for
19: melhor_desempenho := 0
20: for grupos:=0 até qtd_grupo_clusters do
21:   if  vetor_qtd_solucoes_cluster[grupos] < 30 % melhores_soluções then
22:      passa para o próximo grupo de clusters
23:   end if
24:   if novas_soluções[grupos].desempenho > melhor_desempenho then
25:      solução_escolhida_final := novas_soluções[grupos]
26:      melhor_desempenho := novas_soluções[grupos].desempenho
27:   end if
28: end for
29: return  solução_escolhida_final
```

A procedure inicia_nova_solução, inicia a solução alocando as máquinas e peças nos clusters. O funcionamento desta procedure é descrita no algoritmo 12.

Algoritmo 12 Procedure inicia_nova_solução(n_clusters_grupo,qtd_solucoes, nova_solucao)

```
1: cria_vetor_qtd_itens_cluster(qtd_itens_cluster)
2: for c:=0 até n_clusters_grupo do
3:     alocacao_inicial(solução,qtd_itens_cluster, c)
4: end for
5: for elemento := 0 até n_maquinas + n_peças do
6:     alocacao_outros_elementos(solução,qtd_itens_cluster, c)
7: end for
```

A procedure alocacao_inicial inicia a nova solução alocando duas máquinas e uma peça em cada cluster. O pseudocódigo desta procedure é descrita no algoritmo 14.

A Procedure alocacao_outros_elementos aloca as outras máquinas e peças nos clusters. O pseudocódigo desta procedure está no algoritmo 13.

Algoritmo 13 Procedure alocacao_outros_elementos(solução,qtd_itens_cluster, cluster c)

```
1: if mineração == gulosa then
2:     for c:=0 até n_clusters_grupo do
3:         if qtd_itens_cluster[c][elemento] > mais_clusters then
4:             mais_clusters := qtd_itens_cluster[c][elemento]
5:             cluster_escolhido := c
6:         end if
7:     end for
8: else
9:     mínimo := 0
10:    for c:=0 até n_clusters_grupo do
11:        intervalo[c] := [ mínimo , mínimo + qtd_itens_cluster[c][elemento] ]
12:    end for
13:    num_escolhido := escolha_aleatoria(qtd_solucoes)
14:    cluster_escolhido := C onde num_escolhido está em intervalo[c]
15: end if
16: if qtd_itens_cluster[c][elemento] > frequencia_mínima then
17:    solução[elemento] := cluster_escolhido
18: end if
```

Algoritmo 14 Procedure alocacao_inicial(solução,qtd_itens_cluster,cluster c)

```
1:  maior_qtd1 := maior_qtd2 := 0
2:  for maq := 0 até n_maquinas do
3:     if  qtd_itens_cluster[c][maq] > maior_qtd1  then
4:        if  maior_qtd1 > maior_qtd2 then
5:           maior_qtd2 := qtd_itens_cluster[c][maq]
6:           maq2 := maq
7:        else
8:           maior_qtd1 := qtd_itens_cluster[c][maq]
9:           maq1 := maq
10:       end if
11:    else
12:       if  qtd_itens_cluster[c][maq] > maior_qtd2 then
13:          maior_qtd2 := qtd_itens_cluster[c][maq]
14:          maq2 := maq
15:       end if
16:    end if
17:    solução[maq1] := c
18:    solução[maq2] := c
19: end for
20: for peça := 0 até n_peças do
21:    if peça é atendida por maq1 then
22:       if qtd_itens_cluster[c][peça] > maior_peca1 then
23:          maior_peca1 := qtd_itens_cluster[c][peça]
24:          peca1 := peça
25:       end if
26:    end if
27:    if peça é atendida por maq2 then
28:       if qtd_itens_cluster[c][peça] > maior_peca2 then
29:          maior_peca2 := qtd_itens_cluster[c][peça]
30:          peca2 := peça
31:       end if
32:    end if
33: end for
34: solução[peca1] := c
35: solução[peca2] := c
```

A procedure completa_nova_solução aloca cada máquina e peça restante, que não foi alocada devido a frequência. As máquinas são alocadas no cluster que possuir a maior quantidade de máquinas que atendam as mesmas peças que ela e as peças são alocadas no cluster que possuir o maior número de máquinas que as atendam. O pseudocódigo desta procedure está no algoritmo 15.

Algoritmo 15 Procedure completa_nova_solução(nova_solucao)

```
1: for elemento := 0 até n_maquinas + n_peças do
2:    if  elemento < n_maquinas then
3:       cluster_escolhido := busca_cluster_mais_similar_maq;
4:    else
5:       cluster_escolhido := busca_cluster_mais_similar_peca;
6:    end if
      nova_solucao[elemento] := cluster_escolhido
7: end for
```

5.9.4 GRASP_PR

O algoritmo GRASP_PR foi implementado utilizando o algoritmo GRASP_DM como base. Ele utiliza a técnica de reconexão por caminhos (*path relinking*) para encontrar novas soluções. O algoritmo GRASP_DM foi usado como base porque já havia sido implementado nele funções e estruturas básicas que também serão utilizados pelo GRASP_PR.

A reconexão por caminhos é executada em todas as iterações desde que o vetor de melhores soluções possua mais de uma solução.

Segue a descrição detalhada do algoritmo.

5.9.4.1 Parâmetros de Entrada

Este algoritmo recebe por parâmetro o nome do arquivo que contém a matriz binária com informações do PFCM e o nome do arquivo de configuração com as seguintes informações:

- Número de iterações que o algoritmo irá executar;
- Intervalo (número de iterações) de execução da mineração de dados;
- Valor que indica o percentual (de 0 a 1) de iterações que serve como limite para começar a usar os melhores valores encontrados até o momento dos parâmetros alfa, beta e número de clusters;
- Freqüência mínima para um cluster ser alocado em uma máquina ou peça durante a mineração;
- Percentual mínimo de diferença entre as soluções do vetor;
- Número de soluções gerada na fase de construção antes de retornar a melhor;
- Número de repetições de mineração para um grupo de soluções;

- Tamanho do vetor de melhores soluções;
- Tamanho do vetor de soluções iniciais;
- Número de repetições de solução inicial toleradas antes de terminar o algoritmo.

5.9.4.2 Funcionamento da Reconexão por Caminhos

A busca através de reconexão por caminhos é realizada em toda iteração desde que o parâmetro que define a sua execução esteja ativado e que o vetor de melhores soluções possua mais de uma solução.

A busca é realizada alterando uma solução base elemento a elemento até chegar a uma solução alvo. Este processo é realizado da seguinte forma:

1. O algoritmo utiliza como solução base a solução gerada no passo anterior.
2. A solução alvo será escolhida no vetor de melhores soluções entre as soluções que contenham a mesma quantidade de clusters da solução base. A solução escolhida será a que possuir o maior número de elementos diferentes da solução base.
3. O algoritmo gera soluções intermediárias alterando a solução base elemento a elemento. São geradas soluções intermediárias até encontrar a solução alvo.
4. Cada solução intermediária é gerada criando soluções temporárias alterando cada elemento da solução base (um de cada vez), validando e avaliando cada solução temporária obtida. A melhor solução encontrada será a solução intermediária.
5. O algoritmo tenta inserir esta solução no vetor de melhores soluções e verifica se é a melhor solução encontrada até o momento.
6. Se a solução intermediária gerada for igual à solução alvo o algoritmo termina, se não for, é gerada uma nova solução intermediária utilizando a anterior como solução base.

Abaixo segue um exemplo do funcionamento da busca por reconexão por caminhos:

Seja a solução B a solução base e a solução A a solução alvo:

Solução Base								
M0	M1	M2	M3	P0	P1	P2	P3	P4
0	1	1	0	0	1	0	1	1
Solução Alvo								
M0	M1	M2	M3	P0	P1	P2	P3	P4
1	0	0	1	1	0	0	0	1

Para obter a primeira solução intermediárias são geradas soluções temporárias. Cada solução temporária será formada através da troca de um elemento da solução base que estava diferente da solução alvo. Desta forma são geradas as 7 soluções a seguir:

Solução 1								
M0	M1	M2	M3	P0	P1	P2	P3	P4
1	1	1	0	0	1	0	1	1
Solução 2								
M0	M1	M2	M3	P0	P1	P2	P3	P4
0	0	1	0	0	1	0	1	1
Solução 3								
M0	M1	M2	M3	P0	P1	P2	P3	P4
0	1	0	0	0	1	0	1	1
Solução 4								
M0	M1	M2	M3	P0	P1	P2	P3	P4
0	1	1	1	0	1	0	1	1
Solução 5								
M0	M1	M2	M3	P0	P1	P2	P3	P4
0	1	1	0	1	1	0	1	1
Solução 6								
M0	M1	M2	M3	P0	P1	P2	P3	P4
0	1	1	0	0	0	0	1	1
Solução 7								
M0	M1	M2	M3	P0	P1	P2	P3	P4
0	1	1	0	0	1	0	0	1

Digamos que a solução válida com melhor desempenho seja a solução 5, então o algoritmo tenta inserir a solução 5 no vetor de melhores soluções, verifica se é a melhor solução já encontrada e a utiliza como solução base para gerar a próxima solução intermediária.

Solução 5 - Base								
M0	M1	M2	M3	P0	P1	P2	P3	P4
0	1	1	0	1	1	0	1	1
Solução Alvo								
M0	M1	M2	M3	P0	P1	P2	P3	P4
1	0	0	1	1	0	0	0	1

Sendo assim são geradas as soluções:

Solução 1								
M0	M1	M2	M3	P0	P1	P2	P3	P4
1	1	1	0	1	1	0	1	1
Solução 2								
M0	M1	M2	M3	P0	P1	P2	P3	P4
0	0	1	0	1	1	0	1	1
Solução 3								
M0	M1	M2	M3	P0	P1	P2	P3	P4
0	1	0	0	1	1	0	1	1
Solução 4								
M0	M1	M2	M3	P0	P1	P2	P3	P4
0	1	1	1	1	1	0	1	1
Solução 5								
M0	M1	M2	M3	P0	P1	P2	P3	P4
0	1	1	0	1	0	0	1	1
Solução 6								
M0	M1	M2	M3	P0	P1	P2	P3	P4
0	1	1	0	1	1	0	0	1

Digamos que a solução válida com melhor desempenho gerada neste passo seja a solução 4, então o algoritmo tenta inserir a solução 4 no vetor de melhores soluções, verifica se é a melhor solução já encontrada e a utiliza como solução base para gerar a próxima solução intermediária.

Estes passos são repetidos até que a solução alvo seja gerada.

O algoritmo 16 ilustra o pseudocódigo da reconexão por caminhos.

Algoritmo 16 Grasp com reconexão por Caminhos

```
1: solução_base := solução_atual
2: solução_alvo := escolhe_sol_alvo(vetor_melhores_sol,solução_base)
3: melhor_solução_intermediaria := solução_base
4: while melhor_solução_intermediaria ≠ solução_alvo do
5:    solucao_intermediaria := VAZIO
6:    solução_parcial := melhor_solução_intermediaria
7:    gera nova solução intermediária
8:    while elemento < n_maquinas + n_peças do
9:       if solução_parcial[elemento] ≠ solução_alvo[elemento] then
10:         solução_parcial[elemento] := solução_alvo[elemento]
11:      end if
12:      if solução_parcial.desempenho > solucao_intermediaria.desempenho then
13:         solucao_intermediaria := solução_parcial
14:      end if
15:   end while
16:   avalia_solução(solucao_intermediaria)
17:   preenche_vetor_melhores_soluções(solucao_intermediaria)
18:   if solucao_intermediaria.desempenho > solucao_final.desempenho then
19:      solucao_final := solucao_intermediaria
20:   end if
21:   melhor_solucao_intermediaria := solucao_intermediaria
22: end while
23: return solucao_final
```

5.9.5 GRASP_DM_PR

O algoritmo GRASP_DM_PR é a combinação do algoritmo GRASP_DM e o algoritmo GRASP_PR, ou seja, ele utiliza as técnicas de mineração de dados e de reconexão por caminhos.

A mineração de dados é executada a cada k iterações e a reconexão por caminhos é executada em todas as iterações.

O usuário define através de parâmetros de entrada se o algoritmo deve executar somente a mineração de dados, somente a reconexão por caminhos, ambas as técnicas ou nenhuma delas.

5.9.5.1 Parâmetros de Entrada

Este algoritmo recebe por parâmetro o nome do arquivo que contém a matriz binária com informações do PFCM e o nome do arquivo de configuração com as seguintes informações:

- Número de iterações que o algoritmo irá executar;
- Intervalo (número de iterações) para execução da mineração de dados;
- Valor que indica o percentual (de 0 a 1) de iterações que serve como limite para começar a usar os melhores valores encontrados até o momento dos parâmetros alfa, beta e número de clusters;

- Freqüência mínima para um cluster ser alocado em uma máquina ou peça durante a mineração;
- Percentual mínimo de diferença entre as soluções do vetor;
- Número de soluções gerada na fase de construção antes de retornar a melhor;
- Número de repetições de mineração para um grupo de soluções;
- Tamanho do vetor de melhores soluções;
- Parâmetro que determina se a mineração é ou não gulosa;
- Parâmetro que define se a busca por mineração de dados será executada;
- Parâmetro que define se a busca por reconexão por caminhos será executada;
- Tamanho do vetor de soluções iniciais;
- Número de repetições de solução inicial toleradas antes de terminar o algoritmo.

5.9.5.2 Funcionameto do GRASP_DM_PR

O algoritmo GRASP_DM_PR executa os seguintes passos:

1. Executa a fase de construção utilizando filtro.
2. Executa a busca local a partir da melhor solução inicial gerada.
3. Se a mineração estiver ativada e se o algoritmo estiver em uma iteração divísel pelo intervalo para execução da mineração de dados então executa a mineração.
4. Executa busca local sobre a solução gerada pela mineração de dados.
5. Executa a reconexão por caminhos.
6. Após todas as iterações o algoritmo executa mais uma vez a mineração de dados e a busca local.

O pseudocódigo ilustrando estes passos está descrito no algoritmo 17.

Algoritmo 17 Main GRASP_DM_PR

```
1:  Recebe parâmetros de entrada k := 0
2:  while  k < número_de_iterações do
3:     while  filtro < parâmetro_repeticões_filtro do
4:        solução1 = construção
5:        avalia_solucao(solução1)
6:        if solução1.desempenho > solucao_inicial.desempenho then
7:           solucao_inicial := solução1
8:        end if
9:     end while
10:    Executa_buscas(vetor_Melhores_solucoes,solução_inicial,melhor_solução)
11:    Incrementa k
12: end while
13: if  busca_por_mineração = ativada  then
14:    solucao_dm := realiza_busca_mineracao(vetor_melhores_solucoes)
15:    solução_local = busca_local(solucao_dm)
16:    preenche_vetor_melhores_solucoes(vetor_Melhores_solucoes,solução_local)
17:    if solução_local.desempenho > melhor_solução.desempenho then
18:       melhor_solução := solução_local
19:    end if
20: end if
21: return  melhor_solução
```

A procedure Executa_buscas está descrita no algoritmo 18

Algoritmo 18 Executa_buscas(vetor_Melhores_solucoes,solução_inicial,melhor_solução)

```
1: solução_local = busca_local(solução_inicial)
2: avalia_solucao(solução_local)
3: preenche_vetor_melhores_solucoes(vetor_Melhores_solucoes,solução_local)
4: if solução_local.desempenho > melhor_solução.desempenho then
5:    melhor_solução := solução_local
6: end if
7: if  busca_por_mineração = ativada  then
8:    if  k é divisível por n_iterações_mineração  then
9:       solucao_dm := realiza_busca_mineracao(vetor_melhores_solucoes)
10:      solução_local = busca_local(solucao_dm)
11:      preenche_vetor_melhores_solucoes(vetor_Melhores_solucoes,solução_local)
12:      if solução_local.desempenho > melhor_solução.desempenho then
13:         melhor_solução := solução_local
14:      end if
15:   end if
16: end if
17: if  busca_por_reconexão = ativada  then
18:   solucao_pr := realiza_busca_reconexao(vetor_melhores_solucoes)
19:   if solução_pr.desempenho > melhor_solução.desempenho then
20:      melhor_solução := solução_pr
21:   end if
22: end if
```

5.9.6 Mapeamento de técnicas dos Algoritmos Propostos

A tabela 5.3 mostra as técnicas utilizadas por cada um dos algoritmos propostos.

Algoritmo x Técnica	Construção	Busca Local	Filtro	Mineração de dados	Reconexão por caminhos
GRASP1	S	S	N	N	N
GRASP_FILTRO	S	S	S	N	N
GRASP_DM	S	S	S	S	N
GRASP_PR	S	S	S	N	S
GRASP_DM_PR	S	S	S	S	S

Tabela 5.3: Mapeamento de Técnicas e Algoritmos Propostos.

Capítulo 6

Testes Realizados e Resultados Obtidos

6.1 Introdução

Os algoritmos propostos nestes trabalhos foram avaliados utilizando 36 instâncias da literatura e os resultados obtidos foram comparados com os resultados de outros algoritmos também disponíveis na literatura.

As instâncias utilizadas estão listadas na tabela 6.1 e disponíveis em [12].

Instância	Arquivo	Maqs	Peças	Fonte
1	King_Nakornchai.dat	5	7	King and Nakornchai (1982)
2	Waghodecar_Sahu.dat	5	7	Waghodekar and Sahu (1984)
3	Seiffodini.dat	5	18	Seiffodini (1989)
4	Kusiak.dat	6	8	Kusiak (1992)
5	Kusiak_Chow.dat	7	11	Kusiak and Chow (1987)
6	Boctor.dat	7	11	Boctor (1991)
7	Seiffodini_Wolfe.dat	8	12	Seiffodini and Wolfe (1986)
8	Chandran_Rajago.dat	8	20	Chandrasekharan and Rajagopalan (1986a)
9	Chandra_Rajagob.dat	8	20	Chandrasekharan and Rajagopalan (1986b)
10	Mosier_Taube.dat	10	10	Mosier and Taube (1985a)
11	Chan_Milner.dat	10	15	Chan and Milner (1982)
12	Askin_Subramanian.dat	14	23	Askin and Subrammania (1987)
13	Stanfel.dat	14	24	Stanfel (1985)
14	McCormick.dat	16	24	McCormick et al. (1972)
15	Srinivasan.dat	16	30	Srinivasan et al. (1990)
16	King.dat	16	43	King (1980)
17	Burbidge.dat	20	35	Burbidge (1969)
18	Carrie.dat	18	24	Carrie (1973)
19	Mosier_Taubeb.dat	20	20	Mosier and Taube (1985b)
20	Kumar.dat	20	23	Kumar et al. (1986)
21	CarrieB.dat	20	35	Carrie (1973)

Instância	Arquivo	Maqs	Peças	Fonte
22	Boe_Cheng.dat	20	35	Boe and Cheng (1991)
23	Chandra_Raja2440a.dat	24	40	Chandrasekaran and Rajagopalan (1989) - 1
24	Chandra_Raja2440b.dat	24	40	Chandrasekaran and Rajagopalan (1989) - 2
25	Chandra_Raja2440c.dat	24	40	Chandrasekaran and Rajagopalan (1989) - 3
26	Chandra_Raja2440d.dat	24	40	Chandrasekaran and Rajagopalan (1989) - 4
27	Chandra_Raja2440e.dat	24	40	Chandrasekaran and Rajagopalan (1989) - 5
28	Chandra_Raja2440f.dat	24	40	Chandrasekaran and Rajagopalan (1989) - 6
29	McCormick2727.dat	27	27	McCormick et al. (1972)
30	Carrie2846.dat	28	46	Carrie (1973)
31	Kumar_Vannelli.dat	30	41	Kumar and Vannelli (1987)
32	Stanfel3050.dat	30	50	Stanfel (1985)
33	Stanfel3050b.dat	30	50	Stanfel (1985)
34	King_Nakornchai3090.dat	30	90	King and Nakornchai (1982)
35	McCormick3753.dat	37	53	McCormick et al. (1972)
36	Chandra_Rajago40100.dat	40	100	Chandrasekaran and Rajagopalan (1987)

Tabela 6.1: Instâncias utilizadas nos testes.

Os resultados obtidos pelos algoritmos propostos foram comparados com os algoritmos listados na tabela 6.2.

Código	Nome	Fonte
HGA	algoritmo genético hibrido	Resende e Gonçalves (2002)
Zodiac	Zodiac	Srinivasan e Narendan (1991)
Grafics	Grafics	Srinivasan e Narendan (1991)
CA	algortimo de clusterização	Srinivasan (1991)
GA	algoritmo genético	Cheng (1998)
AG-JCK	algoritmo genético	Joines
AE	algoritmo evolutivo	Áthila Rocha Trindade (2004)

Tabela 6.2: Algoritmos da literatura.

6.2 Resultados dos Algoritmos propostos

Os testes dos algoritmos propostos foram realizados em uma máquina com processador AMD Athlon XP 1600 com 1.4 GHz e 256 MB de RAM utilizando sistema operacional Windows XP Professional versão 2002 com service pack 2.

Cada algoritmo proposto foi executado dez vezes com cada instância e aqui serão mostrados os melhores resultados e as médias encontradas.

Os resultados serão apresentados em tabelas que conterão as seguintes informações:

Campo	Descrição
Inst	Identificação da instância que foi processada
Aptidão	Desempenho da melhor solução encontrada. A aptidão varia de 0 a 100 e quanto maior o valor melhor é a solução.
Aptidão Média	Média das aptidões das soluções encontradas em cada uma das execuções do algoritmo.
# clusters	Número de cluster´s utilizado na melhor solução.
Tempo Total médio	O tempo médio que o algoritmo levou para terminar de executar todas as iterações.

Tabela 6.3: Campos das tabelas de resultados.

6.2.1 Resultados do Algoritmo GRASP1

O algoritmo GRASP1 foi executado 10 vezes para cada instância em três configurações diferentes. A diferença de uma configuração para outra foi o percentual utilizado para indicar em qual iteração o algoritmo passaria a utilizar os melhores valores de alfa, beta e número de cluster.

Estes parâmetros guardam os valores utilizados na iteração que obteve o melhor resultado até o momento. Desta forma, pode-se concluir que quanto mais tarde usarmos os valores encontrados, melhores eles serão. Por este motivo e baseado em avaliações preliminares foram utilizados os valores 0,8 ; 0,9 e 1 para este parâmetro.

A tabela 6.4 mostra a comparação entre os resultados dos testes com as três configurações. Os melhores resultados deste conjunto de testes, estão em negrito.

		Config 1 (0,8)	Config 2 (0,9)	Config 3 (1)
Instância	Melhor Aptidão	Aptidão	Aptidão	Aptidão
1	73,68	**73,68**	**73,68**	**73,68**
2	60,87	**60,87**	**60,87**	**60,87**
3	77,36	**77,36**	**77,36**	**77,36**
4	76,92	**76,92**	**76,92**	**76,92**
5	53,13	**53,13**	**53,13**	**53,13**
6	70,37	**70,37**	**70,37**	**70,37**
7	68,29	**68,29**	**68,29**	**68,29**
8	58,41	**58,41**	58,33	58,33
9	85,25	**85,25**	**85,25**	**85,25**
10	70,59	**70,59**	**70,59**	**70,59**
11	92	**92**	**92**	**92**
12	67,65	**67,65**	**67,65**	67,12
13	69,33	**69,33**	**69,33**	**69,33**
14	50	49,55	49,11	**50**
15	67,83	**67,83**	**67,83**	**67,83**
16	54,86	**54,86**	**54,86**	**54,86**
17	75,71	**75,71**	**75,71**	**75,71**
18	51,22	50	47,62	**51,22**
19	42,55	41,67	**42,55**	42
20	49,65	**49,65**	**49,65**	**49,65**
21	76,14	**76,14**	**76,14**	**76,14**
22	57,95	57,71	57,71	**57,95**
23	100	**100**	**100**	**100**
24	85,11	**85,11**	**85,11**	**85,11**
25	73,51	**73,51**	**73,51**	**73,51**
26	51,57	51,28	51,22	**51,57**
27	46,62	46,53	46,25	**46,62**
28	44,59	44,25	**44,59**	44,25
29	54,27	**54,27**	**54,27**	**54,27**
30	43,28	43,06	**43,28**	43
31	57,06	**57,06**	**57,06**	**57,06**
32	59,66	**59,66**	**59,66**	58,7

		Config 1 (0,8)	Config 2 (0,9)	Config 3 (1)
Instância	Melhor Aptidão	Aptidão	Aptidão	Aptidão
33	50,25	**50,25**	**50,25**	**50,25**
34	40,29	40,05	39,89	**40,29**
35	55,08	**55,08**	**55,08**	**55,08**
36	84,03	**84,03**	**84,03**	**84,03**
Total de melhores soluções		27	29	**30**

Tabela 6.4: Comparação dos resultados do algoritmo GRASP1.

Analisando a tabela 6.4, pode-se notar que as três configurações obtiveram resultados semelhantes. A configuração que obteve o melhor resultado no maior número de instâncias foi a **configuração 3** que utilizou o percentual **1,0** para dizer quando o algoritmo deveria começar a utilizar os melhores valores dos parâmetros dinâmicos.

A tabela 6.5 detalha o resultado dos testes com a configuração 3.

Configuração de parâmetros 3

- Número de iterações que o algoritmo irá executar: 500.
- percentual para começar a utilizar os melhores valores dos parâmetros gerados dinamicamente: **1,0**. Desta forma o algoritmo não irá utilizar os melhores valores dos parâmetros dinâmicos e ficará variando eles até o final.

Inst	Aptidão	Aptidão Média	# Clusters	Tempo Total Médio
1	73,68	73,68	2	0,06
2	60,87	60,87	2	0,04
3	77,36	77,36	2	0,05
4	76,92	76,92	2	0,05
5	53,13	51,83	3	0,05
6	70,37	70,37	3	0,05
7	68,29	68,29	3	0,08
8	58,33	57,25	2	0,10
9	85,25	85,25	3	0,10
10	70,59	70,59	3	0,12
11	92	92	3	0,13
12	67,12	66,94	5	0,29

Inst	Aptidão	Aptidão Média	# Clusters	Tempo Total Médio
13	69,33	68,42	5	0,28
14	50	47,43	6	0,45
15	67,83	65,32	4	0,36
16	54,86	54,82	5	0,47
17	75,71	75,71	4	0,72
18	51,22	47,48	5	0,95
19	42	41,2	4	1,63
20	49,65	47,44	5	1,13
21	76,14	76,14	4	0,77
22	57,95	57,11	6	0,75
23	100	100	7	1,31
24	85,11	85,11	7	1,39
25	73,51	73,51	7	1,64
26	51,57	50,33	9	1,64
27	46,62	45,78	11	1,39
28	44,25	43,92	8	1,26
29	54,27	54,1	4	3,03
30	43	42,22	6	4,48
31	57,06	55,75	9	3,92
32	58,7	56,57	11	3,54
33	50,25	49,59	11	3,97
34	40,29	39,28	11	4,70
35	55,08	53,86	2	10,01
36	84,03	84,03	10	13,32

Tabela 6.5: Resultados do algoritmo GRASP1 com configuração 3.

6.2.2 Resultados do Algoritmo GRASP_FILTRO

O algoritmo GRASP_FILTRO também foi executado 10 vezes para cada instância em três configurações diferentes. A diferença de uma configuração para outra foi o percentual utilizado para indicar em qual iteração o algoritmo passaria a utilizar os melhores valores de alfa, beta e número de cluster. Assim como no caso do GRASP 1, os valores utilizados foram 0,8 ; 0,9 e 1.

A tabela 6.6 mostra a comparação entre os resultados dos testes com as três configurações. Os melhores resultados deste conjunto de testes, estão em negrito.

		Config 1 (0,8)	Config 2 (0,9)	Config 3 (1)
Instância	Melhor Aptidão	Aptidão	Aptidão	Aptidão
1	73,68	**73,68**	**73,68**	**73,68**
2	60,87	**60,87**	**60,87**	**60,87**
3	77,36	**77,36**	**77,36**	**77,36**
4	76,92	**76,92**	**76,92**	**76,92**
5	53,13	**53,13**	**53,13**	**53,13**
6	70,37	**70,37**	**70,37**	**70,37**
7	68,29	**68,29**	**68,29**	**68,29**
8	58,41	**58,41**	58,33	**58,41**
9	85,25	**85,25**	**85,25**	**85,25**
10	70,59	**70,59**	**70,59**	**70,59**
11	92	**92**	**92**	**92**
12	67,65	67,12	67,12	**67,65**
13	69,33	**69,33**	**69,33**	**69,33**
14	50	**50**	49,11	49,11
15	67,83	**67,83**	**67,83**	**67,83**
16	54,86	**54,86**	**54,86**	**54,86**
17	75,71	**75,71**	**75,71**	**75,71**
18	51,22	50	48,55	**51,22**
19	42,76	**42,76**	**42,76**	**42,76**
20	49,65	**49,65**	**49,65**	**49,65**
21	76,14	**76,14**	**76,14**	**76,14**
22	57,95	**57,95**	**57,95**	**57,95**
23	100	**100**	**100**	**100**
24	85,11	**85,11**	**85,11**	**85,11**
25	73,51	**73,51**	**73,51**	**73,51**
26	51,57	**51,57**	51,28	51,28
27	46,98	**46,98**	**46,98**	**46,98**
28	44,59	**44,59**	44,25	44,3
29	54,27	**54,27**	**54,27**	**54,27**
30	43,7	43,28	**43,7**	43,42
31	57,06	**57,06**	**57,06**	**57,06**
32	59,66	**59,66**	58,76	**59,66**
33	50,25	**50,25**	**50,25**	**50,25**
34	40,98	**40,98**	40,69	40,29

		Config 1 (0,8)	Config 2 (0,9)	Config 3 (1)
Instância	Melhor Aptidão	Aptidão	Aptidão	Aptidão
35	55,67	**55,67**	**55,67**	**55,67**
36	84,03	**84,03**	**84,03**	**84,03**
Total de melhores soluções		**33**	28	31

Tabela 6.6: Comparação dos resultados do algoritmo GRASP_FILTRO.

Analisando a tabela 6.6, pode-se notar que todas as configurações obtiveram resultados semelhantes, ou seja, pelos resultados empíricos aqui efetuados tanto no GRASP1 como no GRASP_FILTRO, se observou que a forma de atualizar os parâmetros não provoca mudanças drásticas nos resultados finais da heurística GRASP.

A configuração que obteve o melhor resultado no maior número de instâncias foi a configuração 1 que utilizou o percentual 0,8 para dizer quando o algoritmo deveria começar a utilizar os melhores valores dos parâmetros dinâmicos.

A tabela 6.7 detalha os resultados dos testes com a configuração 1.

Configuração de parâmetros 1

- Número de iterações que o algoritmo irá executar: 500.
- Percentual para começar a utilizar os melhores valores dos parâmetros gerados dinamicamente: **0,8**. Desta forma o algoritmo começará a utilizar os melhores valores dos parâmetros dinâmicos após a iteração 500 * 0,8 = 400.
- Número de soluções gerada na fase de construção antes de retornar a melhor: 20
- Tamanho do vetor de soluções iniciais: 20
- Número de repetições de solução inicial toleradas antes de terminar o algoritmo: 10

Inst	Aptidão	Aptidão Média	# Clusters	Tempo Total Médio
1	73,68	73,68	2	0,03
2	60,87	60,87	2	0,02
3	77,36	77,36	2	0,03
4	76,92	76,92	2	0,03
5	53,13	50,52	3	0,06
6	70,37	70,37	3	0,02
7	68,29	66	3	0,06
8	58,41	57,33	2	0,06
9	85,25	85,25	3	0,07
10	70,59	70,59	3	0,06
11	92	92	3	0,07
12	67,12	66,81	5	0,20
13	69,33	68,65	5	0,24
14	50	47,84	6	2,01
15	67,83	67,25	4	3,74
16	54,86	54,82	5	6,04
17	75,71	75,71	4	4,27
18	50	47,41	5	3,96
19	42,76	42,4	4	30,48
20	49,65	49,65	5	24,15
21	76,14	76,14	4	5,88
22	57,95	57,73	6	13,95
23	100	100	7	7,68
24	85,11	85,11	7	30,28
25	73,51	73,51	7	28,81
26	51,57	51,34	9	25,57
27	46,98	46,66	11	23,80
28	44,59	44,34	10	20,70
29	54,27	54,27	4	42,55
30	43,28	42,76	8	79,22
31	57,06	57,06	9	62,91
32	59,66	58,63	12	59,31

Inst	Aptidão	Aptidão Média	# Clusters	Tempo Total Médio
33	50,25	50,25	11	112,17
34	40,98	40,09	10	306,75
35	55,67	55,3	3	193,72
36	84,03	84,03	10	272,39

Tabela 6.7: Resultados do algoritmo GRASP_FILTRO com configuração 1.

Comparação Parcial dos algoritmos propostos

A tabela 6.8 compara os resultados parciais dos algoritmos propostos neste trabalho e testados até este momento.

Os melhores resultados estão em negrito.

	GRASP1	GRASP_FILTRO
Instância	Aptidão	Aptidão
1	**73,68**	**73,68**
2	**60,87**	**60,87**
3	**77,36**	**77,36**
4	**76,92**	**76,92**
5	**53,13**	**53,13**
6	**70,37**	**70,37**
7	**68,29**	**68,29**
8	**58,41**	**58,41**
9	**85,25**	**85,25**
10	**70,59**	**70,59**
11	**92**	**92**
12	**67,65**	**67,65**
13	**69,33**	**69,33**
14	**50**	**50**
15	**67,83**	**67,83**
16	**54,86**	**54,86**
17	**75,71**	**75,71**
18	**51,22**	**51,22**
19	42,55	**42,76**
20	**49,65**	**49,65**
21	**76,14**	**76,14**
22	**57,95**	**57,95**
23	**100**	**100**

	GRASP1	GRASP_FILTRO
Instância	Aptidão	Aptidão
24	**85,11**	**85,11**
25	**73,51**	**73,51**
26	**51,57**	**51,57**
27	46,62	**46,98**
28	**44,59**	**44,59**
29	**54,27**	**54,27**
30	43,28	**43,7**
31	**57,06**	**57,06**
32	**59,66**	**59,66**
33	**50,25**	**50,25**
34	40,29	**40,98**
35	55,08	**55,67**
36	**84,03**	**84,03**
Total	31	**36**

Tabela 6.8: Comparação do GRASP_FILTRO com os algoritmos propostos.

Analisando a tabela 6.8, pode-se notar que o algoritmo GRASP_FILTRO melhorou o resultado do GRASP1 em 5 instâncias, ou seja, a inclusão do filtro na fase de construção do GRASP tende a melhorar o seu desempenho final, em relação a qualidade das soluções geradas. No entanto a versão com filtro exige tempo computacional bem maior como ilustram as tabelas 6.5 e 6.7

6.2.3 Resultados do Algoritmo GRASP_DM

O algoritmo GRASP_DM foi submetido a três baterias de testes:

- A primeira serviu para avaliar qual o melhor valor para o parâmetro que define o momento de utilizar os melhores valores de alfa, beta e número de clusters.
- A segunda serviu para avaliar qual o melhor valor para o parâmetro que define a freqüência mínima. Este parâmetro é utilizado no momento de montar uma nova solução através da mineração. Uma máquina ou peça será preenchida com um determinado cluster apenas se a quantidade de vezes que este cluster é utilizado por esta maquina / peça nas melhores soluções satisfizer a freqüência mínima.
- A terceira serviu para avaliar o melhor valor para o parâmetro que define o limite para inserir uma solução no vetor de melhores soluções. Uma solução será inserida apenas se ela for maior ou igual a max_vetor % da melhor solução já existente no vetor.

O algoritmo recebe também outros parâmetros além destes, porém eles são considerados de influência menor e este trabalho não apresenta resultados variando-os. Os valores para eles foram definidos em testes preliminares e a partir do que entendemos ser o melhor valor para eles.

Este algoritmo foi executado primeiramente sem utilizar o controle de soluções inicias apresentado no algoritmo GRASP_FILTRO, que faz com que o algoritmo não utilize a mesma solução inicial mais de uma vez. Depois foi executado utilizando este controle. Os resultados mostrados em cada bateria de testes são os resultados obtidos sem este controle.

Após a apresentação dos resultados das três baterias é mostrado um comparativo dos resultados com e sem controle de soluções inicias obtidos nos testes da terceira bateria, pois esta utiliza os melhores parâmetros definidos por todas as baterias de testes.

Em cada Bateria de testes o algoritmo foi executado 10 vezes para cada instância e a configuração das baterias 2 e 3 utilizaram os melhores valores definidos pelas baterias anteriores.

6.2.3.1 Resultados da Primeira Bateria - variando o limite para usar os melhores parâmetros dinâmicos.

Esta bateria define qual o melhor valor para o parâmetro que define o momento de utilizar os melhores valores de alfa, beta e número de clusters. Os valores utilizados foram os mesmos utilizados nos testes dos algoritmos anteriores: 0,8 ; 0,9 e 1.

O algoritmo foi executado 10 vezes para cada instância com cada uma das 3 configurações.

A tabela 6.9 mostra a comparação entre os resultados dos testes com as 3 configurações. Os melhores resultados deste conjunto de testes estão em negrito.

	Config 1 (0,8)	Config 2 (0,9)	Config 3 (1)
Instância	Aptidão	Aptidão	Aptidão
1	**73,68**	**73,68**	**73,68**
2	**60,87**	**60,87**	**60,87**
3	**77,36**	**77,36**	**77,36**
4	**76,92**	**76,92**	**76,92**
5	**53,13**	**53,13**	**53,13**
6	**70,37**	**70,37**	**70,37**
7	**68,29**	**68,29**	**68,29**
8	**58,41**	**58,41**	**58,41**
9	**85,25**	**85,25**	**85,25**
10	**70,59**	**70,59**	**70,59**
11	**92**	**92**	**92**
12	**67,65**	**67,65**	**67,65**
13	**69,33**	**69,33**	**69,33**
14	49,11	**50**	**50**
15	**67,83**	**67,83**	**67,83**
16	**54,86**	**54,86**	**54,86**

	Config 1 (0,8)	Config 2 (0,9)	Config 3 (1)
Instância	Aptidão	Aptidão	Aptidão
17	**75,71**	**75,71**	**75,71**
18	51,22	**51,64**	51,22
19	**42,76**	42,55	42,11
20	**49,65**	**49,65**	**49,65**
21	**76,14**	**76,14**	**76,14**
22	57,71	**57,95**	57,71
23	**100**	**100**	**100**
24	**85,11**	**85,11**	**85,11**
25	**73,51**	**73,51**	**73,51**
26	51,57	**51,88**	51,28
27	**46,98**	**46,98**	**46,98**
28	**44,87**	44,52	44,52
29	**54,27**	**54,27**	**54,27**
30	43,28	**43,42**	**43,42**
31	**57,06**	**57,06**	**57,06**
32	**58,76**	**58,76**	**58,76**
33	50,25	50,25	**50,51**
34	**41,4**	40,84	40,39
35	56,29	55,67	**56,62**
36	**84,03**	**84,03**	**84,03**
Total de melhores soluções	29	**31**	30

Tabela 6.9: Comparação dos Resultados das 3 configurações da bateria 1 do algoritmo GRASP DM.

Analisando a tabela 6.9, pode-se notar que todas as configurações obtiveram resultados semelhantes. A configuração que obteve o melhor resultado no maior número de instâncias foi a configuração 2 que utilizou o percentual 0,9 para dizer quando o algoritmo deveria começar a utilizar os melhores valores dos parâmetros dinâmicos.

6.2.3.2 Resultados da Segunda Bateria - variando a freqüência mínima.

Nesta bateria foram utilizadas 4 configurações diferentes variando a freqüência mínima. Este parâmetro é utilizado no momento de montar uma nova solução através da mineração. Uma máquina (peça) será preenchida com um determinado cluster apenas se a quantidade de vezes que esta maquina (peça) foi alocada neste cluster nas soluções mineradas satisfizer uma freqüência mínima passada como dado de entrada.

Se o valor deste parâmetro for 0 a nova solução será montada inteiramente pela mineração e se for igual a 1 a nova solução será montada inteiramente por uma nova fase de construção sem o uso da mineração. Como entendemos que estes valores extremos não são os mais indicados utilizamos os seguintes valores nos testes: 0,2 ; 0,4 ; 0,6 e 0,8 .

E como os resultados da primeira bateria de testes mostrou que o melhor valor para o parâmetro limite é 0,9; este valor foi utilizado nesta bateria em todas as configurações.

O algoritmo foi executado 10 vezes para cada instância com cada uma das 4 configurações.

A tabela 6.10 mostra a comparação entre os resultados dos testes com as 4 configurações. Os melhores resultados deste conjunto de teste estão em negrito.

	Config 1 (0,2)	Config 2 (0,4)	Config 3 (0,6)	Config 3 (0,8)
Instância	Aptidão	Aptidão	Aptidão	Aptidão
1	**73,68**	**73,68**	**73,68**	**73,68**
2	**60,87**	**60,87**	**60,87**	**60,87**
3	**77,36**	**77,36**	**77,36**	**77,36**
4	**76,92**	**76,92**	**76,92**	**76,92**
5	**53,13**	**53,13**	**53,13**	**53,13**
6	**70,37**	**70,37**	**70,37**	**70,37**
7	**68,29**	**68,29**	**68,29**	**68,29**
8	**58,41**	**58,41**	**58,41**	**58,41**
9	**85,25**	**85,25**	**85,25**	**85,25**
10	**70,59**	**70,59**	**70,59**	**70,59**
11	**92**	**92**	**92**	**92**
12	**67,65**	**67,65**	**67,65**	**67,65**
13	**69,33**	**69,33**	**69,33**	**69,33**
14	**50**	**50**	**50**	**50**
15	**67,83**	**67,83**	**67,83**	**67,83**
16	**54,86**	**54,86**	**54,86**	**54,86**
17	**75,71**	**75,71**	**75,71**	**75,71**
18	**51,64**	**51,64**	51,22	51,22
19	**42,74**	42,55	**42,74**	**42,74**
20	**49,65**	**49,65**	**49,65**	**49,65**
21	**76,14**	**76,14**	**76,14**	**76,14**
22	**57,95**	**57,95**	**57,71**	57,71
23	**100**	**100**	**100**	**100**
24	**85,11**	**85,11**	**85,11**	**85,11**
25	**73,51**	**73,51**	**73,51**	**73,51**
26	51,28	**51,88**	51,28	51,57
27	**46,98**	**46,98**	**46,98**	**46,98**
28	44,3	44,52	44,52	**44,87**
29	**54,27**	**54,27**	**54,27**	**54,27**
30	**43,42**	**43,42**	43,28	43,28
31	**57,06**	**57,06**	**57,06**	**57,06**
32	58,76	58,76	58,47	**59,32**
33	**50,25**	**50,25**	**50,25**	**50,25**

	Config 1 (0,2)	Config 2 (0,4)	Config 3 (0,6)	Config 3 (0,8)
Instância	Aptidão	Aptidão	Aptidão	Aptidão
34	40,27	**40,84**	40,57	40,29
35	**55,67**	**55,67**	**55,67**	**55,67**
36	**84,03**	**84,03**	**84,03**	**84,03**
Total de melhores soluções	32	**33**	29	31

Tabela 6.10: Comparação dos Resultados das 3 configurações da bateria 2 do algoritmo GRASP_DM.

Analisando os resultados da tabela 6.10 pode-se notar que todas as configurações obtiveram resultados semelhantes. A configuração que obteve o melhor resultado no maior número de instâncias foi a **configuração 2** que utilizou freqüência **0,4**.

6.2.3.3 Resultados da Terceira Bateria - variando percentual de diferença no vetor de melhores soluções.

Nesta bateria foram utilizadas 4 configurações distintas variando o limite para inserir uma solução no vetor de melhores soluções. Uma solução será inserida apenas se ela for maior ou igual a max_vetor % da melhor solução já existente no vetor.

Se este parâmetro for igual a 0 qualquer solução poderá ser inserida no vetor de melhores soluções, podendo influenciar negativamente a análise do vetor e se este parâmetro for igual a 1 o vetor de melhores soluções conterá apenas a melhor solução. Como entendemos que esses extremos não são eficazes, os valores utilizados para este parâmetro em cada configuração foram: 0,3 ; 0,5 ; 0,7 e 0,9 .

Foi utilizado os melhores resultados dos parâmetros testados nas baterias anteriores, ou seja, o parâmetro limite de 0,9 e freqüência 0,4.

O algoritmo foi executado 10 vezes para cada instância com cada uma das 4 configurações.

A tabela 6.11 mostra a comparação entre os resultados dos testes com as 4 configurações mostradas acima. Os melhores resultados estão em negrito.

	Config 1 (0,3)	Config 2 (0,5)	Config 3 (0,7)	Config 3 (0,9)
Instância	Aptidão	Aptidão	Aptidão	Aptidão
1	**73,68**	**73,68**	**73,68**	**73,68**
2	**60,87**	**60,87**	**60,87**	**60,87**
3	**77,36**	**77,36**	**77,36**	**77,36**
4	**76,92**	**76,92**	**76,92**	**76,92**
5	**53,13**	**53,13**	**53,13**	**53,13**
6	**70,37**	**70,37**	**70,37**	**70,37**
7	**68,29**	**68,29**	**68,29**	**68,29**
8	**58,72**	**58,72**	58,41	58,41
9	**85,25**	**85,25**	**85,25**	**85,25**
10	**70,59**	**70,59**	**70,59**	**70,59**
11	**92**	**92**	**92**	**92**
12	**67,65**	**67,65**	**67,65**	**67,65**
13	**69,33**	**69,33**	**69,33**	**69,33**
14	49,11	**50**	**50**	48,72
15	**67,83**	**67,83**	**67,83**	**67,83**
16	**54,86**	**54,86**	**54,86**	**54,86**
17	**75,71**	**75,71**	**75,71**	**75,71**
18	**51,64**	51,22	**51,64**	50
19	42,55	**42,96**	42,55	42,76
20	**49,65**	**49,65**	**49,65**	**49,65**
21	**76,14**	**76,14**	**76,14**	**76,14**
22	**57,95**	**57,95**	**57,95**	**57,95**
23	**100**	**100**	**100**	**100**
24	**85,11**	**85,11**	**85,11**	**85,11**
25	**73,51**	**73,51**	**73,51**	**73,51**
26	51,28	51,28	**51,88**	51,59
27	**46,98**	**46,98**	**46,98**	**46,98**
28	44,51	**44,87**	44,52	44,25
29	**54,27**	**54,27**	**54,27**	**54,27**
30	**43,58**	43,45	43,42	42,91
31	**57,06**	**57,06**	**57,06**	**57,06**
32	**58,76**	**58,76**	**58,76**	**58,76**
33	**50,51**	50,25	50,25	**50,51**

	Config 1 (0,3)	Config 2 (0,5)	Config 3 (0,7)	Config 3 (0,9)
Instância	Aptidão	Aptidão	Aptidão	Aptidão
34	40,94	**41,63**	40,84	41,39
35	55,96	**56,82**	55,67	56,75
36	**84,03**	**84,03**	**84,03**	**84,03**
Total de Melhores soluções	30	**32**	29	27

Tabela 6.11: Comparação dos Resultados das 4 configurações da bateria 3 do algoritmo GRASP_DM.

Analisando os resultados da tabela 6.11 pode-se notar novamente que todas as configurações obtiveram resultados semelhantes. A configuração que obteve o melhor resultado no maior número de instâncias foi a **configuração 2** que utilizou percentual para entrar no vetor igual a **0,5**.

6.2.3.4 Conclusões Parciais

Conclusões das 3 baterias de testes com o algoritmo GRASP_DM

A tabela 6.12 compara os melhores resultados das 3 baterias de testes realizados.

	Bateria 1 (limite)	Bateria 2 (freqüência)	Bateria 3 (Max vet)
Instância	Aptidão	Aptidão	Aptidão
1	**73,68**	**73,68**	**73,68**
2	**60,87**	**60,87**	**60,87**
3	**77,36**	**77,36**	**77,36**
4	**76,92**	**76,92**	**76,92**
5	**53,13**	**53,13**	**53,13**
6	**70,37**	**70,37**	**70,37**
7	**68,29**	**68,29**	**68,29**
8	**58,41**	**58,41**	**58,72**
9	**85,25**	**85,25**	**85,25**
10	**70,59**	**70,59**	**70,59**
11	**92**	**92**	**92**
12	**67,65**	**67,65**	**67,65**
13	**69,33**	**69,33**	**69,33**
14	**50**	**50**	**50**

	Bateria 1 (limite)	Bateria 2 (freqüência)	Bateria 3 (Max vet)
Instância	Aptidão	Aptidão	Aptidão
15	**67,83**	**67,83**	**67,83**
16	**54,86**	**54,86**	**54,86**
17	**75,71**	**75,71**	**75,71**
18	**51,64**	**51,64**	**51,64**
19	42,76	42,74	**42,96**
20	**49,65**	**49,65**	**49,65**
21	**76,14**	**76,14**	**76,14**
22	**57,95**	**57,95**	**57,95**
23	**100**	**100**	**100**
24	**85,11**	**85,11**	**85,11**
25	**73,51**	**73,51**	**73,51**
26	**51,88**	**51,88**	**51,88**
27	**46,98**	**46,98**	**46,98**
28	**44,87**	**44,87**	**44,87**
29	**54,27**	**54,27**	**54,27**
30	43,42	43,42	**43,58**
31	**57,06**	**57,06**	**57,06**
32	58,76	**59,32**	58,76
33	**50,51**	50,25	**50,51**
34	41,4	40,84	**41,63**
35	56,62	55,67	**56,82**
36	**84,03**	**84,03**	**84,03**
Total de Melhores soluções	30	30	**35**

Tabela 6.12: Comparação entre as 3 baterias de testes do GRASP_DM.

Como cada bateria utilizou os parâmetros definidos como melhores pela bateria anterior a última bateria obteve os melhores resultados em todas as instâncias, exceto na instância 32. Isto porque o melhor valor encontrado para esta instância foi utilizando freqüência 0,8 e a utilizada na terceira bateria foi freqüência 0,4.

Após as 3 baterias de testes realizadas e analisar os resultados parciais, pode-se concluir de forma empírica que a melhor configuração de parâmetros para estas instâncias noconjunto de testes realizados é a configuração dada a seguir:

- Número de iterações que o algoritmo irá executar: 500
- Intervalo (número de iterações) de execução da mineração de dados: 25
- Percentual para começar a utilizar os melhores valores dos parâmetros gerados dinamicamente: **0,9**. Desta forma o algoritmo começará a utilizar os melhores valores dos parâmetros dinâmicos após a iteração 500 * 0,9 = 450.
- Freqüência mínima para um cluster ser alocado em uma máquina ou peça durante a mineração: **0,4**

- Percentual mínimo para uma solução ser inserida no vetor de melhores soluções: **0,5**
- Número de soluções gerada na fase de construção antes de retornar a melhor: 20
- Número de repetições de mineração para um grupo de soluções: 10
- Tamanho do vetor de melhores soluções: 20
- Parâmetro que determina se a mineração é ou não gulosa: 0
- Parâmetro que define se a mineração será realizada: 1
- Parâmetro que define se reconexão por caminhos será realizado: 0
- Tamanho do vetor de soluções iniciais: 20
- Número de repetições de solução inicial toleradas antes de terminar o algoritmo: 10

Os melhores resultados obtidos pela versão com mineração de dados são mostrados na tabela 6.13, de forma mais detalhada.

Inst	Aptidão	Aptidão Média	# Clusters	Tempo Total Médio
1	73,68	73,68	2	0,35
2	60,87	60,87	2	0,36
3	77,36	77,36	2	0,82
4	76,92	76,92	2	0,79
5	53,13	53,13	3	0,88
6	70,37	70,37	3	0,68
7	68,29	68,29	3	1,25
8	58,72	58,05	2	1,29
9	85,25	85,25	3	1,35
10	70,59	70,59	3	1,93
11	92	92	3	1,78
12	67,65	67,23	6	5,06
13	69,33	69,33	5	4,92
14	50	48,61	6	7,86
15	67,83	67,83	4	6,40
16	54,86	54,86	5	7,45
17	75,71	75,71	4	12,30

Inst	Aptidão	Aptidão Média	# Clusters	Tempo Total Médio
18	51,22	48,98	5	19,16
19	42,96	42,05	5	30,67
20	49,65	49,2	5	24,79
21	76,14	76,14	4	12,75
22	57,95	57,11	6	18,89
23	100	100	7	28,40
24	85,11	85,11	7	23,92
25	73,51	73,51	7	30,39
26	51,28	51,27	9	30,04
27	46,98	46,74	11	30,94
28	44,87	44,29	10	25,13
29	54,27	54,27	4	45,42
30	43,45	42,78	8	96,05
31	57,06	57,06	9	71,07
32	58,76	58,34	12	182,97
33	50,25	50,25	11	70,70
34	41,63	40,33	11	150,05
35	56,82	55,29	3	170,88
36	84,03	84,03	10	680,52

Tabela 6.13: Resultados detalhados do algoritmo GRASP DM na bateria 3 com configuração 2.

Resultados obtidos com controle de soluções inicias

O algoritmo GRASP DM foi submetido as mesmas 3 baterias de testes apresentadas anteriormente utilizando o controle de soluções iniciais apresentado no algoritmo GRASP_FILTRO. A tabela 6.14 compara os resultados destes testes com os anteriores:

	Bateria 3 SEM CONTROLE DE SOL. INICIAIS	Bateria 3 COM CONTROLE DE SOL. INICIAIS
Instância	Aptidão	Aptidão
1	**73,68**	**73,68**
2	**60,87**	**60,87**
3	**77,36**	**77,36**
4	**76,92**	**76,92**
5	**53,13**	**53,13**
6	**70,37**	**70,37**
7	**68,29**	**68,29**
8	**58,72**	58,33

	Bateria 3 SEM CONTROLE DE SOL. INICIAIS	Bateria 3 COM CONTROLE DE SOL. INICIAIS
Instância	Aptidão	Aptidão
9	**85,25**	**85,25**
10	**70,59**	**70,59**
11	**92**	**92**
12	**67,65**	67,12
13	**69,33**	**69,33**
14	**50**	**50**
15	**67,83**	**67,83**
16	**54,86**	**54,86**
17	**75,71**	**75,71**
18	**51,64**	51,22
19	**42,96**	42,76
20	**49,65**	**49,65**
21	**76,14**	**76,14**
22	**57,95**	**57,95**
23	**100**	**100**
24	**85,11**	**85,11**
25	**73,51**	**73,51**
26	**51,88**	51,61
27	46,98	**47,06**
28	**44,87**	**44,87**
29	**54,27**	**54,27**
30	**43,58**	43,28
31	**57,06**	**57,06**
32	58,76	**59,66**
33	**50,51**	50,25
34	41,63	**42,71**
35	**56,82**	56,67
36	**84,03**	**84,03**
Total	**33**	28

Tabela 6.14: Comparação dos resultados do GRASP_DM com e sem controle de soluções iniciais.

Podemos notar através da tabela 6.14 que os resultados empíricos obtidos sem controle de soluções inicias são melhores do que os obtidos com este controle. O algoritmo sem controle obteve melhores resultados em 8 instâncias e com controle em 2 instâncias, empatando nas demais.

Comparação dos resultados com a literatura

A próxima tabela e gráfico comparam os melhores resultados obtidos pelo GRASP_DM e os resultados já existentes na literatura.

		Zodiac	Grafics	CA	GA	HGA	AG-JCK	AE	GRASP_DM	
Inst.	Melhor Valor Lit.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	%
1	73,68	**73,68**	**73,68**	-	-	**73,68**	**73,68**	**73,68**	**73,68**	100,00
2	62,5	56,52	60,87	-	68 (inv)	**62,5**	**62,5**	**62,5**	60,87	97,39
3	79,59	77,36	-	-	77,36	**79,59**	**79,59**	**79,59**	77,36	97,20
4	76,92	**76,92**	-	-	**76,92**	**76,92**	**76,92**	**76,92**	**76,92**	100,00
5	53,13	39,13	53,12	-	46,88	**53,13**	**53,13**	**53,13**	**53,13**	100,00
6	70,37	**70,37**	-	-	**70,37**	**70,37**	**70,37**	**70,37**	**70,37**	100,00
7	68,3	**68,3**	**68,3**	-	-	**68,3**	65,12	68,29	68,29	99,99
8	58,72	58,33	58,13	**58,72**	58,33	**58,72**	57,26	**58,72**	**58,72**	100,00
9	85,25	85,24	85,24	85,24	85,24	**85,25**	**85,25**	**85,25**	**85,25**	100,00
10	70,59	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**	100,00
11	92	**92**	**92**	-	**92**	**92**	**92**	**92**	**92**	100,00
12	69,86	64,36	64,36	64,36	-	**69,86**	**69,86**	**69,86**	67,65	96,84
13	69,33	65,55	65,55	-	67,44	**69,33**	67,05	**69,33**	**69,33**	100,00
14	51,96	32,09	45,52	48,7	-	**51,96**	47,69	**51,96**	50	96,23
15	67,83	**67,83**	**67,83**	**67,83**	-	**67,83**	**67,83**	**67,83**	**67,83**	100,00
16	54,86	53,76	54,39	54,44	53,89	**54,86**	50,88	**54,86**	**54,86**	100,00
17	75,71	-	-	-	-	-	**75,71**	**75,71**	**75,71**	100,00
18	54,46	41,84	48,91	44,2	-	**54,46**	51,24	**54,46**	51,64	94,82
19	42,96	21,63	38,26	-	37,12	**42,96**	41,14	**42,96**	**42,96**	100,00
20	49,65	38,66	49,36	43,01	46,62	**49,65**	44,57	**49,65**	**49,65**	100,00
21	76,22	75,14	75,14	75,14	75,28	76,14	**76,22**	76,14	76,14	99,90
22	58,07	51,13	-	-	55,14	**58,07**	56,52	58,06	57,95	99,79
23	100	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	100,00
24	85,11	**85,11**	**85,11**	**85,11**	**85,11**	**85,11**	69,59	**85,11**	**85,11**	100,00
25	73,51	**73,51**	**73,51**	**73,51**	73,03	**73,51**	**73,51**	**73,51**	**73,51**	100,00
26	51,97	20,42	43,27	51,81	49,37	51,85	50,3	**51,97**	51,88	99,83
27	47,06	18,23	44,51	44,72	44,67	46,5	45,14	**47,06**	**47,06**	100,00
28	44,87	17,61	41,67	44,17	42,5	44,85	43,64	**44,87**	**44,87**	100,00

		Zodiac	Grafics	CA	GA	HGA	AG-JCK	AE	GRASP_DM	
Inst.	Melhor Valor Lit.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	%
29	54,27	52,14	41,37	51	-	**54,27**	54,23	**54,27**	**54,27**	100,00
30	44,62	33,01	32,86	40	-	43,85	27,71	**44,62**	43,58	97,67
31	58,14	33,46	55,43	55,29	53,8	57,69	55,32	**58,14**	57,06	98,14
32	59,66	46,06	56,32	58,7	56,61	59,43	52,54	**59,66**	**59,66**	100,00
33	50,51	21,11	47,96	46,3	45,93	**50,51**	49,23	**50,51**	**50,51**	100,00
34	43,37	32,73	39,41	40,05	-	41,71	15,78	**43,37**	42,71	98,48
35	57,54	52,21	52,21	-	-	56,14	56,83	**57,54**	56,82	98,75
36	84,03	83,66	83,92	83,92	**84,03**	**84,03**	**84,03**	**84,03**	**84,03**	100,00
Total		10	8	6	7	26	16	**33**	23	

Tabela 6.15: Comparação do GRAS_DM com algoritmos da literatura.

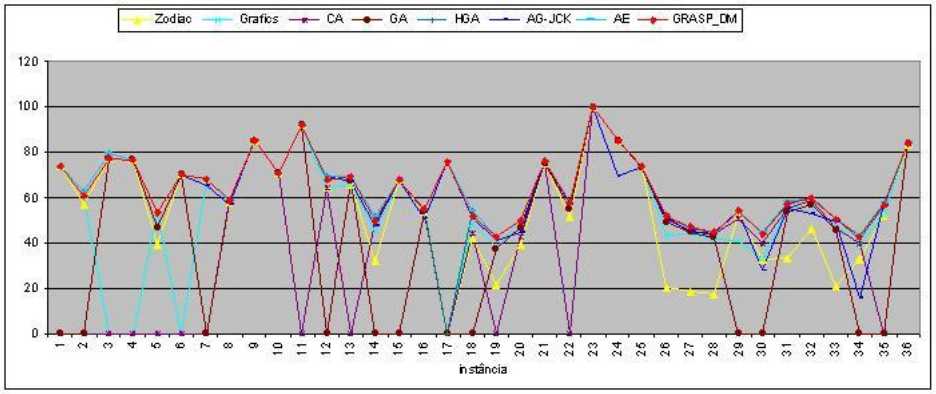

Figura 6.1: Comparação do GRAS_DM com os algoritmos da literatura.

Analisando a tabela 6.15 e o gráfico 6.1 pode-se notar que o algoritmo GRASP_DM alcançou os melhores resultados em 23 instâncias. E o algoritmo que obteve o maior número de melhores soluções foi o algoritmo AE que encontrou a melhor solução em 33 instâncias, seguido pelo HGA em 26 instâncias. Embora o GRASP_DM tenha sido superado por dois algorimos da literatura, ele foi superior a cinco outros algoritmos da literatura.

Comparação dos algoritmos propostos

A tabela 6.16 e o gráfico 6.2 comparam os resultados dos algoritmos propostos neste trabalho e testados até este ponto.

Os melhores resultados estão em negrito.

	GRASP 1	GRASP_FILTRO	GRASP_DM
Instância	Aptidão	Aptidão	Aptidão
1	**73,68**	**73,68**	**73,68**
2	**60,87**	**60,87**	**60,87**
3	**77,36**	**77,36**	**77,36**
4	**76,92**	**76,92**	**76,92**
5	**53,13**	**53,13**	**53,13**
6	**70,37**	**70,37**	**70,37**
7	**68,29**	**68,29**	**68,29**
8	58,41	58,41	**58,72**
9	**85,25**	**85,25**	**85,25**
10	**70,59**	**70,59**	**70,59**

	GRASP 1	GRASP_FILTRO	GRASP_DM
Instância	Aptidão	Aptidão	Aptidão
11	**92**	**92**	**92**
12	**67,65**	**67,65**	**67,65**
13	**69,33**	**69,33**	**69,33**
14	**50**	**50**	**50**
15	**67,83**	**67,83**	**67,83**
16	**54,86**	**54,86**	**54,86**
17	**75,71**	**75,71**	**75,71**
18	51,22	51,22	**51,64**
19	42,55	42,76	**42,96**
20	**49,65**	**49,65**	**49,65**
21	**76,14**	**76,14**	**76,14**
22	**57,95**	**57,95**	**57,95**
23	**100**	**100**	**100**
24	**85,11**	**85,11**	**85,11**
25	**73,51**	**73,51**	**73,51**
26	51,57	51,57	**51,88**
27	46,62	46,98	**47,06**
28	44,59	44,59	**44,87**
29	**54,27**	**54,27**	**54,27**
30	43,28	**43,7**	43,58
31	**57,06**	**57,06**	**57,06**
32	**59,66**	**59,66**	**59,66**
33	50,25	50,25	**50,51**
34	40,29	40,98	**42,71**
35	55,08	55,67	**56,82**
36	**84,03**	**84,03**	**84,03**
Total	26	27	**35**

Tabela 6.16: Comparação do GRAS_DM com os algoritmos propostos.

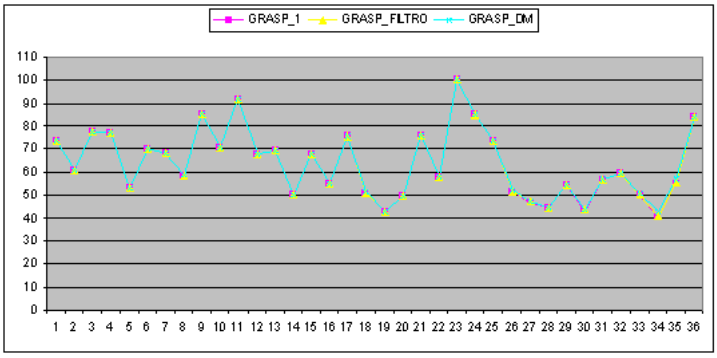

Figura 6.2: Comparação do GRAS_DM com os algoritmos propostos.

Analisando os resultados da tabela 6.16 e do gráfico 6.2, pode-se notar que o algoritmo GRASP_DM conseguiu superar o algoritmo GRASP_FILTRO em 8 instâncias e o GRASP1 em 9 instâncias. Desta forma, embora na comparação com o melhor algoritmo da literatura, a versão GRASP_DM tenha tido desempenho médio inferior, ele se mostra superior do que cinco algoritmos da literatura e tambem em relação as versões GRASP1 e GRASP_FILTRO.

6.2.4 Resultados do Algoritmo GRASP_PR

O algoritmo GRASP_PR foi submetido a duas baterias de testes:

- A primeira serviu para avaliar qual o melhor valor para o parâmetro que define o momento de utilizar os melhores valores de alfas, beta e número de clusters.
- A segunda serviu para avaliar o melhor valor para o parâmetro que define o limite para inserir uma solução no vetor de melhores soluções. Uma solução será inserida apenas se ela for maior ou igual a max_vetor % da melhor solução já existente no vetor.

Assim como nos testes do algoritmo GRASP_DM, estes testes não apresentam resultados variando os demais parâmetros.

Este algoritmo também foi executado primeiramente sem utilizar o controle de soluções inicias apresentado no GRASP_FILTRO que faz com que o algoritmo não utilize a mesma solução inicial mais de uma vez. E depois foi executado utilizando este controle. Os resultados mostrados em cada bateria de testes são os resultados obtidos sem este controle.

Após a apresentação dos resultados das duas baterias é mostrado um comparativo dos resultados com e sem controle de soluções inicias. Para esta comparção foram utilizados os resultados da segunda bateria, pois esta encontrou os melhores resultados.

Em cada Bateria de testes o algoritmo foi executado 10 vezes para cada instância e a configuração da bateria 2 utilizou os melhores valores definidos pela bateria 1.

6.2.4.1 Resultados da Primeira Bateria - variando o limite para usar melhores parâmetros dinâmicos.

Nesta bateria foram utilizadas 4 configurações diferentes variando o limite. Os valores utilizados para este parâmetro em cada configuração, assim como nos outros algoritmos, foram: 0,8 ;

0,9 e 1 .

O algoritmo foi executado 10 vezes para cada instância com cada uma das 4 configurações.

A tabela 6.17 mostra a comparação entre os resultados dos testes com as 3 configurações. Os melhores resultados estão em negrito.

	Config 1 (0,8)	Config 2 (0,9)	Config 3 (1)
Instância	Aptidão	Aptidão	Aptidão
1	**73,68**	**73,68**	**73,68**
2	**60,87**	**60,87**	**60,87**
3	**79,59**	**79,59**	**79,59**
4	**76,92**	**76,92**	**76,92**
5	**53,13**	**53,13**	**53,13**
6	**70,37**	**70,37**	**70,37**
7	**68,29**	**68,29**	**68,29**
8	**58,41**	**58,41**	**58,41**
9	**85,25**	**85,25**	**85,25**
10	**70,59**	**70,59**	**70,59**
11	**92**	**92**	**92**
12	**67,65**	**67,65**	**67,65**
13	**69,33**	**69,33**	**69,33**
14	**50**	**50**	**50**
15	**67,83**	**67,83**	**67,83**
16	**54,86**	**54,86**	**54,86**
17	**75,71**	**75,71**	**75,71**
18	51,22	51,22	**51,64**
19	**42,74**	**42,74**	**42,74**
20	**49,65**	**49,65**	**49,65**
21	**76,14**	**76,14**	**76,14**
22	57,71	**57,95**	**57,95**
23	**100**	**100**	**100**
24	**85,11**	**85,11**	**85,11**
25	**73,51**	**73,51**	**73,51**
26	**51,57**	51,5	51,5
27	**46,98**	**46,98**	**46,98**
28	44,25	44,25	**44,3**
29	**54,27**	**54,27**	**54,27**

	Config 1 (0,8)	Config 2 (0,9)	Config 3 (1)
Instância	Aptidão	Aptidão	Aptidão
30	43,2	43,49	**43,53**
31	**57,06**	**57,06**	**57,06**
32	**58,76**	**58,76**	**58,76**
33	**50,25**	**50,25**	**50,25**
34	41,16	41,04	**41,35**
35	58,55	57,54	**58,89**
36	**84,03**	**84,03**	**84,03**
Total de melhores soluções	30	30	**35**

Tabela 6.17: Comparação da primeira bateria de testes do GRASP_PR.

Analisando os resultados da tabela 6.17 pode-se notar que a configuração que obteve o melhor resultado no maior número de instâncias foi a **configuração 3** que utilizou limite igual a **1**.

6.2.4.2 Resultados da Segunda Bateria - variando percentual de diferença no vetor de melhores soluções.

Nesta bateria foram utilizadas 4 configurações diferentes variando o limite para inserir uma solução no vetor de melhores soluções. Uma solução será inserida apenas se ela for maior ou igual a max_vetor % da melhor solução já existente no vetor.

Se este parâmetro for igual a 0 qualquer solução poderá ser inserida no vetor de melhores soluções, influenciando negativamente a qualidade do vetor e se este parâmetro for igual a 1 o vetor de melhores soluções conterá apenas a melhor solução. Como entendemos que esses valores extremos não são eficazes, os valores utilizados para este parâmetro em cada configuração foram: 0,3 ; 0,5 ; 0,7 e 0,9 .

Foi utilizado o melhor valor para o parâmetro limite encontrado no resultado da bateria 1, ou seja, o valor escolhido foi igual a 1.

O algoritmo foi executado 10 vezes para cada instância com cada uma das 4 configurações.

A tabela 6.18 mostra a comparação entre os resultados dos testes das 4 configurações. Os melhores resultados estão em negrito.

	Config 1 (0,3)	Config 2 (0,5)	Config 3 (0,7)	Config 3 (0,9)
Instância	Aptidão	Aptidão	Aptidão	Aptidão
1	**73,68**	**73,68**	**73,68**	**73,68**
2	**60,87**	**60,87**	**60,87**	**60,87**
3	**79,59**	**79,59**	**79,59**	**79,59**
4	**76,92**	**76,92**	**76,92**	**76,92**
5	**53,13**	**53,13**	**53,13**	**53,13**
6	**70,37**	**70,37**	**70,37**	**70,37**
7	**68,29**	**68,29**	**68,29**	**68,29**
8	**58,41**	**58,41**	**58,41**	**58,41**
9	**85,25**	**85,25**	**85,25**	**85,25**
10	**70,59**	**70,59**	**70,59**	**70,59**
11	**92**	**92**	**92**	**92**
12	**67,65**	67,12	**67,65**	**67,65**
13	**69,33**	**69,33**	**69,33**	**69,33**
14	**50**	49,11	**50**	**50**
15	**67,83**	**67,83**	**67,83**	**67,83**
16	**54,86**	**54,86**	**54,86**	**54,86**
17	**75,71**	**75,71**	**75,71**	**75,71**
18	51,22	51,22	**51,64**	51,22
19	**42,74**	**42,74**	**42,74**	**42,74**
20	**49,65**	**49,65**	**49,65**	**49,65**
21	**76,14**	**76,14**	**76,14**	**76,14**
22	57,71	57,71	**57,95**	57,71
23	**100**	**100**	**100**	**100**
24	**85,11**	**85,11**	**85,11**	**85,11**
25	**73,51**	**73,51**	**73,51**	**73,51**
26	51,28	**51,57**	51,5	**51,57**
27	46,98	46,98	46,98	**47,06**
28	44,37	44,3	44,3	**44,59**
29	**54,27**	**54,27**	**54,27**	**54,27**
30	**43,58**	**43,58**	43,53	43,49
31	57,06	**57,89**	57,06	57,06
32	**59,66**	58,76	58,76	58,76

	Config 1 (0,3)	Config 2 (0,5)	Config 3 (0,7)	Config 3 (0,9)
Instância	Aptidão	Aptidão	Aptidão	Aptidão
33	**50,25**	**50,25**	**50,25**	**50,25**
34	41,29	40,76	**41,35**	41,25
35	**58,89**	58,13	**58,89**	**58,89**
36	**84,03**	**84,03**	**84,03**	**84,03**
Total de melhores soluções	29	27	**30**	**30**

Tabela 6.18: Comparação dos Resultados das 4 configurações da bateria 2 do algoritmo GRASP_PR.

Analisando a tabela 6.18 pode-se notar que duas configurações obtiveram o melhor resultado no maior número de instâncias. Essas configurações foram a **configuração 3** que utilizou percentual **0,7** e a **configuração 4** que utilizou percentual **0,9**. Em seguida tivemos a configuração 1 e com pior desempenho, a configuração 2.

6.2.4.3 Conclusões Parciais

Conclusões das 2 baterias de testes com o algoritmo GRASP_PR

A tabela 6.19 compara os resultados dos melhores resultados das 2 baterias de testes realizados.

	Bateria 1 (limite)	Bateria 2 (freqüência)
Instância	Aptidão	Aptidão
1	**73,68**	**73,68**
2	**60,87**	**60,87**
3	**79,59**	**79,59**
4	**76,92**	**76,92**
5	**53,13**	**53,13**
6	**70,37**	**70,37**
7	**68,29**	**68,29**
8	**58,41**	**58,41**
9	**85,25**	**85,25**
10	**70,59**	**70,59**
11	**92**	**92**
12	**67,65**	**67,65**
13	**69,33**	**69,33**
14	**50**	**50**
15	**67,83**	**67,83**
16	**54,86**	**54,86**
17	**75,71**	**75,71**
18	**51,64**	**51,64**
19	**42,74**	**42,74**

	Bateria 1 (limite)	Bateria 2 (freqüência)
Instância	Aptidão	Aptidão
20	**49,65**	**49,65**
21	**76,14**	**76,14**
22	**57,95**	**57,95**
23	**100**	**100**
24	**85,11**	**85,11**
25	**73,51**	**73,51**
26	**51,57**	**51,57**
27	46,98	**47,06**
28	44,3	**44,59**
29	**54,27**	**54,27**
30	43,53	**43,58**
31	57,06	**57,89**
32	58,76	**59,66**
33	**50,25**	**50,25**
34	**41,35**	**41,35**
35	**58,89**	**58,89**
36	**84,03**	**84,03**
Total	31	**36**

Tabela 6.19: Comparação entre as 2 baterias de testes do GRASP_PR.

Na tabela 6.19, verificamos melhor desempenho na bateria 2. De fato, como a segunda bateria utilizou um dos limites definidos como melhor pela bateria anterior a última bateria obteve os melhores resultados em todas as instâncias.

Após as 2 baterias de testes realizadas e análise dos resultados empíricos pode-se concluir que a melhor configuração de parâmetros para estas instâncias é a configuração descrita a seguir:

- Número de iterações que o algoritmo irá executar: 500
- Intervalo (número de iterações) de execução da mineração de dados: 25
- Percentual para começar a utilizar os melhores valores dos parâmetros gerados dinamicamente: **1**. Desta forma o algoritmo não irá utilizar os melhores valores dos parâmetros dinâmicos e ficará variando eles até o final.
- Freqüência mínima para um cluster ser alocado em uma máquina ou peça durante a mineração: 0,4
- Percentual mínimo para uma solução ser inserida no vetor de melhores soluções: **0,7** ou **0,9**
- Número de soluções gerada na fase de construção antes de retornar a melhor: 20
- Número de repetições de mineração para um grupo de soluções: 10
- Tamanho do vetor de melhores soluções: 20

- Parâmetro que determina se a mineração é ou não gulosa: 0
- Parâmetro que define se a mineração será realizada: 0
- Parâmetro que define se reconexão por caminhos será realizado: 1
- Tamanho do vetor de soluções iniciais: 20
- Número de repetições de solução inicial toleradas antes de terminar o algoritmo: 10

A tabela 6.20 mostra com mais detalhes, os resultados da configuração acima utilizando 0,7 como percentual mínimo para uma solução ser inserida no vetor de melhores soluções.

Inst	Aptidão	Aptidão Média	# Clusters	Tempo Total Médio
1	73,68	73,68	2	0,35
2	60,87	60,87	2	0,36
3	79,59	78,03	2	0,98
4	76,92	76,92	2	0,91
5	53,13	53,13	3	1,02
6	70,37	70,37	3	0,70
7	68,29	68,29	3	1,26
8	58,41	57,94	2	1,55
9	85,25	85,25	3	1,45
10	70,59	70,59	3	2,05
11	92	92	3	1,84
12	67,65	67,23	6	4,63
13	69,33	69,33	5	5,10
14	50	48,26	6	8,92
15	67,83	67,83	4	7,90
16	54,86	54,86	5	8,81
17	75,71	75,71	4	13,39
18	51,64	49,98	5	21,66
19	42,74	41,8	6	33,81

Inst	Aptidão	Aptidão Média	# Clusters	Tempo Total Médio
20	49,65	49,51	5	24,50
21	76,14	76,14	4	13,41
22	57,95	57,28	6	17,37
23	100	100	7	29,93
24	85,11	85,11	7	24,74
25	73,51	73,51	7	31,66
26	51,5	51,32	8	32,71
27	46,98	46,77	11	27,21
28	44,3	44,24	10	26,31
29	54,27	54,27	4	51,90
30	43,53	42,99	9	102,14
31	57,06	57,06	9	72,86
32	58,76	58,53	12	68,11
33	50,25	50,25	11	68,19
34	41,35	40,45	11	149,45
35	58,89	57,23	3	181,19
36	84,03	84,03	10	275,13

Tabela 6.20: Resultados detalhados do algoritmo GRASP_PR na bateria 2 com configuração 3.

Resultados obtidos com controle de soluções inicias

O algoritmo GRASP_PR foi submetido as mesmas 2 baterias de testes apresentadas acima utilizando controle de soluções iniciais. A tabela 6.21 compara os resultados destes testes com os anteriores:

	Bateria 3 SEM CONTROLE DE SOL. INICIAIS	Bateria 3 COM CONTROLE DE SOL. INICIAIS
Instância	Aptidão	Aptidão
1	**73,68**	**73,68**
2	**60,87**	**60,87**
3	**79,59**	77,36
4	**76,92**	**76,92**
5	**53,13**	**53,13**
6	**70,37**	**70,37**
7	**68,29**	**68,29**
8	**58,41**	**58,41**
9	**85,25**	**85,25**
10	**70,59**	**70,59**

	Bateria 3 SEM CONTROLE DE SOL. INICIAIS	Bateria 3 COM CONTROLE DE SOL. INICIAIS
Instância	Aptidão	Aptidão
11	**92**	**92**
12	**67,65**	**67,65**
13	**69,33**	**69,33**
14	**50**	**50**
15	**67,83**	**67,83**
16	**54,86**	**54,86**
17	**75,71**	**75,71**
18	**51,64**	51,22
19	42,74	**42,76**
20	**49,65**	**49,65**
21	**76,14**	**76,14**
22	**57,95**	**57,95**
23	**100**	**100**
24	**85,11**	**85,11**
25	**73,51**	**73,51**
26	**51,57**	51,5
27	**47,06**	**47,06**
28	**44,59**	44,52
29	**54,27**	**54,27**
30	43,58	**43,75**
31	**57,89**	57,06
32	**59,66**	58,76
33	50,25	**50,51**
34	**41,35**	**41,35**
35	58,89	**59,04**
36	**84,03**	**84,03**
Total	**32**	30

Tabela 6.21: Comparação dos resultados do GRADP_PR com e sem controle de soluções iniciais.

Podemos notar através da tabela 6.21 que os resultados empíricos obtidos sem controle de soluções inicias são melhores do que os obtidos com este controle. O algoritmo sem controle obteve melhores resultados em 6 instâncias e com controle em 3 instâncias.

Comparação dos resultados com a literatura

A próxima tabela e gráfico comparam os melhores resultados obtidos pelo GRASP_PR (tabela 6.21) e os resultados já existentes na literatura.

Os melhores resultados estão em negrito.

		Zodiac	Grafics	CA	GA	HGA	AG-JCK	AE	GRASP_PR	
Inst.	Melhor Valor Lit.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	%
1	73,68	**73,68**	**73,68**	-	-	**73,68**	**73,68**	**73,68**	**73,68**	100,00
2	62,5	56,52	60,87	-	68 (inv)	**62,5**	**62,5**	**62,5**	60,87	97,39
3	79,59	77,36	-	-	77,36	**79,59**	**79,59**	**79,59**	**79,59**	100,00
4	76,92	**76,92**	-	-	**76,92**	**76,92**	**76,92**	**76,92**	**76,92**	100,00
5	53,13	39,13	53,12	-	46,88	**53,13**	**53,13**	**53,13**	**53,13**	100,00
6	70,37	**70,37**	-	-	**70,37**	**70,37**	**70,37**	**70,37**	**70,37**	100,00
7	68,3	**68,3**	**68,3**	-	-	**68,3**	65,12	68,29	68,29	99,99
8	58,72	58,33	58,13	**58,72**	58,33	**58,72**	57,26	**58,72**	58,41	99,47
9	85,25	85,24	85,24	85,24	85,24	**85,25**	**85,25**	**85,25**	**85,25**	100,00
10	70,59	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**	100,00
11	92	**92**	**92**	-	**92**	**92**	**92**	**92**	**92**	100,00
12	69,86	64,36	64,36	64,36	-	**69,86**	**69,86**	**69,86**	67,65	96,84
13	69,33	65,55	65,55	-	67,44	**69,33**	67,05	**69,33**	**69,33**	100,00
14	51,96	32,09	45,52	48,7	-	**51,96**	47,69	**51,96**	50	96,23
15	67,83	**67,83**	**67,83**	**67,83**	-	**67,83**	**67,83**	**67,83**	**67,83**	100,00
16	54,86	53,76	54,39	54,44	53,89	**54,86**	50,88	**54,86**	**54,86**	100,00
17	75,71	-	-	-	-	-	**75,71**	**75,71**	**75,71**	100,00
18	54,46	41,84	48,91	44,2	-	**54,46**	51,24	**54,46**	51,64	94,82
19	42,96	21,63	38,26	-	37,12	**42,96**	41,14	**42,96**	42,76	99,53
20	49,65	38,66	49,36	43,01	46,62	**49,65**	44,57	**49,65**	**49,65**	100,00
21	76,22	75,14	75,14	75,14	75,28	76,14	**76,22**	76,14	76,14	99,90
22	58,07	51,13	-	-	55,14	**58,07**	56,52	58,06	57,95	99,79
23	100	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	100,00
24	85,11	**85,11**	**85,11**	**85,11**	**85,11**	**85,11**	69,59	**85,11**	**85,11**	100,00
25	73,51	**73,51**	**73,51**	**73,51**	73,03	**73,51**	**73,51**	**73,51**	**73,51**	100,00
26	51,97	20,42	43,27	51,81	49,37	51,85	50,3	**51,97**	51,57	99,23
27	47,06	18,23	44,51	44,72	44,67	46,5	45,14	**47,06**	**47,06**	100,00

		Zodiac	Grafics	CA	GA	HGA	AG-JCK	AE	GRASP_PR	
Inst.	Melhor Valor Lit.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	%
28	44,87	17,61	41,67	44,17	42,5	44,85	43,64	**44,87**	44,59	99,38
29	54,27	52,14	41,37	51	-	**54,27**	54,23	**54,27**	**54,27**	100,00
30	44,62	33,01	32,86	40	-	43,85	27,71	**44,62**	43,75	98,05
31	58,14	33,46	55,43	55,29	53,8	57,69	55,32	**58,14**	57,89	99,57
32	59,66	46,06	56,32	58,7	56,61	59,43	52,54	**59,66**	**59,66**	100,00
33	50,51	21,11	47,96	46,3	45,93	**50,51**	49,23	**50,51**	**50,51**	100,00
34	43,37	32,73	39,41	40,05	-	41,71	15,78	**43,37**	41,35	95,34
35	57,54	52,21	52,21	-	-	56,14	56,83	57,54	**59,04**	102,61
36	84,03	83,66	83,92	83,92	**84,03**	**84,03**	**84,03**	**84,03**	**84,03**	100,00
Total		10	8	6	7	26	16	32	22	

Tabela 6.22: Comparação do GRAS_PR com a algoritmos da literatura.

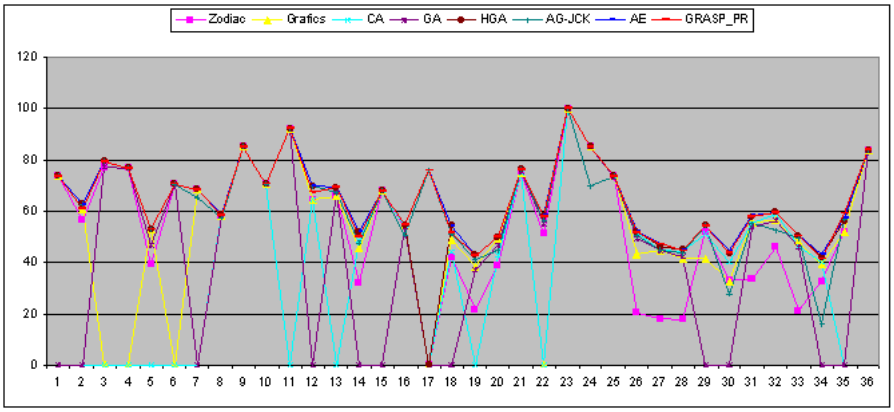

Figura 6.3: Comparação do GRAS_PR com os algoritmos propostos.

Analisando a tabela 6.22 e o gráfico 6.3 pode-se notar que o algoritmo GRASP_PR alcançou os melhores resultados em 22 instâncias, sendo que conseguiu encontrar um valor melhor do que qualquer outro da literatura na instância 35. E o algoritmo que obteve o maior número de melhores soluções foi o algoritmo AE que encontrou a melhor solução em 32 instâncias, seguido pelo HGA com 26. Comparando a versão GRASP_PR com a versão GRASP_DM observa-se um equilíbrio entre ambas as versões (tabela 6.15).

Comparação dos algoritmos propostos

A próxima tabela e gráfico comparam os resultados dos algoritmos propostos neste trabalho e testados até este ponto.

Os melhores resultados estão em negrito.

	GRASP 1	GRASP_FILTRO	GRASP_DM	GRASP_PR
Instância	Aptidão	Aptidão	Aptidão	Aptidão
1	**73,68**	**73,68**	**73,68**	**73,68**
2	**60,87**	**60,87**	**60,87**	**60,87**
3	77,36	77,36	77,36	**79,59**
4	**76,92**	**76,92**	**76,92**	**76,92**
5	**53,13**	**53,13**	**53,13**	**53,13**
6	**70,37**	**70,37**	**70,37**	**70,37**
7	**68,29**	**68,29**	**68,29**	**68,29**
8	58,41	58,41	**58,72**	58,41
9	**85,25**	**85,25**	**85,25**	**85,25**
10	**70,59**	**70,59**	**70,59**	**70,59**
11	**92**	**92**	**92**	**92**
12	**67,65**	**67,65**	**67,65**	**67,65**
13	**69,33**	**69,33**	**69,33**	**69,33**
14	**50**	**50**	**50**	**50**
15	**67,83**	**67,83**	**67,83**	**67,83**
16	**54,86**	**54,86**	**54,86**	**54,86**
17	**75,71**	**75,71**	**75,71**	**75,71**

	GRASP 1	GRASP_FILTRO	GRASP_DM	GRASP_PR
Instância	Aptidão	Aptidão	Aptidão	Aptidão
18	51,22	51,22	**51,64**	**51,64**
19	42,55	42,76	**42,96**	42,76
20	**49,65**	**49,65**	**49,65**	**49,65**
21	**76,14**	**76,14**	**76,14**	**76,14**
22	**57,95**	**57,95**	**57,95**	**57,95**
23	**100**	**100**	**100**	**100**
24	**85,11**	**85,11**	**85,11**	**85,11**
25	**73,51**	**73,51**	**73,51**	**73,51**
26	51,57	51,57	**51,88**	51,57
27	46,62	46,98	**47,06**	**47,06**
28	44,59	44,59	**44,87**	44,59
29	**54,27**	**54,27**	**54,27**	**54,27**
30	43,28	43,7	43,58	**43,75**
31	57,06	57,06	57,06	**57,89**
32	**59,66**	**59,66**	**59,66**	**59,66**
33	50,25	50,25	**50,51**	**50,51**
34	40,29	40,98	**42,71**	41,35
35	55,08	55,67	56,82	**59,04**
36	**84,03**	**84,03**	**84,03**	**84,03**
Total	24	24	**32**	31

Tabela 6.23: Comparação do GRAS_PR com os algoritmos propostos.

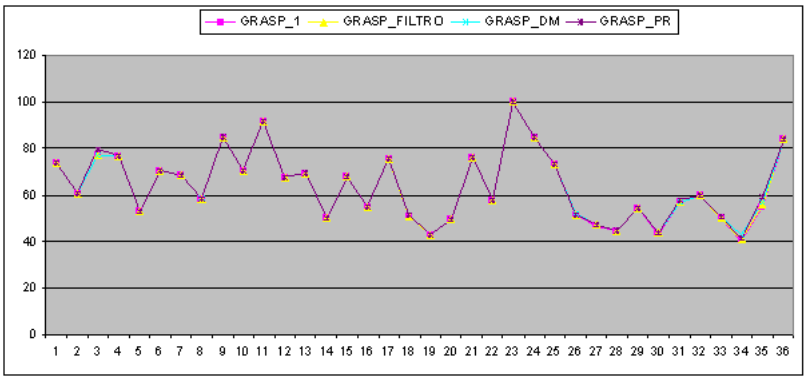

Figura 6.4: Comparação do GRAS_PR com os algoritmos propostos.

Analisando a tabela 6.23 e o gráfico 6.4, pode-se notar que o algoritmo proposto que obteve os melhores resultados até este ponto foi o algoritmo GRASP_DM, seguido pelo GRASP_PR. Os piores resultados foram dos algoritmos GRASP1 e GRASP_FILTRO com melhores resultados em 24 instâncias cada um, porém o GRASP_FILTRO apresentou melhores resultados que o GRASP1.

6.2.5 Resultados do Algoritmo GRASP_DM_PR

O algoritmo GRASP_DM_PR utiliza a técnica de mineração de dados do algoritmo GRASP_DM combinado com a técnica de reconexão por caminhos do algoritmo GRASP_PR.

Por ser uma combinação das duas técnicas já testadas, os testes realizados neste algoritmo utilizou os melhores valores de parâmetros definidos nos testes anteriores. Para o parâmetro limite para usar melhores alfa, beta e número de cluster foram utilizados os valores 0,9 e 1. Para freqüência mínima foi utilizado o valor 0,4. E para percentual mínimo para entrar no vetor de melhores soluções foram utilizados os valores 0,5 ; 0,7 e 0,9 . Para este último parâmetro foi adicionado um teste com o valor 0,8 que não foi utilizado nos testes dos algoritmos anteriores para verificar como o algoritmo se comportaria com este novo valor.

Desta forma foram realizados testes com oito configurações diferentes combinando os valores acima.

Este algoritmo assim como os anteriores foi executado primeiramente sem utilizar o controle de soluções inicias apresentado no GRASP_FILTRO que faz com que o algoritmo não utilize a mesma solução inicial mais de uma vez e depois foi executado utilizando este controle. Os resultados mostrados em cada bateria de testes são os resultados obtidos sem este controle.

Após a apresentação dos resultados das oito baterias é mostrado um comparativo dos resultados com e sem controle de soluções inicias.

6.2.5.1 Resultados variando o limite para usar melhores parâmetros dinâmicos e o percentual mínimo para entrar no vetor de melhores soluções.

O algoritmo foi executado 10 vezes para cada instância com cada uma das 8 configurações. A seguir estão os resultados de cada configuração.

A tabela 6.24 mostra a comparação entre os resultados dos testes das 8 configurações. Os melhores resultados estão em negrito.

	Config 1	Config 2	Config 3	Config 4	Config 5	Config 6	Config 7	Config 8
Inst.	limite 0,9 maxvet 0,5	limite 0,9 maxvet 0,7	limite 0,9 maxvet 0,8	limite 0,9 maxvet 0,9	limite 1 maxvet 0,5	limite 1 maxvet 0,7	limite 1 maxvet 0,8	limite 1 maxvet 0,9
	Aptidão	Aptidão	Aptidão	Aptidão	Aptidão	Aptidão	Aptidão	Aptidão
1	**73,68**	**73,68**	**73,68**	**73,68**	**73,68**	**73,68**	**73,68**	**73,68**
2	**60,87**	**60,87**	**60,87**	**60,87**	**60,87**	**60,87**	**60,87**	**60,87**
3	**79,59**	**79,59**	**79,59**	77,36	**79,59**	77,36	**79,59**	**79,59**
4	**76,92**	**76,92**	**76,92**	**76,92**	**76,92**	**76,92**	**76,92**	**76,92**
5	**53,13**	**53,13**	**53,13**	**53,13**	**53,13**	**53,13**	**53,13**	**53,13**
6	**70,37**	**70,37**	**70,37**	**70,37**	**70,37**	**70,37**	**70,37**	**70,37**
7	**68,29**	**68,29**	**68,29**	**68,29**	**68,29**	**68,29**	**68,29**	**68,29**
8	**58,72**	**58,72**	**58,72**	58,41	**58,72**	**58,72**	**58,72**	**58,72**
9	**85,25**	**85,25**	**85,25**	**85,25**	**85,25**	**85,25**	**85,25**	**85,25**
10	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**
11	**92**	**92**	**92**	**92**	**92**	**92**	**92**	**92**
12	**67,65**	**67,65**	**67,65**	**67,65**	**67,65**	**67,65**	**67,65**	67,12
13	**69,33**	**69,33**	**69,33**	**69,33**	**69,33**	**69,33**	**69,33**	**69,33**
14	**50**	**50**	49,56	**50**	**50**	**50**	**50**	**50**
15	**67,83**	**67,83**	**67,83**	**67,83**	**67,83**	**67,83**	**67,83**	**67,83**
16	**54,86**	**54,86**	**54,86**	**54,86**	**54,86**	**54,86**	**54,86**	**54,86**
17	**75,71**	**75,71**	**75,71**	**75,71**	**75,71**	**75,71**	**75,71**	**75,71**
18	51,22	51,64	**51,72**	51,22	50,82	51,64	51,22	51,22
19	42,74	42,76	42,76	42,76	42,74	42,74	42,76	**42,86**
20	**49,65**	**49,65**	**49,65**	**49,65**	**49,65**	**49,65**	**49,65**	**49,65**
21	**76,14**	**76,14**	**76,14**	**76,14**	**76,14**	**76,14**	**76,14**	**76,14**
22	57,95	57,95	57,95	57,71	**58,06**	57,95	**58,06**	57,71
23	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
24	**85,11**	**85,11**	**85,11**	**85,11**	**85,11**	**85,11**	**85,11**	**85,11**
25	**73,51**	**73,51**	**73,51**	**73,51**	**73,51**	**73,51**	**73,51**	**73,51**
26	51,57	51,57	51,57	**51,59**	51,57	51,57	51,57	51,57
27	46,98	47,06	**47,13**	46,98	46,98	46,98	46,98	47,06
28	**44,72**	44,52	44,65	44,3	44,3	44,52	44,52	44,52

	Config 1	Config 2	Config 3	Config 4	Config 5	Config 6	Config 7	Config 8
Inst.	limite 0,9 maxvet 0,5	limite 0,9 maxvet 0,7	limite 0,9 maxvet 0,8	limite 0,9 maxvet 0,9	limite 1 maxvet 0,5	limite 1 maxvet 0,7	limite 1 maxvet 0,8	limite 1 maxvet 0,9
	Aptidão	Aptidão	Aptidão	Aptidão	Aptidão	Aptidão	Aptidão	Aptidão
29	**54,27**	**54,27**	**54,27**	**54,27**	**54,27**	**54,27**	**54,27**	**54,27**
30	44,06	**44,15**	43,92	43,38	43,77	43,54	43,88	43,56
31	57,06	57,06	**57,89**	57,06	57,06	57,06	57,06	57,06
32	**59,66**	**59,66**	**59,66**	**59,66**	58,76	**59,66**	58,76	58,76
33	**50,25**	**50,25**	**50,25**	**50,25**	**50,25**	**50,25**	**50,25**	**50,25**
34	41,28	41,32	41,03	40,86	41,41	41,03	**41,65**	40,86
35	**59,04**	**59,04**	**59,04**	58,9	**59,04**	**59,04**	58,75	**59,04**
36	**84,03**	**84,03**	**84,03**	**84,03**	**84,03**	**84,03**	**84,03**	**84,03**
Total	28	28	**29**	25	27	26	27	26

Tabela 6.24: Comparação dos resultados do algoritmo GRASP_DM_PR.

Analisando a tabela 6.24 pode-se notar que todas as configurações obtiveram resultados semelhantes. A configuração que obteve o melhor resultado no maior número de instâncias foi a **configuração 3** que utilizou a configuração abaixo e apresentou os resultados mostrados na tabela 6.25.

Configuração de parâmetros 3

- Número de iterações que o algoritmo irá executar: 500
- Intervalo (número de iterações) de execução da mineração de dados: 25
- Percentual para começar a utilizar os melhores valores dos parâmetros gerados dinamicamente: **0,9**. Desta forma o algoritmo começará a utilizar os melhores valores dos parâmetros dinâmicos após a iteração 500 * 0,9 = 450.
- Freqüência mínima para um cluster ser alocado em uma máquina ou peça durante a mineração: **0,4**
- Percentual mínimo para uma solução ser inserida no vetor de melhores soluções: **0,8**
- Número de soluções gerada na fase de construção antes de retornar a melhor: 20
- Número de repetições de mineração para um grupo de soluções: 10
- Tamanho do vetor de melhores soluções: 20
- Parâmetro que determina se a mineração é ou não gulosa: 0
- Parâmetro que define se a mineração será realizada: 1
- Parâmetro que define se reconexão por caminhos será realizado: 1
- Tamanho do vetor de soluções iniciais: 20
- Número de repetições de solução inicial toleradas antes de terminar o algoritmo: 10

Inst	Aptidão	Aptidão Média	# Clusters	Tempo Total Médio
1	73,68	73,68	2	0,36
2	60,87	60,87	2	0,35
3	79,59	77,81	2	0,93
4	76,92	76,92	2	0,81
5	53,13	53,13	3	0,85
6	70,37	70,37	3	0,70
7	68,29	68,29	3	1,25
8	58,72	58,02	2	1,35
9	85,25	85,25	3	1,50
10	70,59	70,59	3	2,06
11	92	92	3	1,82
12	67,65	67,28	6	5,09
13	69,33	69,33	5	5,00
14	49,56	48,57	5	8,03
15	67,83	67,83	4	7,92
16	54,86	54,86	5	9,37
17	75,71	75,71	4	13,03
18	51,72	49,12	5	19,08
19	42,76	42,23	4	34,00
20	49,65	49,51	5	23,24
21	76,14	76,14	4	13,16
22	57,95	57,2	6	16,36
23	100	100	7	28,13
24	85,11	85,11	7	24,09
25	73,51	73,51	7	32,38
26	51,57	51,33	9	31,11
27	47,13	46,83	10	26,86
28	44,65	44,27	10	27,38
29	54,27	54,27	4	48,25
30	43,92	43,15	7	122,73
31	57,89	57,14	10	68,98
32	59,66	58,58	12	71,92
33	50,25	50,25	11	70,66
34	41,03	40,41	12	149,89

Inst	Aptidão	Aptidão Média	# Clusters	Tempo Total Médio
35	59,04	57,71	3	171,70
36	84,03	84,03	10	258,07

Tabela 6.25: Resultados do algoritmo GRASP_DM_PR com configuração 3.

Resultados obtidos com controle de soluções inicias

O algoritmo GRASP_DM_PR foi submetido as mesmas 8 baterias de testes apresentadas acima utilizando controle de soluções iniciais. A tabela 6.26 compara os resultados destes testes com os anteriores:

	Resultados SEM CONTROLE DE SOL. INICIAIS	Resultados COM CONTROLE DE SOL. INICIAIS
Instância	Aptidão	Aptidão
1	**73,68**	**73,68**
2	**60,87**	**60,87**
3	**79,59**	77,36
4	**76,92**	**76,92**
5	**53,13**	**53,13**
6	**70,37**	**70,37**
7	**68,29**	**68,29**
8	**58,72**	58,41
9	**85,25**	**85,25**
10	**70,59**	**70,59**
11	**92**	**92**
12	**67,65**	**67,65**
13	**69,33**	**69,33**
14	**50**	49,55
15	**67,83**	**67,83**
16	**54,86**	**54,86**
17	**75,71**	**75,71**
18	**51,72**	51,24
19	**42,86**	42,76
20	**49,65**	**49,65**
21	**76,14**	**76,14**
22	**58,06**	**58,06**
23	**100**	**100**
24	**85,11**	**85,11**
25	**73,51**	**73,51**

	Resultados SEM CONTROLE DE SOL. INICIAIS	Resultados COM CONTROLE DE SOL. INICIAIS
Instância	Aptidão	Aptidão
26	51,59	**51,61**
27	47,13	**47,37**
28	44,72	**44,87**
29	**54,27**	**54,27**
30	44,15	**44,52**
31	**57,89**	**57,89**
32	**59,66**	**59,66**
33	50,25	**50,51**
34	41,65	**42,54**
35	**59,04**	**59,04**
36	**84,03**	**84,03**
Total	30	**31**

Tabela 6.26: Comparação dos resultados do GRASP_DM_PR com e sem controle de soluções iniciais.

Diferentemente dos resultados obtidos pelo GRASP_DM e GRASP_PR o algoritmo GRASP_DM_PR obteve resultados melhores utilizando controle de soluções iniciais. Este conseguiu resultados melhores em 6 instâncias enquanto que o sem controle foi melhor em 5 instâncias, ocorrendo empates nas demais instâncias.

Comparação dos Resultados com a Literatura

A tabela 6.27 e o gráfico 6.5 comparam os melhores resultados obtidos pelo GRASP_DM_PR com os resultados já existentes na literatura.

Os melhores resultados estão em negrito. Os casos em que o algoritmo GRASP_DM_PR superou os resultados existentes a percentagem está em negrito.

		Zodiac	Grafics	CA	GA	HGA	AG-JCK	AE	GRASP_DM_PR	
Inst.	Melhor Valor Lit.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	%
1	73,68	**73,68**	**73,68**	-	-	**73,68**	**73,68**	**73,68**	**73,68**	100,00
2	62,5	56,52	60,87	-	68 (inv)	**62,5**	**62,5**	**62,5**	60,87	97,39
3	79,59	77,36	-	-	77,36	**79,59**	**79,59**	**79,59**	**79,59**	100,00
4	76,92	**76,92**	-	-	**76,92**	**76,92**	**76,92**	**76,92**	**76,92**	100,00
5	53,13	39,13	53,12	-	46,88	**53,13**	**53,13**	**53,13**	**53,13**	100,00
6	70,37	**70,37**	-	-	**70,37**	**70,37**	**70,37**	**70,37**	**70,37**	100,00
7	68,3	**68,3**	**68,3**	-	-	**68,3**	65,12	68,29	68,29	99,99
8	58,72	58,33	58,13	**58,72**	58,33	**58,72**	57,26	**58,72**	**58,72**	100,00
9	85,25	85,24	85,24	85,24	85,24	**85,25**	**85,25**	**85,25**	**85,25**	100,00
10	70,59	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**	100,00
11	92	**92**	**92**	-	**92**	**92**	**92**	**92**	**92**	100,00
12	69,86	64,36	64,36	64,36	-	**69,86**	**69,86**	**69,86**	67,65	96,84
13	69,33	65,55	65,55	-	67,44	**69,33**	67,05	**69,33**	**69,33**	100,00
14	51,96	32,09	45,52	48,7	-	**51,96**	47,69	**51,96**	50	96,23
15	67,83	**67,83**	**67,83**	**67,83**	-	**67,83**	**67,83**	**67,83**	**67,83**	100,00
16	54,86	53,76	54,39	54,44	53,89	**54,86**	50,88	**54,86**	**54,86**	100,00
17	75,71	-	-	-	-	-	**75,71**	**75,71**	**75,71**	100,00
18	54,46	41,84	48,91	44,2	-	**54,46**	51,24	**54,46**	51,72	94,97
19	42,96	21,63	38,26	-	37,12	**42,96**	41,14	**42,96**	42,86	99,77
20	49,65	38,66	49,36	43,01	46,62	**49,65**	44,57	**49,65**	**49,65**	100,00
21	76,22	75,14	75,14	75,14	75,28	76,14	**76,22**	76,14	76,14	99,90
22	58,07	51,13	-	-	55,14	**58,07**	56,52	58,06	58,06	99,98
23	100	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	100,00
24	85,11	**85,11**	**85,11**	**85,11**	**85,11**	**85,11**	69,59	**85,11**	**85,11**	100,00
25	73,51	**73,51**	**73,51**	**73,51**	73,03	**73,51**	**73,51**	**73,51**	**73,51**	100,00
26	51,97	20,42	43,27	51,81	49,37	51,85	50,3	**51,97**	51,61	99,31
27	47,06	18,23	44,51	44,72	44,67	46,5	45,14	47,06	**47,37**	**100,66**
28	44,87	17,61	41,67	44,17	42,5	44,85	43,64	**44,87**	**44,87**	100,00

		Zodiac	Grafics	CA	GA	HGA	AG-JCK	AE	GRASP_DM_PR	
Inst.	Melhor Valor Lit.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	Apt.	%
29	54,27	52,14	41,37	51	-	**54,27**	54,23	**54,27**	**54,27**	100,00
30	44,62	33,01	32,86	40	-	43,85	27,71	**44,62**	44,52	99,78
31	58,14	33,46	55,43	55,29	53,8	57,69	55,32	**58,14**	57,89	99,57
32	59,66	46,06	56,32	58,7	56,61	59,43	52,54	**59,66**	**59,66**	100,00
33	50,51	21,11	47,96	46,3	45,93	**50,51**	49,23	**50,51**	**50,51**	100,00
34	43,37	32,73	39,41	40,05	-	41,71	15,78	**43,37**	42,54	98,09
35	57,54	52,21	52,21	-	-	56,14	56,83	57,54	**59,04**	**102,61**
36	84,03	83,66	83,92	83,92	**84,03**	**84,03**	**84,03**	**84,03**	**84,03**	100,00
Total		10	8	6	7	26	16	**31**	24	

Tabela 6.27: Comparação do GRAS_DM_PR com a algoritmos da literatura.

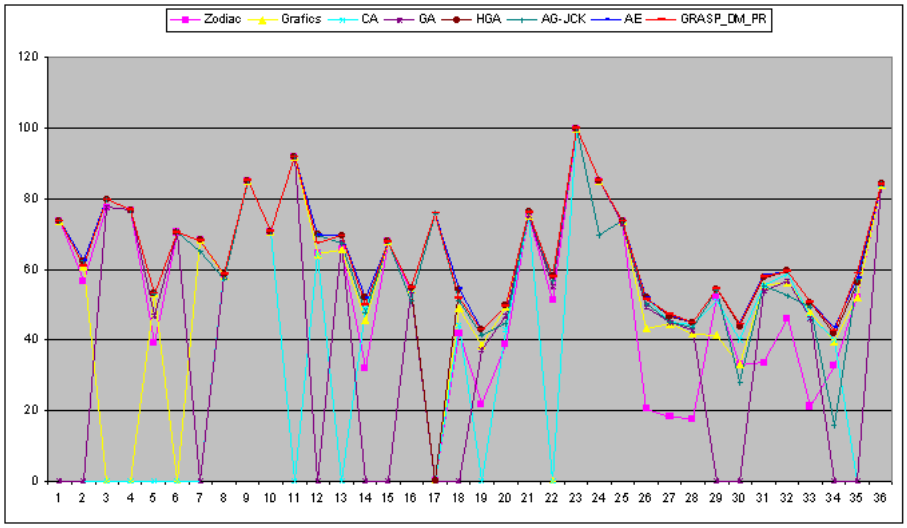

Figura 6.5: Comparação do GRASP_DM_PR com a literatura

Analisando a tabela 6.27 e o gráfico 6.5, pode-se notar que o algoritmo GRASP_DM_PR encontrou os melhores resultados da literatura em 24 instâncias, sendo que conseguiu superar os melhores resultados conhecidos da literatura nas instâncias **27** e **35**.

Embora o algoritmo GRASP_DM_PR não tenha atingido desempenho do AE proposto por Trindade e Ochi [38] e do HGA [12], observamos que a heurística proposta superou cinco algotimos da literatura. Além disso, seu desempenho ficou bem próximo do HGA como ilutra a última linha da tabela 6.27. Em termos de comparação das várias versões propostas neste trabalho, uma síntese é efetuada a seguir.

6.2.6 Avaliação Final

Neste tópico serão dadas as conclusões finais sobre os testes realizados.

O objetivo deste trabalho foi avaliar o comportamento do algoritmo GRASP na resolução do problema de formação de células de manufatura usando diferentes versões. A medida que novas técnicas foram sendo incorporadas nas novas versões os resultados obtidos foram evoluindo.

Segue abaixo um comparativo final dos resultados obtidos.

Comparação dos resultados dos Algoritmos Propostos

A tabela 6.28 e o gráfico 6.6 comparam os melhores valores encontrados para as dez instâncias testadas em cada um dos algoritmos propostos. Os melhores valores estão em negrito.

	GRASP1	GRASP_FILTRO	GRASP_PR	GRASP_DM	GRASP_DM_PR
Instância	Aptidão	Aptidão	Aptidão	Aptidão	Aptidão
1	**73,68**	**73,68**	**73,68**	**73,68**	**73,68**
2	**60,87**	**60,87**	**60,87**	**60,87**	**60,87**
3	77,36	77,36	**79,59**	77,36	**79,59**
4	**76,92**	**76,92**	**76,92**	**76,92**	**76,92**
5	**53,13**	**53,13**	**53,13**	**53,13**	**53,13**
6	**70,37**	**70,37**	**70,37**	**70,37**	**70,37**
7	**68,29**	**68,29**	**68,29**	**68,29**	**68,29**
8	58,41	58,41	58,41	**58,72**	**58,72**
9	**85,25**	**85,25**	**85,25**	**85,25**	**85,25**
10	**70,59**	**70,59**	**70,59**	**70,59**	**70,59**
11	**92**	**92**	**92**	**92**	**92**
12	**67,65**	**67,65**	**67,65**	**67,65**	**67,65**
13	**69,33**	**69,33**	**69,33**	**69,33**	**69,33**
14	**50**	**50**	**50**	**50**	**50**
15	**67,83**	**67,83**	**67,83**	**67,83**	**67,83**
16	**54,86**	**54,86**	**54,86**	**54,86**	**54,86**
17	**75,71**	**75,71**	**75,71**	**75,71**	**75,71**
18	51,22	51,22	51,64	51,64	**51,72**
19	42,55	42,76	42,76	**42,96**	42,86
20	**49,65**	**49,65**	**49,65**	**49,65**	**49,65**
21	**76,14**	**76,14**	**76,14**	**76,14**	**76,14**
22	57,95	57,95	57,95	57,95	**58,06**
23	**100**	**100**	**100**	**100**	**100**
24	**85,11**	**85,11**	**85,11**	**85,11**	**85,11**
25	**73,51**	**73,51**	**73,51**	**73,51**	**73,51**
26	51,57	51,57	51,57	**51,88**	51,61
27	46,62	46,98	47,06	47,06	**47,37**
28	44,59	44,59	44,59	**44,87**	**44,87**
29	**54,27**	**54,27**	**54,27**	**54,27**	**54,27**
30	43,28	43,7	43,75	43,58	**44,52**
31	57,06	57,06	**57,89**	57,06	**57,89**
32	**59,66**	**59,66**	**59,66**	**59,66**	**59,66**
33	50,25	50,25	**50,51**	**50,51**	**50,51**

	GRASP1	GRASP_FILTRO	GRASP_PR	GRASP_DM	GRASP_DM_PR
Instância	Aptidão	Aptidão	Aptidão	Aptidão	Aptidão
34	40,29	40,98	41,35	**42,71**	42,54
35	55,08	55,67	**59,04**	56,82	**59,04**
36	**84,03**	**84,03**	**84,03**	**84,03**	**84,03**
Total	23	23	27	29	**33**

Tabela 6.28: Comparação dos resultados dos algoritmos propostos.

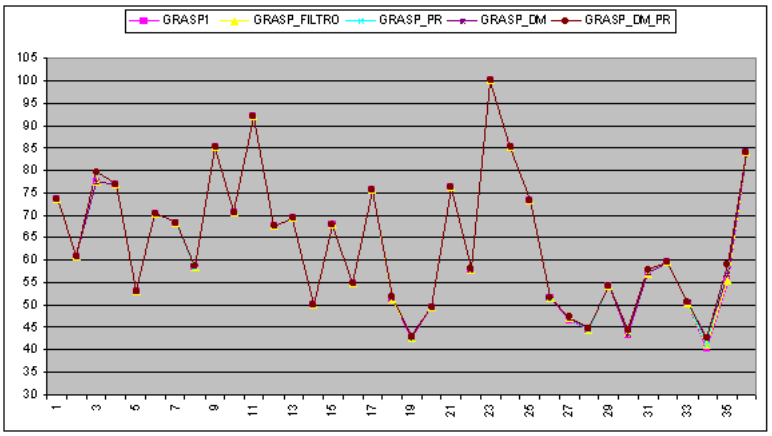

Figura 6.6: Comparação do GRASP_DM_PR com os algoritmos Propostos

A tabela 6.28 e o gráfico 6.6 mostram que o algoritmo GRASP_DM_PR obteve os melhores resultados em 33 das 36 instâncias. As outras 3 instâncias tiveram a melhor solução encontrada pelo algoritmo GRASP_DM. Isto mostra que a combinação das técnicas utilizadas torna o algoritmo GRASP mais eficiente.

Comparação dos Tempos dos Algoritmos Propostos

A tabela 6.29 compara o tempo médio em segundos de cada um dos algoritmos propostos. Nesta tabela é mostrado a iteração média e o tempo médio que o algoritmo levou para encontrar a melhor solução e para executar todas as iterações.

	GRASP1			GRASP_FILTRO			GRASP_PR			GRASP_DM			GRASP_DM_PR		
Inst.	Iter.	Tempo p/ Encontrar	Tempo Total	Iter.	Tempo p/ Encontrar	Tempo Total	Iter.	Tempo p/ Encontrar	Tempo Total	Iter.	Tempo p/ Encontrar	Tempo Total	Iter.	Tempo p/ Encontrar	Tempo Total
1	0	0	0,01	0	0	0,38	0	0	0,34	0	0	0,33	0	0	0,33
2	0	0	0,01	0	0	0,38	0	0	0,34	0	0	0,34	0	0	0,34
3	0	0	0,05	0	0	0,86	0	0	0,8	53	0,11	0,89	79	0,14	0,88
4	0	0	0,03	0	0	0,48	0	0	0,44	0	0	0,45	0	0	0,44
5	0	0	0,05	0	0	0,81	0	0	0,75	0	0	0,77	0	0	0,75
6	0	0	0,03	0	0	0,72	0	0	0,67	0	0	0,67	0	0	0,67
7	0	0	0,05	0	0	0,89	0	0	0,81	0	0	0,86	0	0	0,81
8	0	0	0,08	0	0	1,34	75	0,2	1,28	0	0,01	1,48	50	0,14	1,3
9	0	0	0,08	0	0	1,34	0	0	1,25	0	0	1,28	0	0	1,26
10	0	0	0,08	0	0	1,75	0	0	1,47	0	0	1,5	0	0	1,61
11	0	0	0,11	0	0	1,91	0	0	1,76	0	0	1,81	0	0	1,75
12	0	0	0,23	0	0,03	4,47	1	0,01	3,59	59	0,56	3,88	0	0,01	3,94
13	0	0	0,22	0	0	4,41	0	0,01	4,01	0	0,01	4,14	0	0,01	4,17
14	34	0,05	0,45	51	1,23	8,95	9	0,14	7,58	39	0,92	8,13	4	0,09	7,75
15	25	0,03	0,31	0	0,01	5,97	0	0,01	5,76	0	0,01	6,56	0	0,01	6,63
16	0	0	0,38	0	0,01	7,38	0	0,01	6,64	0	0,01	7,94	0	0,01	7,33
17	0	0,01	0,61	0	0,01	11,91	0	0,01	11,78	0	0,01	12,64	0	0,01	12,02
18	399	0,77	0,95	34	1,23	18,31	75	2,45	17,95	142	5,72	21,66	350	14,48	19,08
19	7	0,17	2,41	36	3,13	27,83	325	22,77	30,67	8	1,8	28,95	250	18,81	34,42
20	0	0,01	1,13	2	0,23	19,41	0	0,05	20,53	2	0,08	21,52	2	0,17	20,16
21	0	0	0,61	0	0,01	12,13	0	0,01	11,8	0	0,01	12,7	0	0,01	11,84
22	328	0,53	0,75	1	0,11	13,31	167	5,13	13,36	125	3,47	17,33	200	6,3	16,75
23	0	0	1	0	0,03	21,58	0	0,03	27,84	0	0,03	29,67	0	0,03	27,77
24	0	0	1,05	0	0,06	22,13	0	0,03	22,39	0	0,03	23,58	0	0,03	22,83
25	0	0,01	1,3	0	0,05	27,47	0	0,03	28,64	0	0,05	30,64	0	0,03	29,23
26	111	0,23	1,64	2	0,13	28,2	375	22,34	31	151	9,47	32,2	175	10,59	29,2
27	92	0,22	1,39	38	2,52	24,84	5	0,23	24,41	85	4,73	27,14	34	2,2	26,86

	GRASP1			GRASP_FILTRO			GRASP_PR			GRASP_DM			GRASP_DM_PR		
Inst.	Iter.	Tempo p/ Encontrar	Tempo Total	Iter.	Tempo p/ Encontrar	Tempo Total	Iter.	Tempo p/ Encontrar	Tempo Total	Iter.	Tempo p/ Encontrar	Tempo Total	Iter.	Tempo p/ Encontrar	Tempo Total
28	445	1,09	1,26	145	7,72	25,09	125	6,05	25,13	68	3,86	26,83	50	2,59	27,98
29	76	0,61	2,3	2	0,22	41,41	6	0,78	43,72	0	0,06	49	0	0,05	45,7
30	274	1,8	12,34	116	23,52	90,42	400	75,7	85,56	94	19,7	98,69	250	47,58	88,2
31	25	0,31	3,22	2	0,7	62,56	4	0,52	66,31	178	26,34	73,16	100	15,03	68,98
32	204	0,92	3,83	29	4,03	59,45	47	6,59	63,61	352	47,41	67,66	25	3,2	65,2
33	0	0,01	3,22	3	0,33	61,02	300	41,22	63,41	5	0,53	66,17	0	0,11	62,84
34	387	3,94	4,7	8	2,52	131,59	475	145,11	150,05	222	64,83	149,45	325	100,45	144,98
35	11	0,53	8,42	78	23,52	170,25	300	99,84	170,88	71	26,94	177,47	75	27,14	165,25
36	0	0,03	9,81	0	0,19	219,53	0	0,22	240,94	0	0,22	271,64	0	0,23	251,44

Tabela 6.29: Tempos dos algoritmos propostos.

Os tempos mostrados na tabela 6.29 mostram as seguintes características:

- O algoritmo GRASP_1 obteve os menores tempos em todas as iterações por ser o algoritmo mais simples.
- O algoritmo GRASP_FILTRO teve tempos maiores que o algoritmo GRASP_DM nas 18 primeiras instâncias, que são menores e mais rápidas e tempos menores nas outras instâncias que são maiores e mais lentas.
- O algoritmo GRASP_PR teve tempos menores que o algoritmos GRASP_DM em 5 instâncias e tempos maiores nas outras. Isto se explica porque o algoritmo GRASP_PR aplica a função de reconexão por caminhos em cada iteração e o algoritmo GRASP_DM aplica a mineração de dados apenas a cada X iterações (passado por parâmetro).
- O algoritmo GRASP_DM_PR apesar de utilizar dois métodos para encontrar a melhor solução teve tempo maior do que o algoritmo GRASP_PR em apenas 6 iterações e tempos menores nas outras. Isto pode ser explicado pelo fato do algoritmo GRASP ser muito aleatório e obter soluções diferentes e em tempos diferentes a cada execução. Outro motivo é o fato de que o GRASP_DM_PR encontra boas soluções utilizando a mineração de dados e diminui o trabalho realizado pela reconexão por caminhos.

A tabela 6.29 mostra também em qual iteração a melhor solução foi encontrada por cada algoritmo em cada instância. O resultado foi o seguinte:

- O algoritmo GRASP_1 encontrou o melhor resultado em menos de 100 iterações em 80,55 % das instâncias.
- O algoritmo GRASP_FILTRO encontrou o melhor resultado em menos de 100 iterações em 94,44 % das instâncias.
- O algoritmo GRASP_DM encontrou o melhor resultado em menos de 100 iterações em 77,77 % das instâncias.
- O algoritmo GRASP_PR encontrou o melhor resultado em menos de 100 iterações em 83,33 % das instâncias.
- O algoritmo GRASP_DM_PR encontrou o melhor resultado em menos de 100 iterações em 80,55 % das instâncias.

Analisando estes resultados podemos afirmar que não era necessário executar os algoritmos com 500 iterações para encontrar as melhores soluções.

Capítulo 7

Conclusões e Trabalhos Futuros

O ojetivo deste trabalho, foi o de desenvolver e analisar empiricamente diferentes heurísticas GRASP para o problema de formação de células de manufattura(PFCM). Embora este problema já tenha diferentes contribuições heurísticas na literatura, inicialmente acreditávamos que o uso da metaheurística GRASP pudesse trazer contribuiçoes relevantes na solução do PFCM.

Para isso, inicialmente apresentamos uma versão tradicional do GRASP, composto apenas por uma etapa de construção e outra de busca local. O algoritmo de construção segue a filosofia do GRASP de forma a permitir que a cada iteração GRASP, uma solução distinta das anteriores possa ser gerada. Denotamos esta versão de GRASP1.

Numa segunda versão, denotado por GRASP_FILTRO, apresentamos uma proposta para tentar melhorar a qualidade das soluções geradas na etapa de construção do GRASP1. Para isso, simplesmente passamos a gerar um conjunto de soluções viáveis a cada iteração do GRASP ao invés de uma única solução por iteração, como é feito nos algortimos GRASP de uma forma geral e em particular no GRASP1.

Resultados empíricos mostram que este filtro na fase de construção, consegue na média melhorar o desempenho final do GRASP1. Embora tenha como contra partida um aumento no tempo computacional final exigido.

Na terceira versão, denotada por GRASP_DM, é sugerida a inclusão de um módulo de Mineração de Dados (*Data Mining*) no GRASP. A idéia do módulo de DM, é tornar o GRASP adaptativo, ou seja, incorporar um tipo de memória entre as iterações GRASP. Neste caso, a mineração tenta num conjunto das melhores soluções geradas até um dado momento pelo GRASP, gerar determinados tipos de padrões comuns nestas melhores soluções intermediárias. Descobertos estes padrões, estes são utilizados para gerar novas soluções nas iterações seguintes do GRASP. Os resultados das simulações efetuadas, mostram que esta alternativa é viável, decorrente da significativa melhora obtida nesta versão, quando comparada com as versões anteriores GRASP1 e GRASP_FILTRO.

Na versão seguinte, denominada GRASP_PR, é incluído o módulo de Reconexão de Caminhos (*Path Relinking*) no GRASP. O módulo de PR é uma outra forma de tornar as iterações GRASP adaptativas. Resultados computacionais efetuados mostram que esta versão híbrida se comporta melhor que o GRASP1 e o GRASP_FILTRO; e apresenta resultados similares ao GRASP_DM.

Finalmente, na quinta e última versão proposta, GRASP_DM_PR, é sugerido o uso em conjunto dos módulos de DM e PR numa estrutura GRASP. Esta versão híbrida foi a que obteve os melhores resultados nos testes efetuados, quando comparadas com as versões aqui propostas.

Em termos de comparações com as heurísticas da literatura, entretanto, o GRASP_DM_PR foi superado na média pelas heurísticas evolutivas apresentadas por Trindade e Ochi [38], e ficou um pouco abaixo do desempenho apresentado pelo algoritmo genético híbrido apresentado por Gonçalves e Resende [29]. Entretanto, vale aqui ressaltar, que tanto o GRASP_DM_PR como o GRASP_DM e o GRASP_PR, obtiveram resultados médios superiores ao obtidos pelos demais cinco métodos da literatura. Além disso, em 36 instâncias disponíveis na literatura, o GRASP_DM_PR conseguiu superar em dois casos os melhores resultados existentes até então.

Neste contexto, apesar dos resultados apresentado pelo GRASP_DM_PR não terem sido os melhores da literatura em todos os casos, acreditamos que o uso da metaheurística GRASP possa ser mais competitiva, por exemplo, se utilizarmos versões híbridas tais como: GRASP + VNS; GRASP + Busca Tabu; ou GRASP + *Iterated Local Search*. Que ficam como sugestões para trabalho futuros.

Outro aspecto que pode ser analisado em trabalhos futuros, é o desempenho das heurísticas aqui propostas, utilizando critérios de parada diferente do número máximo de iterações permitidas. Isso ficou evidente nos testes finais, onde versões mais sofisticadas tendem a possuir uma iteração mais onerosa. Porém se utilizássemos critérios de parada tais como: atingir um valor alvo ou um tempo limite de cpu, poderíamos chegar a conclusões de que versões mais sofisticadas como o GRASP_DM_PR podem exigir muito menos iterações para atingir determinados valores alvo.

Finalmente, uma outra contribuição futura poderia ser o uso destas heurísticas na solução de modelos alternativos de construção de células de manufatura onde existem restrições adicionais que os tornam modelos mais complexos.

Referências

[1] ALJABER, N., BAEK, W., AND CHEN, C. L. A tabu search approach to the cell formation problem. *Computers Industrial Engineering 1,32* (1997), 169 a 185.

[2] ASKIN, R., CRESSWELL, S. H., GOLDBERG, J. B., AND VAKHARIA, A. J. Hamiltonian path approach to reordering the part-machine matrix for cellular manufacturing. *International Journal of Production Research 29* (1991), 1081 a 1100.

[3] ATEME-NGHEMA, H., AND DAO, T. Optimum design of cellular manufacturing systems using hopfield neural network approach: the first step. *Department of Mechanical Engineering École de Technologie Supérieure, University of Quebec. Montreal, CANADA* (1997).

[4] BOCTOR, F. A linear formulation of the machine-part cell formation problem. *International Journal of production research 29(2) (1991), 343-356 29(2)* (1991), 343 a 356.

[5] CHANDRASEKHARAN, M. P., AND RAJAGOPALAN, R. M. An extension of rank order clustering for group technology. *International Journal of production research 5,24* (1986), 1221 a 1233.

[6] CHANDRASEKHARAN, M. P., AND RAJAGOPALAN, R. M. An algorithm for concurrent formation of part-families and machine cells. *International Journal of production research 6,25* (1987), 835 a 850.

[7] CHOOBINEH, F. A framework for the design of cellular manufacturing systems. *International Journal of production research 26(7)* (1988), 1161 a 1172.

[8] DIAZ, B. A., AND LOZANO, S. ANDRACERO, J. A. F. Machine cell formation in generalized group technology. *Computers Industrial Engineering 41* (2001), 227 a 240.

[9] FEO, T. A., AND RESENDE, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization 6* (1995), 109 a 134.

[10] FILHO, G. R., AND LORENA, L. A. N. A constructive evolutionary approach to the machine-part cell formation problem. Tech report, LAC/INPE - Instituto Nacional de Pesquisa Espaciais, 1998.

[11] GLOVER, F. Tabu search and adaptive memory programing - advances, applications and challenges. *Interfaces in Computer Science and Operations Research* (1996), 1 a 75.

[12] GONÇALVES, J. F., AND RESENDE, M. G. C. A hybrid genetic algorithm for manufacturing cell formation. Tech report, AT&T Labs Research Technical Report TD-5FE6RN, 2002.

[13] GUNASINGH, K., AND LASHKARI, R. Simultaneous grouping of parts and machines in cellular manufacturing systems - na integer programming approach. *Computer industrial engineering 20(1) (1990), 111-117 20(1)* (1990), 111 a 117.

[14] HARHALAKIS, G., HILGER, J., NAGI, R., AND PROTH, J. M. Formation of manufacturing cells: An algorithm for minimizing the intercell traffic. Tech report, University of Maryland., 1989.

[15] HARHALAKIS, G., PROTH, J. M., AND XIE, X. L. Manufacturing cell design using simulated annealing: na industrial application. Tech report, University of Maryland., 1990.

[16] JOGLECAR, J. A., CHUNG, Q., AND TAVANA, M. Note on a comparative evaluation of nine well-know algorithms for solving the cell formation problem in group technology. *Journal of Applied Mathematics & Decision Sciences* (2001), 253 a 268.

[17] JOINES, J. A., CULBRET, C. T., AND KING, R. E. Manufacturing cell desing: An integer programming model employing genetic algorithms. Tech report, Department of Industrial Engineering , North Carolina State University, 1996.

[18] KUMAR, K., AND VANNELI, A. Strategic subcontracting for efficient disaggregated manufacturing. *International Journal of Production Research 25(12)* (1987), 1715 a 1728.

[19] KUSIAK, A. The generalized group technology concept. *International Journal of production research 25(4)* (1987), 561 a 569.

[20] LAGUNA, M., AND MART, R. Grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing 11* (1999), 44 a 52.

[21] MAK, K. L., WONG, Y. S., AND WANG, X. X. An adaptive genetic algorithm for manufacturing cell formation. *The international Journal of Advanced Manufacturing Technology 16* (2000), 491 a 497.

[22] MALAKOOTI, B., AND YANG, Z. Multiple criteria approach and generation of efficient alternatives for machine-part family formation in group technology. *IIE Transactions 34* (2002), 837 846.

[23] MARTINS, S.L.AND RESENDE, M., RIBEIRO, C., AND PARDALOS, P. A parallel grasp for the steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization 17* (2000), 167 a 283.

[24] MUKATTASH, A. M., ADIL, M. B., AND TAHBOUB, K. K. A modified cell formation grouping measure. *Disponível em http://www.umoncton.ca/cie/Conferences/29thconf/29thICCIE/Papers/paper099.PDF* (2001).

[25] PARDALOS, P. M., QIAN, T., AND RESENDE, M. G. C. A greedy randomized adaptive search procedure for the feedback vertex set problem.pardalos p. m. , qian t. , resende m. g. c.journal of combinatorial optimization, 2:399-412, 1999. *Journal of Combinatorial Optimization 2* (1999), 399 a 412.

[26] PURCHECK, G. A linear programming method for the combinatorial grouping of an incomplete power set. *Journal of Cybernetics 5* (1975), 51 a 76.

[27] RAJAGOPALAN, R.AND BATRA, J. Design of cellular production systems: A graph-theoretic approach. *International Journal of production research 13(6) (1975), 567-579 13(6)* (1975), 567 a 579.

[28] RAVICHANDRAN, K. S., AND RAO, K. C. S. A new approach to fuzzy part-family formation in cellular manufacturing systems. *The international Journal of Advanced manufacturing Technology 18* (2001), 591 a 597.

[29] RESENDE, M. G. C., AND GONÇALVES, J. F. A hybrid genetic algorithm for manufacturing cell formation. Tech report, AT&T Labs Research Technical Report TD-5FE6RN, October 2002. Revised February 13,2004.

[30] RESENDE, M. G. C., AND RIBEIRO, C. C. Greedy randomized adaptive search procedures. Tech report, AT&T Labs Research Technical Report TD-53RSJY, 2002.

[31] SANTOS, H. S., AND OCHI, L. S. Grasp com mineração de dados para a solução de problemas de programação de horários em escolas de segundo grau no brasil. *TEMA-SBMAC* (2003).

[32] SANTOS, H. G.AND DRUMMOND, L. M. A. O. L. S. Grasp com mineração de dados para a solução do problema de programação de horários em escolas (resumo). *Anais do XXVI CNMAC da SBMAC*.

[33] SARKER, B. Measures of grouping efficiency in cellular manufacturing systems. *European Journal of Operational Research 130* (2000), 588 a 611.

[34] SHANKAR, R., VRAT, P., AND SOLEYMANPOUR, M. A transiently chaotic neural network approach to the design of cellular manufacturing. *International Journal of Production Research 10 , 40* (2002), 2225 a 2244.

[35] SRINIVASAN, G., AND NARENDRAN, T. Graphics - a non hierarquichal clustering clustering algorithm for group technology. *International Journal of production research 29(3)* (1991), 463 a 478.

[36] STEUDAL, H., AND BALLAKUR, A. A dynamic programming based heuristic for machine grouping in manufacturing cell formation. *Computer industrial engineering 12(4)* (1987), 215 a 222.

[37] SUN, D., AND LIN, L.AND, B. R. Cell formation using tabu search. *Computers Industrial Engineering 3,28* (1995), 485 a 494.

[38] TRINDADE, A. R., AND OCHI, L. S. An efficient evolutionary algorithm for the manufacturing cell design problem. *XII Latin-Ibero-American Congress on Operations Research (XII CLAIO), Havana 1* (2004).

[39] VAKHARIA, A. J., AND WEMMERLOV, U. A comparative investigation of hierarchical clustering techniques and dissimilarity measures applied to the cell formation problem. *Journal of Operations Management 13* (1995), 117 a 138.

[40] VANNELI, A., AND KUMAR, K. A method for finding minimal bottleneck cells for grouping part-machine families. *International Journal of Production Research 24(2)* (1986), 387 a 400.

[41] WANG, J. A linear assignment clustering algorithm based on the least similar cluster representatives. *IEEE Transactions on systems, Man and Cybernetics - Part A: Systems and Humans 1,29* (January 1999), 100 a 104.

[42] XAMBRE, A. R., AND VILARINHO, P. M. A simulated annealing approach for manufacturing cell formation with multiple identical machines. *European Journal of Operational Research 151* (2003), 434 a 446.