Clusterização em Mineração de Dados

Article · January 2004

CITATION
1

READS
381

3 authors, including:

Luiz Satoru Ochi
Universidade Federal Fluminense
169 PUBLICATIONS **1,657** CITATIONS
SEE PROFILE

Stenio Soares
Federal University of Juiz de Fora
2 PUBLICATIONS **1** CITATION
SEE PROFILE

Clusterização em Mineração de Dados

Luiz Satoru Ochi, Carlos Rodrigo Dias, Stênio S. Furtado Soares

Programa de Pós Graduação em Computação

Instituto de Computação – Universidade Federal Fluminense (IC – UFF)

Niterói, Rio de Janeiro, Brasil

satoru@ic.uff.br

Resumo. O objetivo deste trabalho, é apresentar uma contribuição na solução dos Problemas de Clusterização tratados na área de Mineração de Dados. Para tanto, apresentamos uma breve descrição do problema e posteriormente apresentamos algumas aplicações e técnicas eficientes para a sua solução aproximada. Em particular destacamos a classe de problemas de clusterização conhecida como Problema de Clusterização Automática e propostas de metodologias baseadas em conceitos de metaheurísticas evolutivas.

1. Introdução

Um ***Problema de Clusterização*** (PC) consiste em dado uma base de dados X, agrupar (*clusterizar*) os objetos (elementos) de X de modo que objetos mais similares fiquem no mesmo cluster e objetos menos similares sejam alocados para clusters distintos.

Existe basicamente duas classes de Problemas de Clusterização; o caso mais estudado é onde o número de clusters já é previamente definido (também conhecido como o *Problema de K - Clusterização* ou simplesmente *Problema de Clusterização (PC)*) é o caso onde este número K não é conhecido previamente, neste caso o Problema é denotado por *Problema de Clusterização Automática (PCA)* [8, 13].

Os Problemas de clusterização já são bastante estudados na literatura, principalmente na estatística e matemática. Na área de computação, este tema ressurgiu com a popularização do conceito de mineração de dados (*data mining*).

O objetivo de problemas de clusterização em mineração de dados (MD), é o mesmo que em outras áreas, o que diferencia este problema em MD, é que nesta área, a base de dados sempre é de grande porte e cada objeto normalmente contém um número elevado de atributos ou características [8].

Neste mini curso selecionamos algumas das aplicações de PCA de nosso interesse que inclui clusterização de células de um sistema de manufatura, problemas de escalonamento de tarefas em múltiplos processadores, problemas de roteamento e *scheduling* de veículos, aplicações em computação médica e biologia computacional.

Para algumas aplicações selecionadas, propomos novos algoritmos heurísticos ou metaheurísticos para a sua solução aproximada. Este trabalho apresenta na seção 2, uma descrição do problema de clusterização e algumas refêrencias deste tema encontradas na literatura, em 3 apresentamos algumas aplicações deste problema e que

estão sendo pesquisadas pelo nosso grupo de trabalho (LabIC), na seção 4, descrevemos resumidamente os algoritmos evolutivos e na seção 5 selecionamos algumas aplicações listadas anteriormente e para estas, são apresentadas novas propostas de métodos heurísticos. Finalmente a seção 6 apresenta as conclusões e a seguir são listadas as referencias bibliográficas.

2. O Problema de Clusterização

De uma forma geral, obter a solução para um problema de clusterização corresponde ao processo de agrupar os elementos (objetos) de uma base de dados (conjunto) de tal forma que os grupos formados, ou *clusters*, representem uma configuração em que cada elemento possua uma maior similaridade com qualquer elemento do mesmo *cluster* do que com elementos de outros *clusters*. As técnicas de clusterização vêm sendo tratadas com freqüência na literatura para a solução de vários problemas de aplicações práticas em diversas áreas do conhecimento. De uma forma mais formal, podemos definir Problemas de Clusterização da seguinte forma: Dado um conjunto com n elementos $X = \{X_1, X_2, ..., X_n\}$, o problema de clusterização consiste na obtenção de um conjunto de k *clusters*, $C = \{C_1, C_2, ..., C_k\}$, tal que os elementos contidos em um *cluster* C_i possuam uma maior similaridade entre si do que com os elementos de qualquer um dos demais *clusters* do conjunto C. O conjunto C é considerado uma *clusterização* com k *clusters* caso as seguintes condições sejam satisfeitas:

$$\bigcup_{i=1}^{k} C_i = X \tag{2.1}$$

$$C_i \neq \varnothing, \text{ para } 1 \leq i \leq k \tag{2.2}$$

$$C_i \cap C_j = \varnothing, \text{ para } 1 \leq i,j \leq k \text{ e } i \neq j \tag{2.3}$$

O valor de k pode ser conhecido ou não. Caso o valor de k seja fornecido como parâmetro para a solução, o problema é referenciado na literatura como "*problema de k-clusterização*" [15]. Caso contrário, isto é, caso o k seja desconhecido, o problema é referenciado como "*problema de clusterização automática*" e a obtenção do valor de k faz parte do processo de solução do problema, como em [13].

Em uma k-clusterização, o número total de diferentes formas de agrupamento de n elementos de um conjunto em k *clusters*, equivale à função $N(n, k)$ apresentada em (2.4).

$$N(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^i \binom{k}{i}(k-i)^n \tag{2.4}$$

Com o intuito de ilustrar o crescimento exponencial do número de soluções possíveis para um problema de *k*-clusterização, considerando a equação (2.4), para combinar 10 elementos em 2 *clusters*, 100 elementos em 2 *clusters*, 100 elementos em 5 *clusters* e 1000 elementos em 2 *clusters*, temos respectivamente os seguintes números de soluções possíveis: $N(10, 2) = 511$, $N(100, 2) = 6{,}33825 \times 10^{29}$, $N(100, 5) = 6{,}57384 \times 10^{67}$ e $N(1000, 2) = 5.3575 \times 10^{300}$.

Para o problema de clusterização automática o número total de combinações sofre uns incrementos significativos, sendo definido de acordo com a equação (2.5).

$$N(n) = \sum_{k=1}^{n} \frac{1}{k!} \sum_{i=0}^{k} (-1)^i \binom{k}{i} (k-i)^n \tag{2.5}$$

Dessa forma, para um conjunto com 10 elementos, a clusterização automática tem que considerar 115.975 diferentes maneiras de combinar os elementos em um número de *clusters* que pode variar de 1 a 10.

Outro aspecto a ser considerado em relação ao problema de clusterização é como medir o quanto um elemento é similar a outro e, assim, identificar se ambos devem estar contidos em um mesmo *cluster* ou não. Para isto deve ser utilizada uma "*medida de similaridade"*, que é específica para cada problema de clusterização a ser tratado.

Um importante critério utilizado para identificar a similaridade entre dois elementos é à *distância* entre eles, que trabalha com as diferenças entre os valores de cada atributo dos elementos. Neste caso, quanto menor for à distância entre um par de elementos maior é a similaridade entre eles. Como medidas de distância muito utilizadas podemos citar as seguintes:

- *distância euclidiana*: considera a distância *d* entre dois elementos X_i e X_j no espaço *p*-dimensional:

$$d(X_i, X_j) = \left[\sum_{l=1}^{p} (x_{il} - x_{jl})^2 \right]^{\frac{1}{2}} \tag{2.6}$$

- *distância* "*city-block*": corresponde à soma das diferenças entre todos os *p* atributos de dois elementos X_i e X_j, não sendo indicada para os casos em que existe uma correlação entre tais atributos:

$$d(X_i, X_j) = \sum_{l=1}^{p} \left| x_{il} - x_{jl} \right| \tag{2.7}$$

Existem problemas de clusterização em que a distância não pode ser utilizada, ou não é conveniente que seja utilizada, como medida de similaridade, tendo em vista que os valores dos atributos não são escalares. Como exemplo, ao tratar um problema de

clusterização que envolve atributos como sexo e endereço, são necessárias outras medidas que demonstrem o grau de similaridade entre as instâncias da base de dados.

Outro exemplo, em que a medida de distância não se aplica diz respeito a alguns problemas de clusterização de vértices em estruturas de grafos em que não são considerados os pesos das arestas. Nestes problemas, também referenciados como problemas de particionamento de grafos não ponderados, são necessárias, portanto, medidas que considerem apenas as conexões entre os seus vértices (veja [10, 11, 12, 13]).

Métodos Utilizados para Clusterização

No processo de clusterização, a busca pela melhor solução no espaço de soluções viáveis é um problema NP-Difícil. A partir das equações (2.4) e (2.5), conforme exposto anteriormente, verifica-se que a avaliação exaustiva de todas as configurações de clusterizações possíveis é computacionalmente inviável, restringindo com isso o uso de métodos exatos para a sua solução.

Dessa forma, métodos heurísticos ou aproximados têm sido propostos com freqüência, os quais fornecem soluções sub-ótimas com significativa redução da complexidade na solução do problema. Entretanto, devido à grande heterogeneidade das aplicações de problemas de clusterização, as heurísticas são normalmente desenvolvidas para determinadas classes de problemas, ou seja, não existe uma heurística que seja genérica a tal ponto que possa obter bons resultados em todas as aplicações de clusterização.

As heurísticas existentes para a solução de problemas de clusterização podem ser classificadas, de forma geral, em métodos hierárquicos e métodos de particionamento [15].

Nos algoritmos tradicionais para a clusterização hierárquica os *clusters* vão sendo formados gradativamente através de aglomerações ou divisões de elementos/*clusters*, gerando uma hierarquia de *clusters*, normalmente representada através de uma estrutura em árvore, conforme exemplificado na Figura 2.1. Nesta classe de algoritmos, cada *cluster* com tamanho maior que 1 pode ser considerado como sendo composto por *clusters* menores.

Nos algoritmos de aglomeração, que utilizam uma abordagem *bottom-up*, cada elemento do conjunto é, inicialmente, associado a um *cluster* distinto, e novos *clusters* vão sendo formados pela união dos *clusters* existentes. Esta união ocorre de acordo com alguma medida que forneça a informação sobre quais deles estão mais próximos uns dos outros. Nos algoritmos de divisão, com uma abordagem *top-down*, inicialmente tem-se um único *cluster* contendo todos os elementos do conjunto e, a cada passo, são efetuadas divisões, formando novos *clusters* de tamanhos menores, conforme critérios pré-estabelecidos.

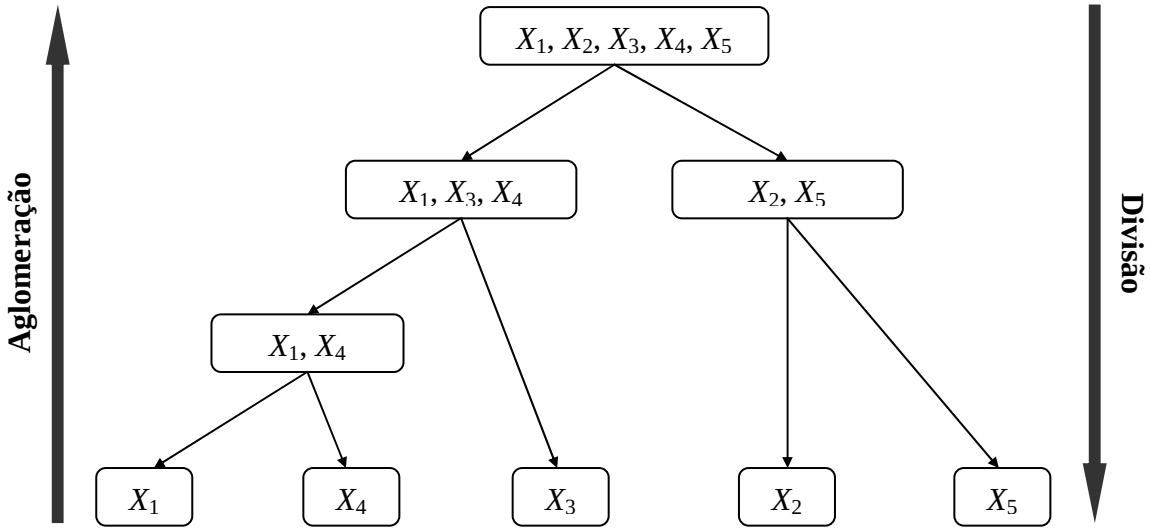

Figura 2.1 – Exemplo de árvore de *clusters* na clusterização hierárquica

Berkhin [8] aponta como vantagens dos algoritmos de clusterização hierárquica a facilidade em lidar com qualquer medida de similaridade utilizada e a sua conseqüente aplicabilidade a qualquer tipo de atributo (numérico ou categórico). As desvantagens relacionam-se à imprecisão do critério de parada e ao fato de que a maioria dos algoritmos desta classe não re-visitarem os *clusters* formados ao longo de suas execuções. Este último aspecto está relacionado ao fato dos algoritmos para clusterização hierárquica serem apenas algoritmos construtivos, não permitindo o refinamento de soluções obtidas durante a sua execução. Com relação ao critério de parada nos algoritmos de clusterização hierárquica, a formação dos *clusters* pode ser interrompida quando o número de *clusters* desejado for obtido, no caso de uma *k*-clusterização, ou caso alguma outra condição de parada ocorra. A falta de refinamento no processo de agrupamento ou desagregação normalmente fornece um caráter guloso ao método hierárquico tradicional.

Nos algoritmos de clusterização que utilizam algum método de particionamento, o conjunto de elementos é dividido em *k* subconjuntos, podendo *k* ser conhecido ou não, e cada configuração obtida é avaliada através de uma função-objetivo. Caso a avaliação da clusterização indique que a configuração não atende ao problema em questão, nova configuração é obtida através da migração de elementos entre os *clusters*, e o processo continua de forma iterativa até que algum critério de parada seja alcançado. Neste esquema de migração dos elementos entre os *clusters*, referenciado na literatura como *otimização iterativa* [8], os *clusters* podem ser melhorados gradativamente, o que não ocorre nos métodos hierárquicos.

Os métodos de particionamento para *k*-clusterização incluem ainda as técnicas *k-medoids* e *k-means*, de acordo com o tipo de representatividade utilizada para os *clusters*: no *k-medoids*, o elemento que melhor representa o *cluster*, é definido de acordo com seus atributos sem que haja muita influência dos valores próximos aos limites do *cluster*; no *k-means* o elemento representativo de um *cluster* é o seu centróide, que possui um valor médio para os atributos considerados, relativos a todos os elementos do cluster. A utilização do centróide como elemento representativo de um *cluster* é conveniente apenas para atributos numéricos e possui um significado

geométrico e estatístico claro podendo, entretanto, receber muita influência de um único elemento que se encontre próximo à fronteira do *cluster*.

Além dos métodos hierárquicos e de particionamento, é possível observar um crescimento significativo de propostas utilizando metaheurísticas aplicadas a problemas de clusterização, como os algoritmos evolutivos (AEs), com destaque para os algoritmos genéticos (AGs) [2, 5, 7, 9, 10, 11, 12, 13, 18, 19, 25, 26, 38, 39, 40].

3. Algumas aplicações do Problema de Clusterização

O problema de clusterização possui aplicações nas mais variadas áreas de pesquisa incluindo por exemplo: computação visual e gráfica, computação médica, biologia computacional, redes de comunicações, engenharia de transportes, redes de computadores, sistemas de manufatura, entre outras. Nesta seção apresentamos algumas destas aplicações que atualmente estão sendo pesquisadas pelo grupo de Inteligência Computacional do IC/UFF denominado: " Laboratório de Inteligência Computacional – LabIC)".

3.1. Clusterização de Grafos

O problema de clusterização aplicado a grafos, também referenciado na literatura como "problema de particionamento de grafos", consiste em, dado um grafo $G = (V, E)$, com V sendo o conjunto de vértices e E o conjunto de arestas, particionar o conjunto de vértices em subconjuntos disjuntos, ou *clusters*, otimizando alguma função-objetivo. No problema de particionamento *balanceado* de grafo a diferença de cardinalidade entre o maior *cluster* e o menor *cluster* deve ser de, no máximo, uma unidade. Quando o número de *clusters* é igual a dois, o problema é referenciado na literatura como problema de bisseção de grafos ou problema de bi-particionamento de grafos [4]. Em [5] o problema de particionamento de grafos é considerado como consistindo do particionamento do conjunto de vértices do grafo em *k clusters* disjuntos e com a mesma cardinalidade, tal que o total de arestas da qual participam pares de vértices de diferentes *clusters* seja minimizado. A Figura 3.1 apresenta um exemplo de bi-particionamento balanceado de um grafo constituído de 14 vértices e 27 arestas.

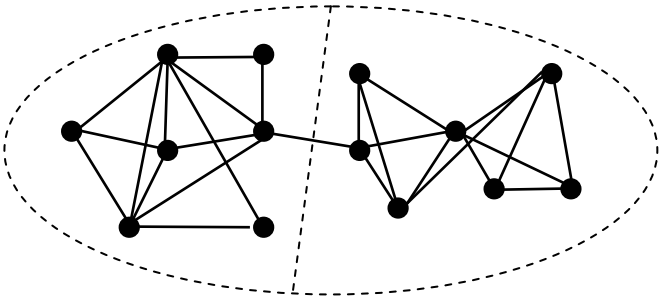

Figura 3.1 – Exemplo de bi-particionamento balanceado de grafos

O problema de particionamento de grafos é NP-Difícil, mesmo com o valor de k igual a dois ou quando algum desbalanceamento é permitido. Segundo Battiti et al. [5], para grafos com mais de 100 vértices, as únicas opções viáveis são os algoritmos heurísticos.

O problema de clusterização de grafos tratado neste trabalho com mais destaque corresponde a clusterização automática considerando grafos orientados e sem valores de peso nos arcos (grafos não ponderados). Neste problema, o objetivo da clusterização é agrupar os vértices do grafo em *clusters* de tal forma que seja maximizado o número total dos arcos internos a cada *cluster*, ao mesmo tempo em que seja minimizado o número total de arcos entre pares de vértices que estejam em diferentes *clusters* [13]. Nesse contexto, a Figura 3.2 exemplifica duas clusterizações possíveis para um grafo com 8 vértices e 12 arcos.

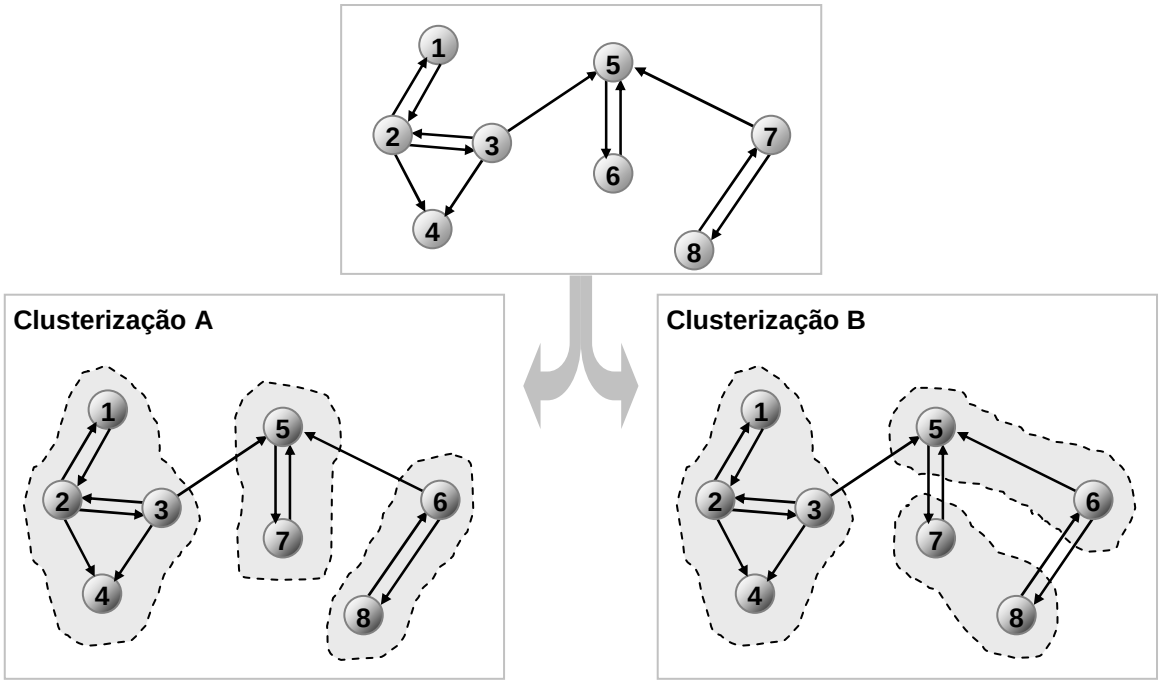

Figura 3.2 – Exemplos de clusterização de um grafo orientado com 8 vértices [12]

Em cada uma das duas clusterizações indicadas na Figura 3.2 cada *cluster* é delimitado por uma linha tracejada. De forma intuitiva podemos considerar a clusterização A mais adequada do que a clusterização B, tendo em vista que nesta existe um *cluster* sem qualquer arco interno e existe também um número maior de arcos entre diferentes *clusters* do em A. Para maiores informações sobre este problema, consulte [12]

3.2. Clusterização em Sistemas de Manufatura

O problema de formação de células de manufatura (PFCM) na sua versão original é representado como uma matriz *"parte x máquina"* onde as linhas representam as partes e as colunas às máquinas, ou vice-versa. Considerando aqui uma matriz A = (*parte x máquina)*, cada célula a_{ij} da matriz é igual a 1 se à *parte i* utiliza a máquina *j*, e 0 caso contrário. A formação dos grupos (clusters) "célula / família" é feita através da permutação das linhas e colunas desta matriz.

Por exemplo, considere um fluxo de produção composto por 6 *partes*, 4 máquinas e 14 atividades (posições da matriz com valor 1). As tabelas 3.3 e 3.4 representam respectivamente a matriz de entrada do problema e uma possível matriz solução com formação de duas células/*famílias*.

	M1	M2	M3	M4
P1		1	1	
P2	1	1		1
P3		1	1	
P4	1			1
P5	1	1		
P6		1	1	1

Tabela 3.3: Matriz de entrada do um PFCM [40]

	M2	M3	M1	M4
P1	1	1		
P3	1	1		
P6	1	1		1
P2	1		1	1
P4			1	1
P5	1		1	

Tabela 3.4: exemplo de uma clusterização da matriz da Tabela 3.3 [40]

Na matriz da tabela 3.4 existem duas *famílias*: [P1,P3,P6] e [P2,P4,P5]. Associadas a estas *famílias* temos 2 células: [M2,M3] e [M1,M4]. Para maiores informações sobre este problema, consulte [40]

3.3 Problemas de Escalonamento de Tarefas em Múltiplos Processadores

O problema de escalonamento estático ou dinâmico de um conjunto de tarefas de uma aplicação paralela em um conjunto de processadores (homogêneos e totalmente conectados), pode também ser visto como um problema de clusterização. Neste caso, o conjunto de tarefas alocadas a um processador forma um cluster. No modelo aqui abordado, cada um desses processadores possui sua própria memória local e a comunicação entre eles é feita exclusivamente através de troca de mensagens, o que caracteriza um sistema distribuído. Por tarefas não-preemptivas, podemos entender que a execução de cada tarefa não pode ser interrompida por nenhum outro evento, ou seja, uma vez iniciada a execução de uma tarefa, o processador só será liberado ao terminar tal execução.

Dada uma aplicação paralela a ser executada num sistema com p processadores, o objetivo do problema aqui apresentado é encontrar um escalonamento que *minimize* o tempo total de execução (*makespan*) da aplicação. Este escalonamento é feito de maneira que todas as relações de precedência entre as tarefas sejam respeitadas. Além disso, uma tarefa v_i só pode ser escalonada se todos os seus *predecessores imediatos* já

tiverem sido e todos os dados necessários para a sua execução já estiverem disponíveis no processador p_j onde v_i será escalonada. Para a solução deste problema, as características da aplicação e da arquitetura utilizada devem ser bem definidas. Tais características são especificadas pelo *modelo de escalonamento* que é composto pelo *modelo de aplicação* e pelo *modelo arquitetural*, descritos a seguir.

3.3.1 – Modelo de Aplicação

Uma aplicação paralela pode ser representada por um Grafo Acíclico Direcionado (*GAD*) denotado por $G = (V,E,\varepsilon,\omega)$, onde V é o conjunto de n nós do grafo e E o conjunto de arcos. Cada nó $v \in V$ representa uma tarefa com tempo de execução $\varepsilon(v)$ e cada arco $(u,v) \in E$ representa a restrição de precedência entre as tarefas u e v, ou seja, a tarefa u deve completar sua execução antes que a tarefa v comece a sua. Um peso $\omega(u,v)$ pode estar associado ao arco (u,v), representando a quantidade de dados a serem enviados de u para v. Uma tarefa consiste numa unidade de computação indivisível que pode ser uma instrução, uma sub-rotina ou um programa inteiro. O conjunto das tarefas predecessores imediatas à tarefa $v \in V$ é denotado por $pred\ (v) = \{u \mid (u,v) \in E\}$ enquanto que o conjunto de seus sucessores imediatos é dado por $succ\ (v) = \{z \mid (v,z) \in E\}$.

3.3..2 – Modelo Arquitetural

O modelo arquitetural define as características da arquitetura paralela a ser considerada. Neste trabalho, considera-se um computador paralelo com memória distribuída, sendo cada processador associado à sua própria memória local. Os processadores se comunicam exclusivamente através de troca de mensagens e estão interconectados conforme alguma topologia. Além disso, considera-se que os processadores estão totalmente conectados entre si e são homogêneos, isto é, possuem as mesmas características. Atualmente, existe uma grande variedade de máquinas paralelas em uso. Tais máquinas possuem características particulares que as diferem uma das outras, como por exemplo, o tipo de acesso à memória ou o tipo de interconexão entre os processadores. O ideal ao se criar uma aplicação paralela é que o programador não se preocupe com detalhes da máquina e, para isso, é importante a criação de um *modelo de computação paralela* suficientemente abstrato para que detalhes da máquina sejam ignorados, mas que seja, ao mesmo tempo, versátil para permitir que a estrutura computacional do programa possa ser mapeada eficientemente para uma grande variedade de plataformas paralelas. O primeiro modelo de computação paralela definido foi o *PRAM* . Neste modelo assume-se que o número de processadores é ilimitado e que eles trabalham de forma síncrona. Além disso, a comunicação entre os processadores pode ser considerada nula já que eles se comunicam através de uma memória compartilhada. Outra característica deste modelo é o fato de permitir que vários processadores acessem simultaneamente a memória, sem que esta se torne um gargalo, nem quando o número de processadores que a compartilham for muito elevado. Devido a essas características, o modelo *PRAM* não é considerado realístico e, portanto, algoritmos desenvolvidos para este modelo possuem na maioria dos casos desempenhos ruins quando são assumidas características que representam máquinas paralelas reais.

Com o desenvolvimento das máquinas paralelas distribuídas, fez-se necessário à criação de modelos de computação paralela que representassem estas máquinas de

forma mais realística e que definissem as características de comunicação com mais precisão. Neste cenário, surgiu o modelo de latência, onde o único parâmetro de comunicação considerado é a latência (*delay*), que consiste no tempo de transferência de uma unidade de dado entre dois processadores distintos. Um outro modelo, proposto recentemente, é o modelo LogP onde, além da latência, também são considerados outros importantes parâmetros de comunicação que tornam este modelo o mais realístico atualmente. A seguir, são descritos mais detalhadamente os modelos de latência e LogP, que consistem nos modelos de computação paralela mais considerados na área de escalonamento de tarefas.

→ Modelo de Latência

Neste modelo, o único parâmetro arquitetural associado ao custo de comunicação é a latência, denotada por τ. Assume-se que um processador não gasta tempo preparando o envio nem o recebimento de mensagens permitindo, dessa forma, que comunicação e computação se sobreponham totalmente. Uma vez que um processador não gasta tempo preparando o envio de mensagens, ele pode enviar simultaneamente várias mensagens distintas para vários destinos

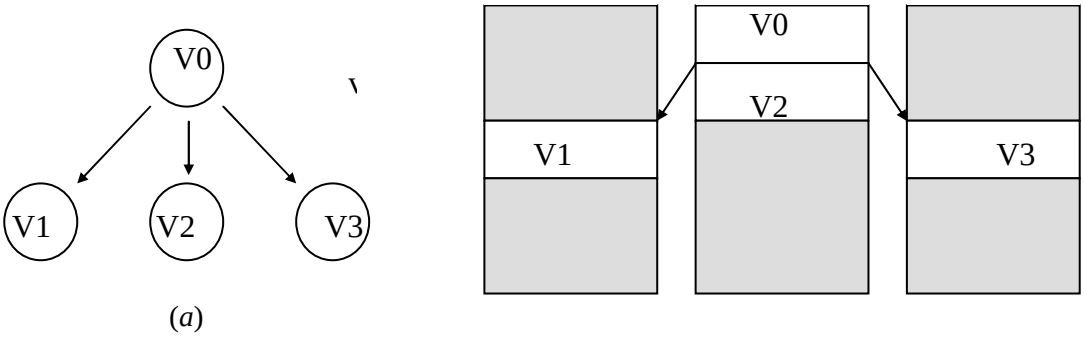

Figura 3.3.1: (a) Grafo do tipo fork. (b) Possível escalonamento [42]

(*multicast*), como ilustrado na Figura 3.3.1 (b), onde a tarefa v_0 envia, simultaneamente, mensagens para as tarefas v_1 e v_3. Como podemos observar, o envio de tais mensagens se sobrepõe com a execução da tarefa v_2.

→ Modelo LogP

O modelo LogP representa o fato de que nas máquinas paralelas atuais, o processador deve tratar ou ao menos iniciar cada comunicação e que, além disso, comunicação e computação não podem ser sobrepostas totalmente. O próprio nome dado ao modelo já define os parâmetros considerados, ou seja:

- *L* – latência de comunicação.

- *o* – sobrecarga (*overhead*), que é o tempo durante o qual o processador permanece preparando o recebimento (*sobrecarga de recebimento*) ou o envio

(*sobrecarga de envio*) de uma mensagem, tempo este durante o qual o processador não pode realizar outras operações.

- *g – gap* que é o intervalo mínimo permitido entre dois envios ou dois recebimentos consecutivos em um mesmo processador. Este parâmetro é uma característica da rede e está associado à capacidade do canal de saída do processador.

- *P* – conjunto dos processadores disponíveis.

Sempre que uma tarefa $v_i \in V$ for escalonada num processador $p_j \in P$ no qual um de seus predecessores imediatos não se encontra, duas tarefas extras deverão ser escalonadas: uma tarefa de envio e uma de recebimento, como ilustrado na Figura 3.3.2 Uma tarefa de envio deve ser escalonada imediatamente após cada predecessor de v_i que não se encontra em p_j. Para cada uma dessas tarefas de envio, deve estar associada uma

tarefa de recebimento que será escalonada em p_j antes de v_i. Tais tarefas representam a sobrecarga gasta pelo processador para o envio ou o recebimento de uma mensagem.

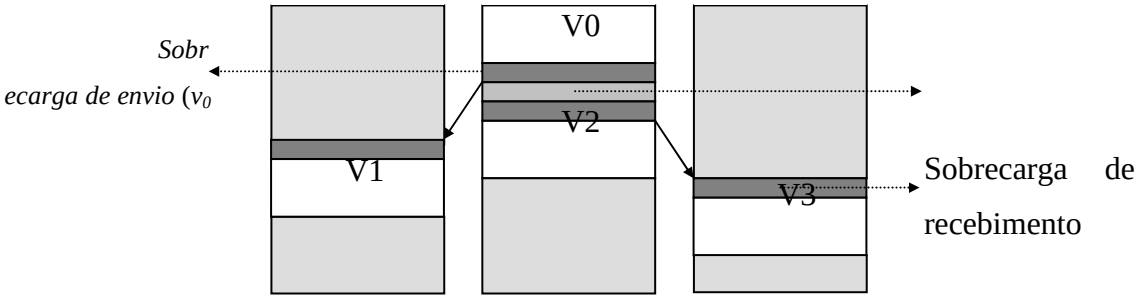

Figura 3.3.2: Possível escalonamento para a aplicação da Figura 1(a), utilizando o modelo LogP

Alguns autores preferem utilizar os termos *tempo de empacotamento* e *tempo de desempacotamento* para se referirem ao tempo de preparação do envio e do recebimento de uma mensagem, respectivamente. Como podemos ver, esse modelo é mais realístico do que o modelo de latência pelo fato de também considerar outros importantes parâmetros de comunicação (veja maiores detalhes em [42]).

3.4 O Problema de roteamento e scheduling periódico de uma frota de veículos

O problema de roteamento periódico básico consiste de uma frota homogênea de veículos que deve atender a um conjunto de clientes a partir de uma origem (depósito) de onde os veículos devem sair e retornar ao final da jornada. Cada veículo possui uma capacidade fixa que não pode ser excedida e cada cliente possui uma demanda conhecida que deve ser totalmente satisfeita numa única visita por um único veículo. O período de planejamento é de T dias. Quando T=1, o PRPV se restringe ao clássico Problema de Roteamento de Veículos (PRV). Cada cliente no PRPV deve ser visitado k vezes, onde $1 \leq k \leq T$ e no modelo clássico, a demanda diária de um cliente é sempre igual para cada dia de visita. O objetivo do PRPV pode ser visto como a de gerar um conjunto de rotas para cada dia de modo que as restrições envolvidas sejam atendidas e os custos globais minimizados. Tanto o PRV como o PRPV podem ser vistos como

Problemas de Clusterização, neste caso, os clientes alocados a uma rota definem um cluster. Existem várias generalizações do PRPV básico incorporando variações e/ou condições adicionais. Os mais comuns são:

- A frota de veículos pode ser heterogênea (capacidades e/ou custos distintos).

- Cada rota diária de um veículo pode ser limitado em função da sua distância e/ou tempo.

- O número de veículos disponíveis a cada dia pode ser limitado e fixado previamente ou ser variável.

- A demanda diária de um cliente pode ser variável e atendida em mais de uma visita.

- Pode existir mais de uma origem (depósito).

- Os clientes podem ter exigências do tipo: *time-windows* onde devem ser visitados, restrições de precedências entre dois clientes. Para maiores informações sobre este problema, consulte [31, 32, 33, 34, 35].

Como neste trabalho, apresentamos diferentes propostas de metaheuristicas usando conceitos de algoritmos genéticos (AGs) e algoritmos evolutivos (AEs) mostramos a seguir uma breve revisão destes conceitos.

4. Algoritmos Evolutivos

Como todas as contribuições propostas neste trabalho enfocam o conceito de algoritmos evolutivos, vamos inicialmente dar um breve resumo sobre eles enfocando principalmente o seu representante mais popular que são os algoritmos genéticos. A expressão “Algoritmos Evolutivos” (ou a sua variação, “Algoritmos Evolucionários”) corresponde à classe de algoritmos para a solução de problemas de otimização que utilizam modelos computacionais baseados na teoria da evolução das espécies, proposta por Charles Darwin, e nos princípios básicos da herança genética, descritos por Gregor Mendel [6, 20, 21, 27]. A partir de muitas evidências colhidas em suas viagens a bordo do navio Beagle, Charles Darwin publicou em 1859 a sua teoria sobre a evolução dos seres vivos através da seleção natural. No seu trabalho, intitulado “*The Origin of Species*”, Darwin propõe um modelo de evolução em que uma população de indivíduos sofre um processo de evolução natural e estes são capazes de se adaptarem ao ambiente em que vivem através de processos de seleção natural, reprodução, recombinação sexual e mutação, onde os indivíduos mais adaptados têm maiores chances de sobreviverem e gerarem descendentes. Segundo Bäck [3], o processo de seleção natural, que privilegia os indivíduos com alta capacidade de sobrevivência (mais adaptados ao meio ambiente, ou mais aptos), permite que a qualidade média da população melhore ao longo do processo evolutivo, levando à obtenção de um indivíduo totalmente adaptado ao ambiente, o indivíduo “ótimo”.

No processo de evolução natural, a combinação da teoria da evolução com a genética fornece um mecanismo que permite o surgimento e a adaptação de seres vivos ao meio-ambiente, através de uma busca no imenso espaço de todas as possíveis combinações de seqüências de DNA. Os Algoritmos Evolutivos (AEs) utilizam estas idéias através da

manipulação de uma população de indivíduos (soluções) que evoluem ao longo de várias iterações do AE, chamadas de “gerações”.

Os AEs são divididos, de uma forma geral, nos seguintes grupos:

- *Estratégias de Evolução*: na proposta original, apresentada por Schwefel, não existe seleção de indivíduos para constituição da população da geração seguinte – cada indivíduo de uma população gera um único filho, através da aplicação do operador mutação [27].

- *Programação Evolutiva*: foi desenvolvida para evoluir máquinas de estados finitos através de um processo semelhante às Estratégias de Evolução **Erro! A origem da referência não foi encontrada.**.

- *Algoritmos Genéticos***:** os princípios básicos dos Algoritmos Genéticos (AGs) foram apresentados inicialmente por Holland [21].

- *Programação Genética*: é um método utilizado para a evolução de programas de computador, inicialmente proposto por Koza [23] como uma aplicação dos AGs na evolução de estruturas em árvore.

4.1.1 Algoritmos Genéticos – Versão Clássica

Os Algoritmos Genéticos (AGs) foram propostos por Holland [21] como sendo algoritmos de busca de propósito geral, com características de busca estocástica, busca de múltiplos pontos e busca paralela. Em seu trabalho, Holland estava mais interessado na evolução dos indivíduos de uma população, em uma tentativa de explicar os processos adaptativos em sistemas naturais para desenvolver sistemas artificiais baseados nestes processos, do que em resolver problemas de otimização.

O comportamento dos AGs corresponde a uma analogia com o comportamento dos indivíduos de uma população na natureza. Considerando uma população de indivíduos da natureza, estes competem entre si por diferentes recursos disponíveis no seu meio ambiente (*habitat*), como água, comida e abrigo. Cada um destes indivíduos possui características externas (fenótipo), relacionadas à sua constituição genética (genótipo), que os diferem entre si em relação à adaptação ao meio ambiente em que vivem. Esta adaptação afeta diretamente a capacidade de sobrevivência por período suficiente para se reproduzirem pelo acasalamento. Através do acasalamento, as características genéticas dos dois indivíduos envolvidos são combinadas e transmitidas para a prole. Dessa forma as gerações futuras possuem uma grande probabilidade de serem formadas por indivíduos com as características necessárias para um maior tempo de vida, em relação às gerações anteriores – a este processo é dado o nome de evolução natural. Para facilitar a descrição e utilização dos AGs, a terminologia utilizada na Biologia é adotada naturalmente.

O “fenótipo” de um indivíduo é obtido a partir da sua submissão a uma função que irá avaliar a qualidade do seu “código genético” e, dessa forma, corresponde às suas chances de gerar descendentes. Esta função, chamada de função de aptidão, é uma codificação da função-objetivo do problema e define a qualidade de cada indivíduo em relação ao problema modelado. Assim como na evolução natural, num AG deve haver

maiores chances de que os códigos genéticos dos indivíduos mais aptos sejam transmitidos para as gerações futuras através do processo seleção “natural” e reprodução.

Uma característica importante de um AG é a utilização dos “operadores genéticos” sobre os indivíduos da população para que possam ser exploradas diferentes áreas do espaço de busca evitando, assim, uma convergência do algoritmo para uma solução ótima local. A combinação entre partes do código genético de diferentes indivíduos (através do operador de cruzamento) e a realização de pequenas alterações genéticas (através do operador mutação) permitem a exploração de novas características, que podem corresponder a uma evolução dos indivíduos. Dessa forma, a população de indivíduos tende a convergir para uma combinação de características dos indivíduos que seja ideal para o problema em questão – a solução ótima. Este mecanismo de evolução natural de soluções permite que os AGs possam ser utilizados para a solução de quase todos os problemas de otimização. Beasley, Bull e Martin afirmam que “Se o AG foi implementado corretamente, a população irá evoluir ao longo de sucessivas gerações de tal forma que a aptidão do melhor indivíduo e do indivíduo médio em cada geração será incrementada em direção ao ótimo global. A *convergência* é a progressão em direção à uniformidade crescente.” [6].

Segundo Goldberg [20], apesar da convergência global da população não ser garantida em um tempo finito, os AGs provaram ser uma técnica de busca robusta para a maioria das aplicações reais. A Figura 4.1 apresenta a estrutura básica de um AG tradicional.

Algoritmo Genético Tradicional

```
1.  i = 0;
2.  gerar a população inicial P(0);
3.  avaliar a população inicial P(0);
4.  enquanto não(condição de término) faça
5.      i = i + 1;
6.      selecionar P(i) de P(i – 1);
7.      aplicar os operadores genéticos a P(i);
8.      avaliar os indivíduos de P(i);
9.  fim-enquanto.
```

Figura 4.1 – Algoritmo Genético tradicional

Com base no exposto, para que a utilização do AG alcance sucesso na obtenção da solução para um problema específico, na modelagem do AG é necessário considerar:

1. representação dos indivíduos: como representar as possíveis soluções para o problema;

2. função de aptidão: de que forma a função de aptidão pode representar, de forma precisa, a qualidade de cada solução obtida;

3. seleção e reprodução: como será realizada a seleção dos indivíduos de uma geração para constituírem a população da geração seguinte;

4. operadores genéticos: quais operadores genéticos devem ser aplicados, e de que forma;

5. outros parâmetros: quais os valores que devem ser utilizados para o tamanho da população, taxa de aplicação dos operadores genéticos, critério de parada, etc.

Nas próximas subseções, serão realizadas algumas considerações relativas às características de um AG tradicional indicadas acima.

Representação dos Indivíduos

De uma forma geral, uma solução potencial para um problema de otimização pode ser representada como um conjunto de valores de parâmetros para o problema modelado. A representação, ou codificação, de um indivíduo em um AG deve concatenar todos estes parâmetros (chamados de genes), formando uma cadeia de valores que corresponde a uma solução para o problema.

1. a representação dos indivíduos deve ser completa: a representação deve permitir que todos os parâmetros da solução possam assumir todos os valores possíveis no domínio do problema. Caso o AG não consiga representar uma determinada solução, ele nunca poderá obtê-la. Dessa forma, o "espaço de busca" de um AG corresponde a todas as configurações possíveis que um indivíduo pode assumir e deve ser igual ao espaço de busca do problema;

2. a representação deve ser válida: deve-se evitar que possam ser geradas soluções fora do espaço de busca do problema. Caso isto ocorra, deve haver algum mecanismo no AG para tornar a representação válida;

3. evitar a simetria na codificação: a simetria gera redundância, ou seja, indivíduos diferentes podem representar uma mesma solução. Isto pode trazer problemas para a convergência da população. Um exemplo de codificação com simetria é a codificação *group-number*, utilizada em problemas de clusterização e tratada na Seção 3.2.1.

Função de Aptidão

A função de aptidão normalmente corresponde à função-objetivo do problema modelado e fornece um valor que permite avaliar a qualidade da solução representada pelo indivíduo em relação às demais soluções do espaço de busca. Por utilizar uma função que avalia a aptidão dos indivíduos de uma população durante a sua evolução, os Ags podem ser aplicados a vários problemas de otimização, bastando utilizar uma função de aptidão específica para o problema. A função de aptidão deve ser implementada de forma que seja executada de maneira relativamente rápida. Além disso, a execução da função de aptidão nos Ags pode necessitar de quantidade de tempo considerável tendo em vista que, a cada iteração, é necessário avaliar uma população

inteira de soluções potenciais (os indivíduos), e não apenas uma solução, como acontece em outras técnicas de otimização.

Seleção e Reprodução

A cada iteração, ou geração, de um AG, partindo-se de uma população de *p* indivíduos da geração anterior deve ser obtida uma nova população de *p* indivíduos, sobre a qual serão aplicados os operadores genéticos. Este processo envolve os passos de seleção e reprodução dos indivíduos. A partir dos indivíduos escolhidos no processo de seleção, a reprodução consiste em copiar o código genético dos indivíduos selecionados para a nova população.

Tendo em vista que, ao término de uma iteração, todos os indivíduos são avaliados , é possível saber quais deles possuem qualidades superiores aos demais, como solução para o problema modelado. Dessa forma, a seleção de indivíduos deve corresponder a um mecanismo que permita a sobrevivência dos melhores indivíduos para que estes possam compartilhar suas características com as gerações seguintes.

Existem várias estratégias para a seleção. Beasley, Bull e Martin [6] afirmam que não existe um método considerado absoluto, tendo em vista que os ajustes nas estratégias podem levar a desempenhos semelhantes.

As duas estratégias mais utilizadas para a seleção são descritas a seguir:

1. seleção proporcional à aptidão: criada por Holland [21], esta estratégia é inspirada na seleção natural, que envolve a seleção dos indivíduos conforme o valor da sua aptidão. Uma implementação desta estratégia é conhecida como “método da roleta”, que se utiliza uma analogia com o jogo de roleta encontrado em cassinos. No método da roleta cada indivíduo possui uma região da roleta proporcional ao valor da sua aptidão e assim uma determinada probabilidade de ser selecionado. Cada vez que a roleta é girada um indivíduo é selecionado. O número total de vezes que a roleta é girada correspondente ao tamanho da população, podendo um mesmo indivíduo ser selecionado mais de uma vez;

2. seleção por torneio: existem muitas variações desta estratégia. A mais simples corresponde a, para a seleção de cada indivíduo, inicialmente escolher de forma aleatória *t* indivíduos da população. Em seguida os indivíduos escolhidos competem entre si e o indivíduo selecionado será aquele que possuir o melhor valor para a aptidão. Normalmente *t* é utilizado com o valor 2, e um aumento neste valor irá acelerar a convergência da população. Outro aspecto a ser considerado no processo de seleção é o elitismo, que é uma estratégia utilizada em conjunto com as outras já descritas para a seleção. O elitismo é utilizado para garantir que o melhor indivíduo da população de uma geração seja reproduzido na população da geração seguinte. Segundo Beasley, Bull e Martin [6] o processo de seleção influencia muito o comportamento de um AG, sendo um dos aspectos críticos para que a evolução da população ocorra. O tipo de seleção utilizado pode produzir problemas como a convergência prematura para ótimos locais distantes de um ótimo global ou o caso oposto, onde a convergência é muito lenta.

A convergência prematura ocorre quando as características de indivíduos com uma elevada aptidão (mas que não correspondem à solução ótima) dominam rapidamente a

população, que converge para um ótimo local de baixa qualidade. A convergência lenta, ou a não convergência, é um problema oposto à convergência prematura, em que ótimos locais de boa qualidade, ou um ótimo global, podem também não serem alcançados. Neste caso o valor médio da aptidão normalmente é alto e a diferença entre o melhor indivíduo e o valor médio é muito pequena. Dessa forma, não existe diversificação suficiente na população para que o AG consiga continuar a sua evolução em direção à solução ótima.

Operadores Genéticos

Após a seleção e reprodução dos indivíduos de uma geração, é iniciada uma nova geração utilizando uma população intermediária, sobre a qual serão aplicados os operadores genéticos. Durante a aplicação dos operadores genéticos, ao mesmo tempo em que são gerados novos indivíduos, algumas características de adaptação dos indivíduos das gerações anteriores são mantidas. Os operadores genéticos normalmente utilizados em um AG tradicional são o operador de cruzamento e o operador mutação.

Segundo Goldberg [20], o operador de cruzamento é a principal força direcionadora em um AG. O operador de cruzamento realiza a troca de partes de pares de indivíduos com o objetivo de tentar obter indivíduos melhores a partir dos indivíduos selecionados. Dessa forma, o principal objetivo do cruzamento é utilizar o conhecimento obtido em pontos do espaço de busca visitados previamente. A aplicação do operador de cruzamento a um par de indivíduos normalmente está sujeita a uma taxa de probabilidade de aplicação, definida como parâmetro para a execução do AG. A partir da seleção de um par de indivíduos, existem diversas formas de utilização do operador de cruzamento. As mais comuns são:

1. cruzamento de um ponto: a partir de um ponto de cruzamento dos indivíduos envolvidos, obtido de forma aleatória, os valores constantes dos trechos situados após o ponto de cruzamento são trocados entre os indivíduos do par. Um exemplo é apresentado na Figura 4.1.1;

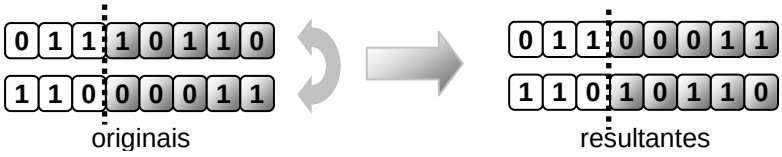

Figura 4.1.1 – Exemplo de aplicação do operador de cruzamento de um ponto

2. cruzamento de múltiplos pontos: é uma generalização do cruzamento de um ponto, em que pares de pontos de cruzamento são obtidos de forma aleatória e os valores dos indivíduos localizados entre cada par de pontos de cruzamento são trocados. Um exemplo é apresentado na Figura 4.1.2;

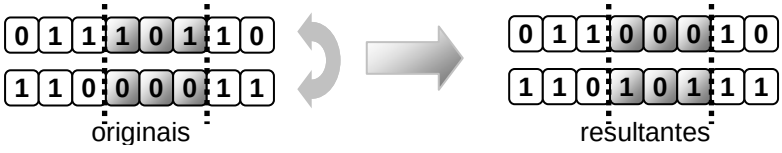

Figura 4.1.2 – Exemplo de aplicação do operador de cruzamento de múltiplos pontos

3. cruzamento uniforme: uma máscara de dígitos binários é obtida de forma aleatória, onde o dígito 1 indica que o valor na respectiva posição dos indivíduos deverá ser trocado e o dígito 0 indica que os valores originais das posições dos indivíduos envolvidos, equivalentes às posições da máscara, devem ser mantidos com os valores originais. Um exemplo é apresentado na Figura 4.1.3.

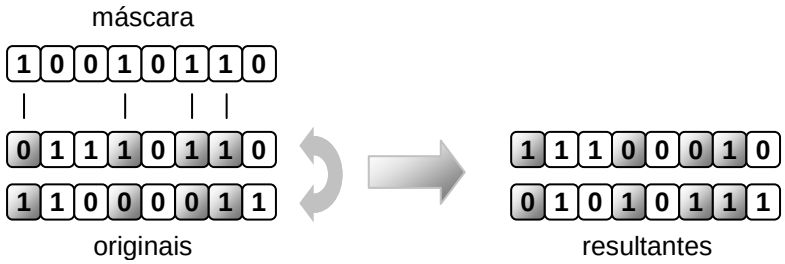

Figura 4.1.3 – Exemplo de aplicação do operador de cruzamento uniforme

Outros tipos de operadores de cruzamento podem ser necessários dependendo do problema modelado. Diferentemente do operador de cruzamento, o operador mutação realiza trocas aleatórias de alguns valores dos indivíduos, com o intuito de pesquisar novas áreas do espaço de busca, a partir de indivíduos selecionados. Ao permitir a manutenção da diversidade genética da população, o operador mutação evita que a população fique estagnada em uma região de ótimo local. A Figura 4.1.4 apresenta um exemplo de aplicação do operador mutação em um indivíduo representado por uma cadeia de dígitos binários, em que os valores dos elementos selecionados, em destaque, são trocados.

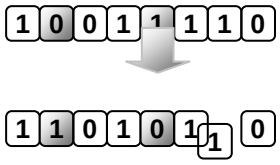

Figura 4.1.4 – Exemplo de aplicação do operador mutação

O operador mutação também está sujeito a uma taxa de aplicação que deve ser definida conforme características do problema. A taxa de aplicação do operador mutação refere-se à probabilidade de cada um dos elementos de um indivíduo sofrer a mutação. Caso a taxa de aplicação seja próxima a 100%, a busca se torna efetivamente aleatória. O parâmetro taxa de mutação é, portanto, determinante para o grau de convergência da população durante a execução de um AG. Segundo Beasley, Bull e Martin [6], a definição dos operadores de cruzamento e mutação, bem como as suas respectivas taxas de aplicação, podem ser determinantes para a ocorrência dos problemas de convergência. Os problemas de convergência originados da definição incorreta das taxas de aplicação do operadores genéticos podem, por exemplo, eliminar a possibilidade de exploração do espaço de busca pelo operador de cruzamento, ficando apenas o operador mutação responsável pela exploração que, por isto, pode se tornar lenta e aleatória.

Outros Parâmetros

Em um AG tradicional existem vários outros parâmetros e características que podem interferir na qualidade das soluções obtidas. Dentre eles os principais são o tamanho da população, o total de gerações e a forma de geração da população inicial:

1. tamanho da população: enquanto populações com poucos indivíduos podem convergir rapidamente para ótimos locais de baixa qualidade, trabalhar com populações grandes pode exigir a realização de muita computação;

2. critério de parada: os dois principais critérios de parada são o número total de gerações ou a convergência da população. Caso seja definido um número fixo de gerações, pode ser que este número não seja suficiente para se obter à solução ótima. Caso o critério de parada seja a convergência da população, pode-se definir que o AG irá terminar quando todos os indivíduos forem iguais, o que pode nunca ocorrer, ou quando exista um percentual de indivíduos iguais na população;

3. geração da população inicial: a população inicial pode ser gerada de forma aleatória ou pode ser utilizada uma heurística que utilize algum conhecimento do problema e possibilite um direcionamento da busca a uma região do espaço de busca próxima à solução ótima.

Algoritmos Genéticos – Versões Aperfeiçoadas

Embora os AGs sejam muito utilizados na literatura para a solução de problemas de otimização, o desempenho destes na sua forma tradicional pode ser pior do que o desempenho de outras metaheurísticas. Com o intuito de melhorar o desempenho dos AGs, novas versões diferentes do AG tradicional têm sido propostas. Dentre as variações de AGs mais utilizadas, estão as propostas de inclusão de novos operadores genéticos e novos mecanismos para a criação da população inicial, bem como o desenvolvimento de novos processos para a seleção e reprodução de indivíduos. Nessa "*evolução*" dos AGs, Glover **Erro! A origem da referência não foi encontrada.** introduziu a abordagem evolucionária denominada *Scatter Search*, que apresenta características semelhantes e complementares aos AGs, e que tem se mostrado promissora na resolução de problemas de otimização[16]. Na abordagem *Scatter Search*, a idéia principal é trabalhar com uma versão mais determinística de AGs utilizando a idéia da combinação linear de boas soluções (soluções elite) obtidas durante as iterações do algoritmo, com o intuito de obter uma solução intermediária entre elas que seja melhor do que as soluções elite envolvidas. Em outra linha de pesquisa relacionada à melhora no desempenho dos AGs, Moscato [29] propôs os Algoritmos Meméticos (AMs), cuja representação formal foi apresentada mais tarde por Radcliffe e Surry [36]. Nos AMs é introduzido um procedimento de busca local, que corresponde ao processo de alteração nos valores dos elementos de uma solução para investigar uma área do espaço de busca próxima a ela, e que é aplicado a todos os indivíduos da população de cada geração. Outra variação dos AGs, proposta por Lorena e Furtado [25], corresponde aos Algoritmos Genéticos Construtivos (AGC), que agregam novas características à proposta tradicional de AG. Dentre as principais características agregadas está à associação de um *rank* a cada indivíduo, considerando que um indivíduo pode ser constituído de blocos de sub-soluções ou uma solução completa, e a utilização de uma função de aptidão bi-objetiva, que é utilizada para avaliar tanto as

soluções completas quanto os blocos de sub-soluções, os chamados *schemata*. Para reduzir o tempo computacional exigido pela execução de um AG, uma alternativa que vem sendo muito utilizada é a sua paralelização. Tendo em vista que os AGs realizam uma busca paralela no espaço de soluções do problema, o desenvolvimento de uma versão paralela pode ser realizado de forma mais direta do que em outros algoritmos desenvolvidos para problemas de otimização [14, 31, 32, 35, 42].

5. Propostas de métodos eficientes para a solução de Problemas de Clusterização.

O objetivo desta seção, é apresentar novas propostas utilizando conceitos de Algoritmos Evolutivos para algumas aplicações de Problemas de Clusterização citadas na seção anterior. Para isso, inicialmente mostramos como obter algoritmos evolutivos eficientes para o Problema de Particionamento de grafos. Numa segunda etapa, propomos um novo algoritmo evolutivo híbrido para o problema de clusterização de células de manufatura e ao final, apresentamos um novo algoritmo também evolutivo para Problemas de Clusterização Automática Genérica.

5.1 Clusterização em Grafos com Algoritmos Genéticos

Os problemas de particionamento de grafos já são bastante explorados e existem diferentes modelos para representar este problema. Um modelo aplicado à Engenharia de Software foi proposto por Doval, Mancoridis e Mitchell [13] e será utilizado como base para os algoritmos propostos neste trabalho, sendo, dessa forma, descrito com maiores detalhes a seguir. Doval et al. [13] propuseram um algoritmo genético tradicional para obter, de forma automática, um bom particionamento (ou clusterização) de um grafo de dependências de módulos (MDG - *Module Dependency Graph*). Um MDG é uma maneira utilizada pelos projetistas de *software* para tornarem sistemas complexos mais compreensíveis e corresponde a um grafo orientado onde os módulos de um sistema são representados pelos vértices e as dependências estáticas entre os módulos são representadas pelos arcos do grafo. No algoritmo genético proposto em [13], e referenciado a partir daqui como AGT (Algoritmo Genético Tradicional), é realizada uma clusterização automática, não sendo necessário especificar previamente o número de *clusters* em que o MDG deve ser particionado. Durante a execução do AGT, cada indivíduo da população corresponde a uma solução válida, podendo cada solução possuir um número diferente de *clusters* das demais. Para avaliar a qualidade das clusterizações de um grafo orientado não ponderado, é apresentada uma função-objetivo que leva em consideração esta característica da clusterização automática.

Representação dos Indivíduos

Seja um grafo orientado $G = (V, E)$, onde V corresponde a um conjunto de n vértices $\{v_1, v_2, ..., v_n\}$, e E representa o conjunto de arcos (v_i, v_j) tal que $v_i, v_j \in V$, onde cada arco incide de v_i e é incidente em v_j. Dado um conjunto finito de *clusters* C, uma clusterização de G pode ser definida pela função Q: $V \rightarrow C$ que mapeia cada vértice de V em um *cluster* de C, sendo $P(C_i)$ o conjunto de vértices associado ao *cluster* C_i, isto é, $P(C_i) = \{v \in V : Q(v) = C_i\}$.

No AGT cada indivíduo da população deve representar um esquema de clusterização *Q*. Para isto, cada indivíduo corresponderá a um vetor de inteiros $\{e_1, e_2, ..., e_n\}$ onde cada elemento e_i de índice i do vetor indica o identificador do *cluster* do vértice v_i, ou seja, $Q(v_i) = e_i$, onde $1 \leq i \leq n$.

A Figura 5.1.1 mostra um exemplo de indivíduo utilizando a codificação *group-number* para representar uma clusterização de um grafo orientado com 8 vértices.

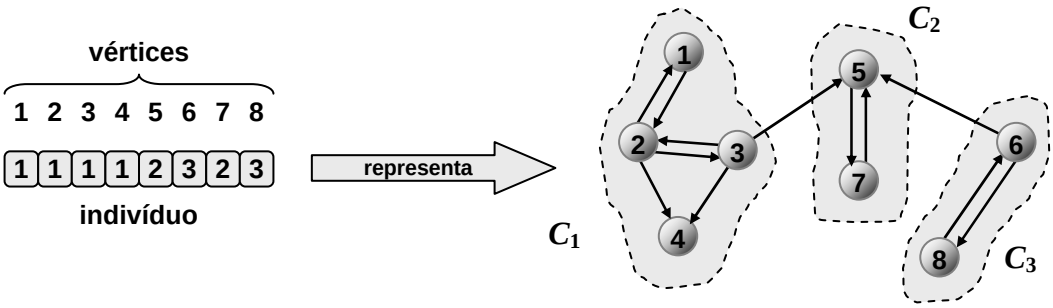

Figura 5.1.1 – Exemplo de codificação *group-number* para clusterização de um grafo orientado

População Inicial

A população inicial do AGT é gerada de forma aleatória tal que, para cada indivíduo *j* da população, a cada vértice *v* é associado um *cluster* $C_i \in C$ tal que $| C | \leq n$, sendo *n* o número de vértices do grafo. Doval et al. [13] propõem que o número de indivíduos da população inicial (tamanho da população) seja igual a $10 \times n$. Contudo, observamos que, utilizando esta proposta, o tamanho da população pode se tornar muito grande para grafos com número elevado de vértices (por exemplo da ordem de centenas de vértices). O tamanho da população é mantido fixo em cada geração ao longo do processo evolutivo do AGT.

Seleção e Reprodução

Partindo-se da população inicial, o processo evolutivo é iniciado de forma que diferentes populações de indivíduos façam parte das diferentes gerações. Cada geração corresponde a uma iteração do AGT, em que ocorrem os passos de seleção e reprodução dos indivíduos de uma população para gerar uma nova população sobre a qual serão aplicados os operadores genéticos. Durante a seleção e reprodução de uma população, *p* indivíduos são escolhidos da população atual levando-se em consideração os valores das suas respectivas funções de aptidão. A seleção utilizada no AGT corresponde à estratégia da roleta, complementada com elitismo, sendo este último responsável por garantir que o indivíduo mais apto da população corrente seja selecionado para a próxima população. Os *p* indivíduos selecionados de uma geração formam a nova população, que será utilizada na próxima geração do AGT.

Função de Aptidão

Para medir a qualidade da configuração de uma clusterização (solução) para um grafo orientado, Doval et al. [13] apresentam uma função que leva em consideração as conexões entre os vértices do grafo. A função, denominada Qualidade de Modularização (*MQ – Modularization Quality*), é utilizada como função de aptidão do

algoritmo genético. O objetivo do AGT será encontrar um bom (possivelmente ótimo) particionamento, através da maximização da função *MQ*. Para calcular o valor da função *MQ* de uma clusterização, é necessária a definição de duas outras funções que levam em consideração o total de arcos que conectam vértices internos a um *cluster* e o total de arcos envolvendo vértices de diferentes *clusters*. São elas:

- Intraconectividade (A_i): a intraconectividade A_i de um *cluster i* é:

$$A_i = \frac{\mu_i}{N_i^2} \qquad (5.1.1)$$

onde μ_i é total de arcos internos ao *cluster i* e N_i o total de vértices do *cluster i*.

- Interconectividade ($B_{i,j}$): considerando um par de *clusters i* e *j*, com ε_{ij} sendo total de arcos entre os *clusters*, N_i e N_j os totais de vértices dos *clusters i* e *j*, respectivamente, a medida da interconectividade $B_{i,j}$ entre o par de *clusters* é :

$$B_{i,j} = \begin{cases} 0 & se \quad i = j \\ \dfrac{\varepsilon_{ij}}{2N_i N_j} & se \quad i \neq j \end{cases} \qquad (5.1.2)$$

A função de intraconectividade de um *cluster* considera o total de arcos internos a ele em relação ao número máximo de arcos possível. É considerada nesta função a possibilidade da existência de laços, ou seja, ligações que envolvam apenas um vértice. Os valores da intraconectividade e interconectividade variam entre 0 e 1 e uma clusterização de boa qualidade é aquela que possui um valor grande para o somatório da intraconectividade de cada um dos *clusters* e um valor pequeno para o somatório da interconectividade de todos os pares de *clusters* possíveis no particionamento. A função *MQ*, que considera a intraconectividade total e a interconectividade total de uma clusterização é então definida como sendo:

$$MQ = \begin{cases} \dfrac{\sum_{i=1}^{k} A_i}{k} - \dfrac{\sum_{i,j=1}^{k} B_{i,j}}{\frac{k(k-1)}{2}} & \forall k > 1 \\ A_i & k = 1 \end{cases} \qquad (5.1.3)$$

onde A_i é a intradependência do *cluster i*, $B_{i,j}$ é a interdependência entre *clusters i* e *j* e *k* é o total de *clusters* da solução.

O valor de *MQ*, equação (5.1.3), corresponde à diferença entre a média de intraconectividade e a média de interconectividade, podendo variar entre –1 e 1. Tendo em vista que a função *MQ* premia a obtenção de *clusters* coesos (com alto valor de intraconectividade) e penaliza particionamentos com muitas dependências entre seus *clusters* (alto valor de interconectividade), quanto maior for o valor de *MQ*, melhor a clusterização. A função *MQ* é então utilizada como função de aptidão do AG, tendo como objetivo a sua maximização. Na Figura 5.1.2 são apresentadas duas clusterizações diferentes para um mesmo grafo e os respectivos valores para a função *MQ*. O valor de *MQ* para a clusterização A é superior ao valor referente a clusterização B e, dessa forma, esta última pode ser considerada uma clusterização pior do que aquela (A), que

por sua vez pode ainda não ser a melhor clusterização possível (solução ótima) para o grafo do exemplo. Para termos certeza que uma clusterização é a solução ótima devemos verificar, para todas as possíveis combinações dos vértices em *clusters*, qual delas possui o maior valor para a função *MQ*.

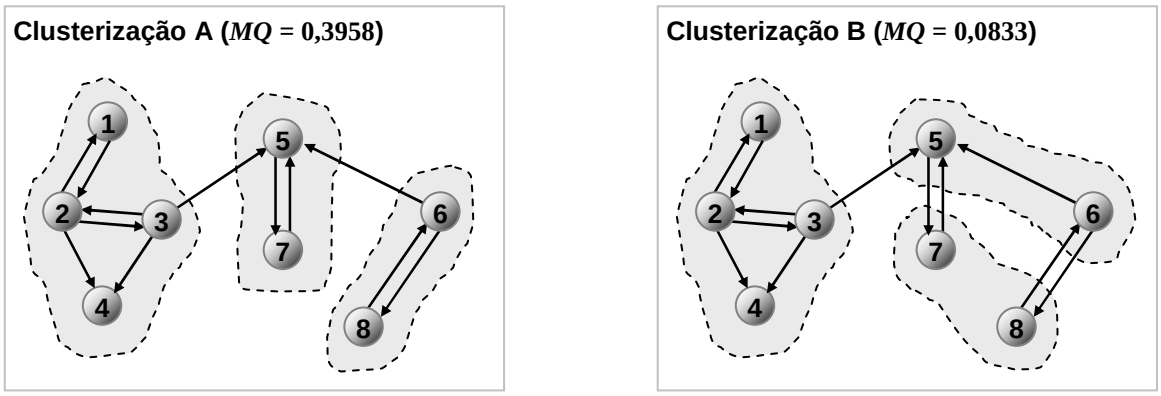

Figura 5.1.2 – Exemplos de valores para a função *MQ* associada a diferentes clusterizações

Operador de Cruzamento

O operador de cruzamento é aplicado imediatamente após a seleção dos indivíduos da atual população e reprodução na nova população, sendo usado para combinar pares de indivíduos (indivíduos pais) com o objetivo de obter novos indivíduos (indivíduos filhos). O operador de cruzamento utilizado é o cruzamento de um ponto, em que, um trecho de um indivíduo do par selecionado é trocado com mesmo trecho do outro indivíduo do par. O trecho inicia em uma posição i ($1 \leq i \leq n$) do indivíduo, obtida de forma aleatória, onde n é o número de vértices do grafo (e tamanho do indivíduo). Os dois indivíduos do par são então alterados pela troca dos valores dos elementos entre as posições $i + 1$ e n (inclusive). A Figura 5.1.3 mostra um exemplo de aplicação do operador de cruzamento, onde foi selecionada a quarta posição para ser o ponto de cruzamento.

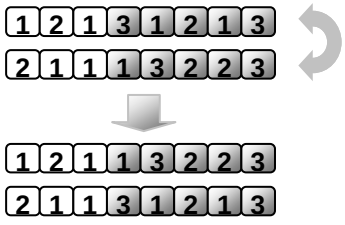

Figura 5.1.3 – Exemplo de aplicação do operador de cruzamento no AGT

Operador Mutação

O operador mutação é aplicado a cada indivíduo da população, após a aplicação do operador de cruzamento. No operador mutação definido por Doval et al. [13], o valor de cada i-ésimo elemento do indivíduo, correspondente ao identificador do *cluster* do vértice v_i com $1 \leq i \leq n$, possui uma probabilidade de ser trocado por um valor aleatório q,tal que $1 \leq q \leq n$. Assim, o número máximo de *clusters* será o número de elementos (vértices) do problema (grafo) associado.

A taxa de aplicação do operador mutação é definida no AGT como sendo igual a 0,004 $\times \log_2(n)$. A Figura 5.1.4 mostra um exemplo de aplicação do operador mutação sobre um indivíduo no AGT.

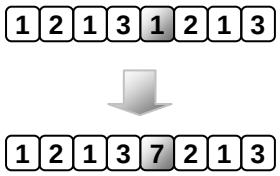

Figura 5.1.4– Exemplo de aplicação do operador mutação no AGT

Propostas para melhorar o desempenho do AGT

Com o intuito de tentar melhorar o desempenho do algoritmo AGT, a partir de uma bateria de testes preliminares, procuramos verificar os prós e contras desta proposta da literatura. Baseado nestas indicações, passamos a propor as seguintes mudanças:

1. variação na taxa de aplicação do operador mutação no AGT.
2. uso de seleção de reprodutores via regra de torneio.
3. outros operadores crossover.
4. módulo de diversificação numa população. Neste caso, de tempos em tempos, quando notamos que a população se tornou muito homogênea, criamos uma nova população replicando o melhor individuo gerado ate o momento pelo AE. Para cada réplica, abrimos uma janela de tamanho e posição aleatória, e somente dentro desta janela, a composição do individuo será refeita aleatoriamente.
5. modulo de busca local no AGT.

Uma segunda bateria de testes foi efetuada avaliando cada proposta isoladamente, destas a que apresentou maior impacto (melhora) foi a de busca local. Nesta abordagem, durante o processo evolutivo do AGT, enquanto o melhor indivíduo obtido a cada iteração não corresponder à solução ótima para a clusterização, pode-se tentar melhorá-lo através da aplicação de um procedimento de busca local. Basicamente, o procedimento de busca local consiste em uma investigação da vizinhança de uma solução, com o objetivo de obter um ótimo local, e normalmente é realizada através da substituição de valores dos genes de um indivíduo por valores que podem ser definidos aleatoriamente ou obtidos conforme alguma característica do problema e da instância.

No AGT com busca local proposto neste trabalho, e referenciado com a indicação "+B", durante o procedimento de busca local parte-se de uma clusterização $C_o \in C$, obtida pelo AGT, e são analisadas várias clusterizações $C_1, C_2, C_3, ..., C_n \in C$ da vizinhança de C_o. A clusterização C_o corresponde à melhor solução obtida em cada geração e cada uma das outras n clusterizações é obtida através de uma única modificação no identificador do *cluster* de algum vértice da clusterização obtida na iteração anterior. Os passos principais da busca local proposta neste trabalho são apresentados no algoritmo da Figura 5.1.5.

Algoritmo BuscaLocal (entrada: clusterização C_0)

1. $MQ_C \leftarrow$ função de aptidão da clusterização C_0;

2. **Para** $i \leftarrow 1$ **até** n **faça** // para todos os n vértices do grafo

3. Para cada *cluster* $q \in Q$ computar $nA_i(q)$ que correspondente à soma do total de arcos incidentes do vértice v_i nos vértices do *cluster* q e o total de arcos incidentes dos vértices de q no vértice v_i;

4. $qMax_i \leftarrow q$ *tal que* $nA_i(q)$ seja o maior valor obtido em 3;

5. **Se** $qMax_i \neq cluster(v_i)$ **então**

6. $NovoMQ_C \leftarrow$ valor da aptidão da clusterização C_i considerando o vértice v_i contido no *cluster* $qMax_i$;

7. **Fim se;**

8. **Se** $NovoMQ_C > MQ_C$ **então**

9. Substituir o *cluster*(v_i) na clusterização C_{i-1} por $qMax_i$;

10. $MQ_C \leftarrow$ função de aptidão da nova clusterização C_i;

11. **Senão**

12. Manter v_i no *cluster* original da clusterização C_{i-1};

13. **Fim se;**

14. **Fim para**;

15. Retornar a clusterização C_n.

Figura 5.1.5– Algoritmo de busca local aplicado a uma clusterização C_0

Noutra bateria de testes, a idéia, foi a de avaliar o impacto do uso de várias das propostas em conjunto. Para isso, foram selecionadas alguns módulos a serem consideradas:

- taxa de aplicação do operador mutação igual a 0,0040;
- calibração no operador mutação;
- seleção por torneio com 2 indivíduos participantes, ou;
- seleção por torneio com 4 indivíduos participantes;
- operador de cruzamento utilizando proposta GGA;
- diversificação após 10 gerações sem evolução média da população, ou;
- diversificação após 25 gerações sem evolução média da população;
- inserção do procedimento de busca local.

O total de combinações das propostas acima, gerou 135 versões do AGT incluindo a versão original proposta em [13], e dentre eles as 9 de melhor desempenho são listadas a seguir:

Para identificação das características de cada versão híbrida indicada na Tabela 5.1.1, devem ser considerados os seguintes indicadores no nome da versão:

1 = utilização da taxa de aplicação do operador mutação igual a 0,0040;

2 = utilização da calibração no número de clusters considerados no operador mutação;

3 = utilização do tipo de seleção por torneio, em substituição ao método da roleta, com dois indivíduos por torneio;

4 = utilização do tipo de seleção por torneio, em substituição ao método da roleta, com quatro indivíduos por torneio;

5 = substituição do operador de cruzamento de um ponto, pelo operador de cruzamento proposto no GGA de Falkenauer ***Erro! A origem da referência não foi encontrada.****;*

6 = ativação do procedimento de diversificação da população após 10 gerações sem uma melhora no valor do MQ médio dos indivíduos da população;

7 = ativação do procedimento de diversificação da população após 25 gerações sem uma melhora no valor do MQ médio dos indivíduos da população;

8 = aplicação do procedimento de busca local ao melhor indivíduo de cada geração.

Desta forma, a versão AEH123568, por exemplo, corresponde ao AGT com a utilização das opções 1, 2, 3, 5, 6 e 8. Considerando todos estes experimentos apresentados, as nove versões híbridas, que obtiveram os melhores resultados médios (e resultados similares) são indicadas na Tabela 5.1.1 (não sendo considerada uma ordem específica das versões).

Nome da Versão
AEH123568
AEH12378
AEH124568
AEH12368
AEH12478
AEH13578
AEH1368
AEH14568
AEH1468

Tabela 5.1.1 – Versões híbridas mais eficientes

Na Tabela 5.1.2 são apresentados os valores médios para a função *MQ* dos melhores indivíduos obtidos após a conclusão das 10 execuções de cada uma das melhores versões híbridas (indicadas na Tabela 5.1.1), para cada um dos grafos considerados.

Grafo	Sol. *Best*	AEH 123568	AEH 12378	AEH 124568	AEH 12468	AEH 12478	AEH 13578	AEH 1368	AEH 14568	AEH 1468
A10	0,7083	**0,7083**	**0,7083**	**0,7083**	**0,7083**	**0,7083**	**0,7083**	**0,7083**	**0,7083**	**0,7083**
A20	0,7371	**0,7371**	**0,7371**	**0,7371**	**0,7371**	**0,7371**	**0,7371**	**0,7371**	**0,7371**	**0,7371**
A40	0,7087	0,6806	0,6806	0,6806	0,6806	0,6806	0,6818	0,6806	0,6806	0,6806
A60	0,7402	**0,7402**	**0,7402**	**0,7402**	**0,7402**	**0,7402**	**0,7402**	**0,7402**	**0,7402**	**0,7402**
A80	0,7780	**0,7780**	**0,7780**	**0,7780**	**0,7780**	**0,7780**	**0,7780**	**0,7780**	**0,7780**	**0,7780**
A100	0,8092	**0,8092**	**0,8092**	**0,8092**	**0,8092**	**0,8092**	**0,8092**	**0,8092**	**0,8092**	**0,8092**
E10	0,7083	**0,7083**	**0,7083**	**0,7083**	**0,7083**	**0,7083**	**0,7083**	**0,7083**	**0,7083**	**0,7083**
E20	0,7089	**0,7089**	**0,7089**	**0,7089**	**0,7089**	**0,7089**	**0,7089**	**0,7089**	**0,7089**	**0,7089**
E40	0,6485	**0,6485**	**0,6485**	**0,6485**	**0,6485**	**0,6485**	**0,6485**	**0,6485**	**0,6485**	**0,6485**
E60	0,7402	**0,7402**	**0,7402**	**0,7402**	**0,7402**	**0,7402**	**0,7402**	**0,7402**	**0,7402**	**0,7402**
E80	0,7075	**0,7075**	**0,7075**	**0,7075**	**0,7075**	**0,7075**	**0,7075**	**0,7075**	**0,7075**	**0,7075**
E100	0,6980	**0,6980**	**0,6980**	**0,6980**	**0,6980**	**0,6980**	**0,6980**	**0,6980**	**0,6980**	**0,6980**

Tabela 5.1.2 – Valores médios para a função *MQ* dos melhores indivíduos obtidos após a conclusão das 10 execuções de cada uma das melhores versões híbridas

Para cada uma as nove melhores versões híbridas (AEH), todas as execuções utilizando os grafos considerados com até 100 vértices obtiveram a solução *best*, exceto as que utilizaram o grafo A40, conforme mostrado na Tabela 5.1.2. Isto significa que cada linha da Tabela 5.1.2 contém os mesmos valores, ou seja, os valores médios são todos iguais ao *best* para todas as versões híbridas, excetuando-se aqueles referentes ao grafo A40.

O valor *best* é dado ao melhor valor considerando todas as soluções geradas por todos os algoritmos aqui implementados para uma dada instância. Um aspecto importante a ser observado com relação às nove melhores versões híbridas é que todas elas possuem no mínimo quatro propostas combinadas: alteração na taxa de aplicação do operador mutação, seleção por torneio, inclusão do procedimento de diversificação e inclusão do procedimento de busca local.

Para verificar se são significativas as diferenças entre os tempos médios de execução de todas as versões utilizando cada grafo, as duas últimas colunas da Tabela 5.1.3 apresentam os valores referentes ao tempo médio de execução para cada grafo e o desvio padrão considerando todas as versões. Com bases nos valores indicados na Tabela 5.1.3 pode-se concluir que não existe uma relação direta entre o número de módulos utilizados em uma versão e o seu tempo médio de execução, tendo em vista que, mesmo comparando duas versões, estas podem apresentar comportamentos diferentes (em relação ao tempo de execução) para grafos diferentes. Pode-se concluir também que não existe uma diferença significativa entre os tempos médios de execução das diferentes versões híbridas.

Grafo	AEH 123568	AEH 12378	AEH 124568	AEH 12468	AEH 12478	AEH 13578	AEH 1368	AEH 14568	AEH 1468	Média	Desvio Padrão
A10	0,094	0,089	0,091	0,093	0,091	0,086	0,086	0,090	0,090	0,090	0,002
A20	0,464	0,448	0,464	0,463	0,446	0,430	0,440	0,439	0,443	0,449	0,011
A40	2,639	2,620	2,654	2,649	2,625	2,543	2,564	2,574	2,569	2,604	0,040
A60	8,925	8,504	8,916	8,908	8,472	8,336	8,763	8,762	8,784	8,708	0,202
A80	17,255	17,225	17,259	17,187	17,246	16,985	17,020	17,000	16,980	17,129	0,119
A100	31,222	31,283	31,345	31,154	31,416	31,101	30,679	30,902	30,946	31,116	0,234
E10	0,092	0,092	0,090	0,090	0,089	0,083	0,089	0,089	0,089	0,089	0,002
E20	0,466	0,448	0,464	0,464	0,446	0,423	0,445	0,445	0,447	0,450	0,012
E40	2,643	2,552	2,645	2,638	2,554	2,469	2,558	2,573	2,572	2,578	0,052
E60	9,065	8,626	9,073	9,055	8,596	8,456	8,893	8,866	8,908	8,838	0,211
E80	17,254	16,704	17,249	17,244	16,687	16,406	16,984	16,968	16,972	16,941	0,271
E100	32,221	31,360	32,189	32,166	31,406	30,932	31,804	31,765	31,797	31,738	0,399

Tabela 5.1.3 – Tempo médio para a execução completa de cada versão híbrida (em segundos)

Para verificar se as melhores versões híbridas para os grafos de 10 a 100 vértices mantêm seus desempenhos para grafos maiores, foram realizados também experimentos para os grafos com 160, 200, 500 e 1000 vértices. Para cada grafo foram realizados 2 experimentos para cada uma das nove melhores versões híbridas. Com o objetivo de reduzir o tempo de execução das versões híbridas para os grafos com 500 e 1000 vértices, foram considerados o tamanho da população fixo e igual a 200 indivíduos e um total de 4000 gerações. Os valores médios para a função *MQ*, obtidos por cada versão aplicada a cada grafo, são mostradas na Tabela 5.1.4 São indicados em negrito os resultados que igualaram ou superaram a solução *best*.

Grafo	Sol. *Best*	AEH 123568	AEH 12378	AEH 124568	AEH 12468	AEH 12478	AEH 13578	AEH 1368	AEH 14568	AEH 1468
A160	0,9090	**0,9090**	**0,9090**	**0,9090**	**0,9090**	**0,9090**	**0,9090**	**0,9090**	**0,9090**	**0,9090**
A200	0,8783	**0,8882**	**0,8882**	**0,8885**	**0,8883**	**0,8878**	**0,8888**	**0,8885**	**0,8888**	**0,8892**
A500	0,8785	**0,8853**	**0,8852**	**0,8837**	**0,8841**	**0,8850**	0,8711	0,8673	0,8668	0,8682
A1000	0,9415	**0,9507**	**0,9499**	**0,9513**	**0,9467**	**0,9512**	0,9257	0,9300	0,9303	0,9297
E160	0,7364	**0,7364**	**0,7364**	**0,7364**	**0,7364**	**0,7364**	**0,7364**	**0,7364**	**0,7364**	**0,7364**
E200	0,7662	**0,7662**	**0,7662**	**0,7662**	**0,7662**	**0,7662**	**0,7662**	**0,7662**	**0,7662**	**0,7662**
E500	0,8759	**0,8759**	**0,8759**	**0,8759**	**0,8759**	**0,8759**	**0,8759**	**0,8759**	**0,8759**	**0,8759**
E1000	0,8865	**0,8865**	**0,8865**	**0,8865**	**0,8865**	**0,8865**	**0,8865**	**0,8865**	**0,8865**	**0,8865**

Tabela 5.1.4 – Valores médios de *MQ* para as soluções obtidas com as versões híbridas do AGT para grafos com 160 e 200 vértices

Nos experimentos utilizando os grafos com dimensões elevadas, as versões híbridas AEH123568, AEH12378, AEH124568, AEH12469 e AEH12478 *conseguiram alcançar a solução best em todas as suas execuções. Vale observar que, para os grafo A200, A500 e A1000, estas versões híbridas conseguiram obter soluções melhores do que a solução best* (recordando, o valor *best* é o melhor valor entre o valor de *MQ* das soluções obtidas considerando todas as versões).

Conclusão

Os resultados apresentados neste capítulo mostraram que a combinação das propostas apresentadas, através de versões híbridas do AGT, permite a obtenção de soluções ainda melhores do que se forem utilizadas separadamente. Além do excelente desempenho obtido para os grafos de 10 a 100 vértices, algumas versões híbridas conseguiram manter este desempenho para grafos ainda maiores, com 160, 200, 500 e 1000 vértices. Por fim, apesar de não termos mostrado explicitamente, vale lembrar que a versão da literatura [13], não ficou sequer entre as 100 melhores versões do total de 135 versões avaliadas preliminarmente.

5.2. Clusterização em Sistemas de Manufatura

O problema de formação de células de manufatura (PFCM) na sua versão original é representado como uma matriz "*parte x máquina"* onde as linhas representam as partes e as colunas às máquinas, ou vice-versa. Considerando aqui uma matriz A = (*parte x máquina)*, cada célula a_{ij} da matriz é igual a 1 se à *parte i* utiliza a máquina *j*, e 0 caso contrário; a formação dos grupos (clusters) "célula / família" é feita através da permutação das linhas e colunas desta matriz. Por exemplo na tabela 2.1, considere um fluxo de produção composto por 6 *partes*, 4 máquinas e 14 atividades (posições da matriz com valor 1). As tabelas 5.2.1 e 5.2.2 representam respectivamente a matriz de entrada do problema e uma possível matriz solução com formação de duas células/*famílias*.

	M1	M2	M3	M4
P1		1	1	
P2	1	1		1
P3		1	1	
P4	1			1
P5	1	1		
P6		1	1	1

Tabela 5.2.1: Matriz de entrada do um PFCM

	M2	M3	M1	M4
P1	1	1		
P3	1	1		
P6	1	1		1
P2	1		1	1
P4			1	1
P5	1		1	

Tabela 5.2.2: exemplo de uma clusterização

Na matriz da tabela 5.2.2 existem duas *famílias*: [P1,P3,P6] e [P2,P4,P5]. Associadas a estas *famílias* temos 2 células: [M2,M3] e [M1,M4].

Este problema (PFCM), já é razoavelmente explorado pela literatura. O objetivo desta seção é citar alguns trabalhos existentes que acreditamos serem mais significativos para o enfoque deste trabalho. Um AG para o PFCM é apresentado por Joines et al. (1996) [19]. O algoritmo utiliza 4 tipos de mutação e 3 tipos de crossover, obtendo bons resultados segundos seus autores. Em Wang(1998) [43], é apresentada uma heurística que, numa primeira etapa, determina a criação de *k famílias* (grupos de partes) ou

células (grupos de máquinas), onde *k* é um parâmetro de entrada; através da determinação dos *k* pares de partes ou máquinas mais dissimilares, de acordo com um índice de dissimilaridade calculado entre elas. A partir daí, é feita uma atribuição das partes e máquinas restantes às células e famílias existentes com base num modelo de atribuição linear. Porém, as soluções se mostram fortemente dependentes do parâmetro inicial k e do número máximo de máquinas por células e de partes por família. Uma metaheurística Busca Tabu foi proposta Aljaber et al. (1997) [2], sendo que ela se constitui basicamente de três fases: a primeira constitui-se na determinação do melhor sequenciamento de máquinas e partes, de maneira a se formar dois caminhos mínimos; um caminho para partes e outro para máquinas, onde as distâncias entre máquinas e entre partes estão diretamente relacionadas com o grau de similaridade entre elas. Estes dois caminhos são determinados através da aplicação do algoritmo de busca tabu propriamente dito. Numa segunda fase, é montada a matriz de clusterização referente à melhor solução encontrada na primeira fase. A terceira fase se constitui da determinação do número de clusters através da identificação das fronteiras de cada cluster. Porém necessita da informação de um limite superior de clusters a serem formados que deve ser informado inicialmente pelo usuário. Em Mak et al. (2000) é proposto um AG adaptativo onde as taxas de utilização dos operadores de mutação e crossover mudam de acordo com a eficiência dos mesmos ao longo da execução do algoritmo. Em 2002, Gonçalves e Resende (2002) [18] apresentaram um algoritmo genético híbrido (HGA) que incorpora ao AG básico um procedimento de busca local, tendo este algoritmo igualado ou melhorado os resultados existentes na literatura para 35 instâncias do PFCM. O HGA constrói uma população inicial de indivíduos de forma aleatória; o critério de parada é pelo número de iterações; a função de aptidão é a mesma que a usada em Joines et al. (1996) [19] e dada pela fórmula:

$$\Gamma = \frac{e - e_0}{e + e_v}$$

onde: e = o número de operações (1's) na matriz de entrada; e_V = o número de lacunas nos blocos diagonais (células/famílias), que representam a esparcidade de cada cluster; e_0 = número de elementos excepcionais (elementos preenchidos com 1 que não pertencem às células/famílias). Observa-se que a função de avaliação procura minimizar o número de elementos excepcionais (ou ruídos) minimizando o movimento entre clusters; e lacunas nos clusters (células/famílias), maximizando o uso das máquinas nos clusters. Quanto mais próximo de 1 é o seu valor, melhor será a solução. A busca local proposta é aplicada a todos os indivíduos e é iniciada sob uma solução parcial, ou seja, sob uma solução que ainda não possui um conjunto de famílias de partes. Basicamente ela se constitui de dois passos, que são os seguintes: Inicialmente a associação de partes a células é feita com base no cálculo de um coeficiente, de maneira que cada parte é associada à célula para a qual possui o maior coeficiente. Caso esta nova associação de partes a células aumente a aptidão da solução, então se executa o passo seguinte sob a nova solução, senão a busca local é finalizada e as partes são associadas às suas células originais. Se for a primeira execução do passo 1, isto significa que foi formado o primeiro conjunto de famílias da solução e então o passo 2 deve ser obrigatoriamente executado. Num passo 2, depois de feita a associação das partes, será formado um

conjunto de famílias que estarão associadas às células de forma similar ao passo anterior. Caso esta nova associação de máquinas aos clusters aumente a aptidão da solução então se executa novamente o passo 1 sob o novo conjunto de células formado, senão a busca local é finalizada.

O algoritmo Evolutivo proposto (AE) para o PFCM

Esta sessão descreve resumidamente as etapas do AE aqui proposto [40].

Representação do indivíduo: Como proposto em Joines et al. (1996) [19], cada indivíduo é composto por (m+n) genes, onde m representa o número de máquinas e n o número de partes do problema. Desta forma, um indivíduo é representado como um string da forma $C = (x_1,...x_m \mid y_1,...y_n)$. Cada gene x_i ou y_i assume valores inteiros, sendo feita à alocação de (máquinas a células) e (partes a famílias) simultaneamente. Considere o indivíduo, para um problema de 12 máquinas, 15 partes e 4 células/famílias: C_1 = (2 4 2 3 2 3 4 1 3 4 3 1 | 2 4 2 4 1 1 2 2 1 1 3 3 4 4 3). Cada gene x_i indica para qual célula cada máquina é alocada. Por exemplo, em C_1 o gene 1 (x_1=2) significa que a máquina M_1 é alocada para a célula 2 e cada gene y_i indica para qual *família* cada *parte* é alocada. O gene m+1 = y_1 = 2 significa que à *parte* P_1 é alocada para a *família* 2.

Função de aptidão: A função de avaliação usada neste trabalho é a mesma usada em Joines et al. (1996) [19] e em Gonçalves et al. (2002) [18] e já descrita anteriormente.

Geração da População inicial: A população inicial constitui-se parte de indivíduos gerados aleatoriamente e parte de indivíduos gerados através de um procedimento heurístico. Este procedimento é baseado na heurística apresenta por Wang (1998) (veja detalhes em [40]) para resolver o PFCM. Esta heurística é adaptada incorporando escolhas aleatórias numa lista restrita composta pelos k melhores candidatos para cada gene de um indivíduo. A heurística de construção de indivíduos é dividida em três fases descritas a seguir:

P1) Na 1ª fase são escolhidos os 3 pares de máquinas menos similares (que possuem o menor número de partes atendidas em comum).

P2) Na 2ª fase é escolhido aleatoriamente um par dentre os 3 pares de máquinas determinados em P1) como sendo os dois clusters iniciais da solução. A partir daí, cada novo cluster inicial é determinado como sendo a máquina menos similar aos clusters já formados. Este processo se repete até que sejam gerados os k clusters, onde k é um parâmetro de entrada da heurística. Após o término desta fase, os k clusters iniciais estarão determinados.

P3) Na 3ª fase, para cada máquina são determinados o 1º e o 2º cluster com maior índice de similaridade (nº de partes atendidas em comum entre a máquina e pelo menos uma máquina do cluster) em relação a ela. Então a máquina é atribuída aleatoriamente a um dos dois, desde que o número máximo de máquinas por cluster no cluster escolhido não esteja alcançado. Caso isso aconteça, a máquina é atribuída a um cluster qualquer que não tenha excedido o número máximo de máquinas. O número máximo de máquinas por cluster é calculado como sendo igual a $\lceil (\text{nº de máquinas/k}) + 1 \rceil$ *e o número máximo de partes por cluster igual a* $\lceil (\text{nº de partes/k}) + 1 \rceil$.

O processo de atribuição das partes aos clusters é idêntico ao processo de atribuição das máquinas aos clusters. A única diferença é que a similaridade entre uma parte e um cluster é dada pelo número de máquinas do cluster em questão que servem à referida

parte. No nosso AE, são gerados *5m* indivíduos na primeira iteração. Destes, permanecem os *m* mais aptos, onde *m* representa o tamanho da população.

Estratégia de seleção dos indivíduos reprodutores: Esta fase trata da escolha dos indivíduos que irão servir como pais para o processo de reprodução e também para decidir entre os pais e filhos gerados, quais irão para a próxima geração. Escolher simplesmente os melhores indivíduos pode levar o algoritmo genético a uma parada prematura em ótimos locais ainda distantes de ótimos globais. Em razão disto, neste trabalho foi dada uma probabilidade a cada indivíduo de ser selecionado, de acordo com o seu valor de aptidão; sendo a probabilidade de escolha maior para indivíduos de melhor aptidão e menor para indivíduos de pior aptidão.

Mecanismo de reprodução: Em nosso trabalho o operador de crossover foi substituído por um mecanismo de reprodução da seguinte forma: A partir de p ($p \geq 2$) pais geramos p filhos. Escolhidos os *p* pais para a reprodução (através de um processo de seleção mostrado no item anterior, onde *p* é um parâmetro de entrada da reprodução), cada gene do filho referente a uma máquina receberá aleatoriamente o valor correspondente a uma das *k'* células nas quais a presença desta máquina é mais freqüente, com base nas *p* soluções pai, onde *k'* é um parâmetro de entrada. Após a associação das máquinas, para cada parte é feita uma lista das *k''* máquinas que mais aparecem no mesmo cluster da referida parte, com base nas *p* soluções pai, sendo *k''* também um parâmetro de entrada. Então cada parte é associada à família correspondente à célula que a máquina escolhida está associada.

Procedimento de busca local: Neste trabalho foi incorporado ao algoritmo evolutivo um procedimento de busca local que numa primeira fase procura associar as partes aos clusters de acordo com a associação das máquinas, gerando um novo conjunto de famílias. Em seguida, caso a aptidão da solução seja melhorada, as máquinas são reassociadas de acordo com as famílias formadas anteriormente, obtendo um novo conjunto de células. Estas duas etapas são executadas alternadamente enquanto houver melhoria na aptidão da solução resultante. Mais detalhadamente, os passos são os seguintes:

P1) Para cada parte *i*, é calculado o coeficiente abaixo, com relação a todo cluster *k*.

P_{ik} = $(N_{1intrak}-N_{ruidosk} - N_{0intrak})$ se $N_{1intrak} \neq 0$; senão (se $N_{1intrak} = 0$), faço $P_k = -\infty$; onde:

$N_{1intrak}$ = número de elementos do clusters *k* iguais a 1 se a parte for associada a ele

$N_{ruidosk}$ = número de elementos iguais a 1 fora do cluster *k* se a parte for associada a ele.

$N_{0intrak}$ = número de elementos iguais a 0 dentro do cluster *k* se a parte for associada a ele.

A penalidade no valor de P_{ik} quando $N_{1intrak} = 0$ faz com que uma parte *não seja* associada pela busca local a um cluster o qual não possui máquinas que a servem. A seguir, cada parte é associada ao cluster *k* para o qual possui o maior coeficiente P_{ik}. Caso haja empate, a parte será associada para o cluster *k* cuja razão entre $N_{1intrak}$ e o número de máquinas pertencentes ao cluster *k* for a maior. Caso não haja melhoria na aptidão da solução, as partes são alocadas aos clusters originais e a busca local é terminada. Caso haja melhoria, executa-se o passo P2.

P2) Usa-se procedimento similar ao passo P1, mas agora se faz a mudança das máquinas de cluster calculando-se M_{ik} para cada máquina *i* em relação a cada cluster *k*, de acordo com a alocação das partes feitas pelo passo P1. Caso haja empate entre coeficientes, a máquina então será associada para o cluster *k* cuja razão entre $N_{1intrak}$ e o número de partes pertencentes ao cluster *k* for a maior. A seguir, cada máquina é associada ao cluster para o qual possui o maior coeficiente. Caso não haja melhoria na aptidão da solução, as máquinas são alocadas aos clusters originais e a busca local é terminada. Caso haja melhoria, executa-se o passo P1 novamente. Estes passos P1) e P2) buscam o mesmo objetivo requerido pela busca local proposta em Gonçalves et al. (2002), contudo a fórmula usada para atingir o objetivo em ambos são distintas.

Calibração do número de clusters: Um parâmetro do PFCM que influi diretamente no resultado da solução final é o número de clusters (células) a serem formados. Se o intervalo analisado para o número de clusters for demasiadamente grande, aumenta-se o espaço de busca e conseqüentemente o tempo de execução do algoritmo. Então é importante que tenhamos uma boa estimativa para os limites inferiores e superiores deste intervalo. Como são consideradas na literatura como soluções inválidas aquelas que contém uma máquina apenas ou uma parte apenas e dado que o número de máquinas é sempre menor que o número de partes, um limite superior do número de clusters usado neste trabalho foi $\lceil(\text{nº de máquinas}/2)\rceil$. Este valor não evita que sejam formadas soluções com clusters unitários, mas restringe o espaço de busca, tendo em vista que soluções com um número de clusters maior que este limite dado possuem obrigatoriamente clusters unitários. O valor do limite mínimo de clusters é igual a 2. Antes da geração de cada indivíduo é sorteado um número dentro desta faixa de limites para ser o número de clusters para a solução associada. O algoritmo então só considera como soluções válidas àquelas sem clusters unitários, sem clusters com linhas ou colunas vazias ou clusters inteiramente vazios.

Parâmetros do AE: No AE são usados os seguintes parâmetros: A população de cada geração (exceto a inicial) é sempre igual a 2,5 vezes o número de máquinas da instância em questão. A busca local é aplicada sobre 30% dos indivíduos, sendo estes escolhidos de acordo com o mesmo critério de escolha de indivíduos reprodutores. Para o mecanismo de reprodução, os valores de *p*, *k'* e *k''* foram respectivamente 15% do tamanho da população, 10% do número máximo de clusters e 10% do número de máquinas. Para *p*, caso este valor seja menor que 5, *p* assume o valor 5. Para *k' e k''*, caso os valores sejam menores que 3, *k' e k''* assumem o valor 3. Para instâncias em que o valor máximo de clusters é igual a 2, o valor de *k'* é igual a 2. O critério de parada usado foi um limite do número de gerações, igual a 150 gerações.

Resultados computacionais

Para avaliar o algoritmo proposto (AE), tomamos como parâmetro de comparação o algoritmo genético híbrido (HGA) proposto por Gonçalves et al. (2002) [18], que segundo seus autores detinha os melhores resultados aproximados da literatura até então. Em relação ao HGA, utilizamos as instâncias disponibilizadas e os resultados por ele obtidos. Os testes computacionais com o AE foram realizados em um computador AMD 1.410 Mhz e 250 Mb de memória. Foram utilizadas 35 instâncias da literatura. O

AE proposto foi executado 10 vezes para cada instância. Além disso, os resultados foram comparados com os resultados obtidos em: AG-JCK, proposto por Joines et al. (1996) [19], e implementado por nós; Outros algoritmos e resultados aqui relacionados estão disponíveis em Gonçalves et al. (2002) [18] como o algoritmo Zodiac, proposto por Srinivasan e Narendan (1991); Grafics, proposto Srinivasan and Narendan (1991); algoritmo de clusterização proposto por Srinivasan em 1991 (Ca); algoritmo genético proposto por Cheng et. al em 1998 (GA) (veja detalhes em [18, 39]) A tabela 4.1 ilustra os resultados para cada instância (colunas 2 a 8), a coluna 9 (Imp) mostra a diferença em percentual da solução do AE com a melhor solução da literatura. Para a execução do AG-JCK, o número de gerações variou de acordo com o tamanho das instâncias, como sugerido pelos autores. O número de gerações para as instâncias foi: instâncias 1 a 9, 150; 10 a 15, 500; 16 a 33, 5.000 e 34 e 35, 20.000 gerações.

Inst.	Dim	AG-JCK	Zodiac	Grafics	Ca	GA	HGA	AE	Imp
1	5 x 7	73.68	73.68	73.68	-	-	73.68	73.68	0%
2	5 x 7	62.50	56.52	60.87	-	**69.56**	62.50	62.50*	-10.14%
3	5 x 18	79.59	77.36	-	-	77.36	79.59	79.59	0%
4	6 x 8	76.92	76.92	-	-	76.92	76.92	76.92	0%
5	7 x 11	53.13	39.13	53.12	-	46.88	53.13	53.13	0%
6	7 x 11	70.37	70.37	-	-	70.37	70.37	70.37	0%
7	8 x 12	68.29	68.29	68.29	-	-	68.29	68.29	0%
8	8 x 20	57.26	58.33	58.13	58.72	58.33	58.72	58.72	0%
9	8 x 20	85.25	85.25	85.25	85.25	85.25	85.25	85.25	0%
10	10 x 10	70.59	70.59	70.59	70.59	70.59	70.59	70.59	0%
11	10 x 15	92.00	92.00	92.00	-	92.00	92.00	92.00	0%
12	14 x 23	66.67	64.36	64.36	64.36	-	69.86	69.86	0%
13	14 x 24	69.33	65.55	65.55	-	67.44	69.33	69.33	0%
14	16 x 24	44.92	32.09	45.52	48.7	-	51.96	51.96	0%
15	16 x 30	56.82	67.83	67.83	67.83	-	67.83	67.83	0%
16	16 x 43	54.86	53.76	54.39	54.44	53.89	54.86	54.86	0%
17	18 x 24	51.26	41.84	48.91	44.20	-	54.46	54.46	0%
18	20 x 20	40.91	21.63	38.26	-	37.12	42.96	42.96	0%
19	20 x 23	47.77	38.66	49.36	43.01	46.62	49.65	49.65	0%
20	20 x 35	76.22	75.14	75.14	75.14	75.28	76.14	76.14	-0,10%
21	20 x 35	56.52	51.13	-	-	55.14	58.06	58.06	0%
22	24 x 40	100.00	100.0	100.0	100.0	100.0	100.0	100.0	0%
23	24 x 40	85.11	85.11	85.11	85.11	85.11	85.11	85.11	0%

24	24 x 40	50.44	73.51	73.51	73.51	73.03	73.51	73.51	0%
25	24 x 40	51.50	20.42	43.27	51.81	49.37	51.85	51.97	+0,23%
26	24 x 40	42.39	18.23	44.51	44.72	44.67	46.5	47.06	+1,20%
27	24 x 40	42.94	17.61	41.67	44.17	42.50	44.85	44.87	+0,04%
28	27 x 27	54.23	52.14	41.37	51.00	-	54.27	54.27	0%
29	28 x 16	21.49	33.01	32.86	40.00	-	43.85	44.62	+1,76%
30	30 x 41	53.55	33.46	55.43	55.29	53.80	57.69	57.93	+0,42%
31	30 x 50	55.45	46.06	56.32	58.7	56.61	59.43	59.66	+0,39%
32	30 x 50	42.02	21.11	47.96	46.30	45.93	50.51	50.51	0%
33	30 x 90	15.78	32.73	39.41	40.05	-	41.71	43.66	+4,68%
34	37 x 53	56.83	52.21	52.21	-	-	56.14	57.54	+2,49%
35	40x100	84.03	83.66	83.92	83.92	84.03	84.03	84.03	0%

Tabela 5.2.3: resultados obtidos por algoritmos da literatura e o algoritmo proposto [39. 40].

Pelos resultados da tabela 5.2.3, onde os melhores resultados estão em negrito, podemos notar que o AE conseguiu melhores resultados isoladamente para 8 das 35 instâncias, obteve a mesma melhor solução da literatura em outros 25 e para as instâncias 2 e 20 obteve uma solução com pior aptidão que a da literatura. Porém vale ressaltar que a solução encontrada para a instância 2 pelo algoritmo GA contém clusters unitários, o que não é permitido no nosso AE e no HGA. Para verificar o desempenho do AE sem esta restrição, executamos o AE e obtivemos na versão relaxada à mesma solução do GA. A análise dos tempos de cpu do AE será descrita resumidamente sem tabelas devido à limitação do tamanho deste paper. Os tempos de cpu do nosso AE foi em média de duas a 4 vezes menor que o tempo exigido pelo HGA. Na maior instância (35), o tempo do AE foi de 62,32 segundos e no HGA de 125, 33 segundos. Em relação ao AG-JCK, os tempos para instância menores o AG-JCK foi mais rápido que o AE, mas à medida que as dimensões cresciam o tempo do AG-JCK foi se tornando bem maior que o do AE. Por exemplo, na instância 34, o tempo do AG-JCK foi de 1765,91 segundos e do AE de 19, 49 segundos.

Conclusões

Pelos resultados computacionais, podemos concluir que o algoritmo proposto obteve em relação à qualidade das soluções, resultados médios superiores do que os algoritmos propostos na literatura e em particular na média superou os resultados obtidos pelo HGA proposto em Gonçalves et al. (2002) [18] que até então detinha os melhores resultados aproximados da literatura. Também em relação aos tempos computacionais, o nosso AE exige tempos similares ou até menores que os de HGA. Desta forma, embora de forma empírica, mostramos o potencial do algoritmo evolutivo aqui proposto, que nos testes realizados se mostrou superior tanto nas soluções por ele geradas como nos tempos computacionais exigidos.

5.3 Clusterização Automática: Um novo Algoritmo Evolutivo com Reconexão de Caminhos (AEC-RC)

Apresentamos aqui, uma nova proposta para a solução do PCA usando conceitos de AEs. O AEC-RC, é composto de três módulos: o módulo de construção, o módulo evolutivo e o módulo de busca local através de reconexão de caminhos. Em termos de aplicação de algoritmos de clusterização, o AEC-RC difere da maioria dos AG's, uma vez que o mesmo é capaz de resolver o problema de agrupamento e também resolver o problema de encontrar o número ideal de clusters.

Uma vez que, em geral, as bases de dados das aplicações de PCA apresentam um número consideravelmente elevado de elementos, buscamos desenvolver uma heurística de construção eficaz para tratar de bases de dados de grande porte e com características tais como: regiões densas e esparsas, ruídos, características estas, usualmente presentes na maioria das aplicações reais.

A fase de construção do AEC-RC

A fase de construção do AEC-RC aqui proposta tem como meta, tentar reduzir o espaço de busca das soluções viáveis do PCA buscando com isso, tanto melhorar a qualidade das soluções geradas pelo algoritmo como também reduzir consideravelmente os tempos computacionais exigidos. De fato, sabe-se que o tamanho da *string* que codifica um indivíduo em um AG pode influir no tempo de processamento do mesmo. Assim, dependendo da codificação utilizada pelo AG para o PC, pode ser interessante substituir grupos de pontos cuja similaridade é considerada alta, por um único ponto. Podemos tomar cada objeto x_i de um conjunto de entrada como uma tupla (x_{i1}, x_{i2}, . , x_{ip}), onde cada coordenada x_{ij} está relacionada com um atributo do objeto. Podemos, então, considerar um objeto como um ponto no espaço R^p. Desta forma, a substituição de um grupo de pontos similares por um único ponto que os represente pode ser feita tomando-se o ponto médio calculado segundo as coordenadas de todos os pontos deste grupo. Nossa proposta é apresentarmos uma abordagem hierárquica baseada em grafos para reduzir o tamanho da *string* que codifica o indivíduo no AE e ao mesmo tempo tirar proveito desta representação na ocasião da geração da população do AEC-RC. Considere os pontos de uma base de dados X como vértices de um grafo construído conforme o conceito de grafo de vizinhança relativa (GVR) (Toussaint (1980) [41]). O GVR do conjunto *X*, denotado *GVR*(*X*), é o grafo associado à matriz de adjacência M dada por:

$$M[i,j] = \begin{cases} 1 \quad se \;\; d(i,j) \leq \max\,[d(x_i, x_k), d(x_j, x_k)], \;\; \forall \;\; x_k \in X, \; k \neq i, j \\ 0, \;\; caso \quad contrário \end{cases} \qquad (5.3.1)$$

onde $d(i,j)$ denota a distância euclidiana entre os pontos x_i e x_j. A equação (5.3.1) diz que dois pontos x_i e x_j são ditos vizinhos relativos se não existir nenhum outro ponto x_k cuja distância em relação x_i e x_j seja inferior a $d(i,j)$. O *GVR*(X) é o grafo $G(V,E)$ onde o conjunto de vértices *V* é formado por todos os pontos do conjunto *X* e o conjunto de arestas *E* é formado por todos os arcos e_{ij} cujo valor de M[i,j] é igual a 1. Como

podemos observar, os componentes conexos de GV R(X) são agrupamentos que refletem uma visão de vizinhança mínima, ou seja, considerando *m* como sendo o número de componentes conexos obtidos de *GVR*(*X*), podemos esperar, como número máximo de clusters o valor *m (m <n)* e como número mínimo 2.

Para concluir a fase de construção do AEC-RC, tomamos os componentes conexos de *GVR*(*X*) e obtemos o conjunto $G = \{G_1, G_2, \ldots, G_m\}$, onde cada G_i é um componente conexo candidato a se tornar um cluster raiz, e V_i, para $i = 1, \ldots, m$ é o centróide do componente conexo G_i. Devemos, então, efetuar agrupamentos entre G_i's de forma a termos a clusterização final. Isto é feito no sentido de otimizar uma função objetivo, tarefa realizada pelo módulo evolutivo do AGS. Antes de explicarmos como se constrói a população inicial do AGS, é importante entendermos a modelagem utilizada para a codificação do indivíduo. Para tanto, considere uma *string* binária $r = (r_1, r_2, \ldots, r_m)$, representando uma solução (semente). Se $r_i = 1$, o cluster $G_i \subset G$ fará parte da solução como cluster pai (cluster raiz). Caso contrário ($r_i = 0$), G_i é considerado um cluster filho e será posteriormente associado a um dos clusters pais indicados pelos r_i's que constam na string como sendo iguais a 1. Como exemplo, da string r = [0 1 0 1 0 0 0 1] de uma entrada que gerou um conjunto $G = \{G_1, G_2, G_3, G_4, G_5, G_6, G_7, G_8\}$, onde cada G_i é um componente conexo de *GVR*(X), podemos dizer que a solução apresentará três clusters, inicialmente G_2, G_4 e G_8 (clusters pais). Os demais clusters (G_1, G_3, G_5, G_6 e G_7) são os clusters filhos, cujos pontos serão anexados a um dos clusters pais para formar com este um novo cluster conforme o critério de similaridade adotado. Na geração do *string* que codifica uma solução, procuramos priorizar os componentes conexos com maior cardinalidade. Assim, a geração da população inicial do AEC-RC se dá de tal forma que a probabilidade de ocorrer o valor 1 em cada elemento r_i da *string r* é proporcional à cardinalidade do componente conexo $G_i \subset$ G. Apresentamos agora o processo de formação de um cluster, ou seja, a transformação de uma semente em solução propriamente dita. Após a geração da semente, considere $Y = \{G_1, G_2, \ldots, G_t\}$ o conjunto formado pelos componentes conexos G_i's associados aos elementos cujo valor é 1 no *string* indivíduo. O conjunto solução $C = \{C_1, \ldots, C_t\}$ é inicialmente formado pelos *t* clusters iniciais G_i's $\subset$ Y e os centróides iniciais H_i's dos clusters da solução são, inicialmente, os centróides V_i's destes clusters iniciais, onde $|C_i| = |G_i|$ para $i = 1, 2, \ldots$, *t* e $|G_i|$ denota o número de pontos pertencentes a G_i. No processo de clusterização, os componentes G_i's em C\Y = $\{G_1, G_2, \ldots, G_m\}$ - Y são analisados um de cada vez de tal forma que o cluster $C_j \subset C$ irá anexar os pontos do componente conexo G_i se $\|V_i - H_j\| \leq \|V_i - H_q\|$ para $1 \leq q \leq t$ e $q \neq j$. Quando algum componente conexo G_i é associado a um cluster C_j, a nova cardinalidade de C_j , denotado por $|C_j|$, e o novo centróide de C_j , denotado por H'_j são atualizados pelas equações (5.3.2) e (5.3.3).

$$H'_j = \frac{H_j * |C_j| + V_i * |G_i|}{|C_j| + |G_i|} \qquad (5.3.2)$$

$$|C'_j| = |C_j| + |G_i| \qquad (5.3.3)$$

Após todos os componentes conexos cujos índices na string apresentam valor zero tiverem sido associados aos clusters pais indicados pelos índices cujo valor é igual a 1, a clusterização está concluída e podemos então avaliar a qualidade da mesma.

O Módulo Evolutivo do AEC-RC

O módulo evolutivo do AEC-RC compõe-se de um AG cuja população inicial é gerada segundo as densidades dos componentes conexos obtidos na fase de construção. Como podemos ver, o número de uns (1s) no indivíduo fornecerá o número de clusters do mesmo. Assim, podemos ver facilmente que dentro do intervalo [1,*m*], com *m* dado pelo número de componentes conexos, o número ideal de clusters a ser encontrado pelo algoritmo é bem menor quando comparado ao número de combinações possíveis dentro do intervalo original [1, *n*] dado pela cardinalidade da base de dados *X*. Para avaliar um indivíduo, utilizamos o critério "*Average Silhouette Width*" apresentado em Kaufman (1989) [22]. Antes de explicarmos os mecanismos dos operadores do AEC-RC, definimos a função de avaliação que resulta no *fitness* de cada indivíduo. Para tanto, seja *i* um ponto pertencente ao cluster C_w construído na clusterização, onde $|C_w| = M > 1$. Considere a dissimilaridade média de *i* em relação a todos os pontos $j \in C_w$ dada pela equação (5.3.4), onde d_{ij} é a distância euclidiana descrita pela equação entre os pontos *i* e *j*. Nos casos em que Cw possuir um único elemento, definimos a(i) = 0.

$$a(i) = \frac{1}{M - 1} \sum_{j=1}^{M} d_{ij} \quad \forall \quad j \neq i \qquad (5.3.4)$$

Considere, ainda, cada um dos clusters $C_k \subset C$ com $k \neq w$. A dissimilaridade média do ponto *i* em relação a todos os pontos de C_k é mostrada na equação (5.2.5), onde *M* é o número de pontos do cluster C_k.

$$d(i, C_k) = \frac{1}{|M|} \sum_{i=1}^{M} d_{ij} \quad \forall \quad j \in C_k \qquad (5.3.5)$$

Denota-se *b(i)* o menor valor dentre todos os $d(i,C_k)$, o que é obtido pela equação (5.2.6). Note que *b(i)* é obtido pelo cluster que seja o vizinho mais próximo do ponto *i*

$$b(i) = \min_{C_k \neq c_w} d(i, C) \qquad (5.3.6)$$

Definimos o valor da silhueta do ponto *i* pela equação , dada por:

$$s(i) = \frac{b(i) - a(i)}{\max\ (a(i),\ b(i))} \qquad (5.3.7)$$

Para avaliarmos a qualidade da solução, calculamos a média aritmética de todos os *s*(*i*)'s obtidos. Desta forma, a função de aptidão do indivíduo é dada por:

$$f = \frac{1}{N} \sum_{i=1}^{N} s(i) \qquad (5.3.8)$$

sujeito a $i \in S$

Para efetuar a seleção dos indivíduos reprodutores, o operador utilizado é o *Método da Roleta*. Entretanto, como a equação que define a aptidão do indivíduo admite valores dentro do intervalo [-1, 1], portanto podendo assumir valores negativos, aumentamos uma unidade no valor de *f* para cada indivíduo. Ao final do processamento, subtraímos uma unidade do valor de *f* do indivíduo campeão. O operador de *crossover* utilizado no AEC é o *crossover* de dois pontos. Foram desenvolvidos dois operadores de mutação clássicos para o AEC cujo emprego é alternado a cada cruzamento.

O Módulo de Reconexão de caminhos (RC)

A busca local aplicada em AEs tem encontrado bastante aceitação por parte dos pesquisadores desta área (veja [14, 31, 32, 33, 34, 35]), fazendo crescer o interesse no desenvolvimento de trabalhos na área de Algoritmos Evolutivos. Neste contexto, a Reconexãp por Caminhos (RC) (*Path-Relinking*) apresenta-se como uma técnica de busca alternativa eficiente em relação às buscas tradicionais [16]. O RC consiste em gerar todas as soluções intermediárias entre uma solução base (SB) e uma solução alvo (AS). Normalmente ambas as soluções são escolhidas dentre as melhores geradas até o momento pela heurística. Pode-se pesquisar no sentido SB para AS ou vice versa, ou em ambas as direções se isso for conveniente. O princípio básico do RC é o de que entre duas soluções de qualidade pode existir uma terceira melhor que os extremos [16].

RESULTADOS COMPUTACIONAIS E ANÁLISE QUALITATIVA

Inicialmente para avaliar o AEC-RC, implementamos este e também um AG da literatura que segundo os autores apresentavam os melhores resultados para o PCA até então. Para isso, antes de comentarmos detalhes das nossas simulações, vamos resumidamente descrever o AG da literatura CLUSTERING. O algoritmo CLUSTERING de Lin and Shiueng (2001) (veja detalhes em [38]) consiste de um Algoritmo Genético (AG) que, a exemplo do AEC-RC, também se propõe resolver o PCA. Este algoritmo faz uso de uma heurística baseada em busca binária para a obtenção da melhor solução através de várias chamadas do próprio AG com variações de um peso *w* que, presente na função de aptidão, prioriza uma clusterização com muitos ou poucos clusters conforme *w* cresça ou diminua. Antes de iniciar a execução do CLUSTERING, vamos explicar a etapa de construção que utiliza também conceitos de componentes conexos como no AEC-RC, mas utilizando fórmulas e critérios diferentes da nossa proposta. Salienta-se que nossa proposta surgiu da necessidade de melhorar o desempenho desta construção de Lin e Shiueng (2001) que como veremos apesar da idéia ser muito boa, a forma de definir os seus passos é extremamente instável. No CLUSTERING calcula-se, para cada ponto *i*, distância euclidiana entre este ponto e seu vizinho mais próximo, denotada $d_v(i)$ e em seguida toma-se a média destas distâncias, dada por $\ell = \sum_{i=1}^{N} d_v(i)$ para $i = 1 \ldots N$, onde *N* é o tamanho do conjunto de entrada. A visão de vizinhança do algoritmo é dependente de um parâmetro de entrada α, que será usado para a construção da matriz de adjacência *M* do conjunto de entrada. A matriz M é gerada de acordo com a equação (5.3.9).

$$M[i,\ j] = \begin{cases} 1 & se \quad d_v(i,\ j) \leq \alpha . \ell \\ o & caso \quad contrário \end{cases} \qquad (5.3.9)$$

Da matriz de adjacência *M*, é construído o conjunto $X = \{C_1, ...C_m\}$, onde cada C_i é um componente conexo obtido do grafo associado a *M*, tal qual foi feito no AEC-RC. O algoritmo construirá sua população como *strings* binárias de tamanho *m*, nas quais a ocorrência de 1 numa dada posição *i* da cadeia indica que o i-ésimo componente conexo iniciará um clusters na solução final. O CLUSTERING tem a função de aptidão *f* dada segundo a relação entre o somatório das distancias entre cada ponto e o centróide do cluster ao qual pertence e a distância entre cada ponto e o centróide dos demais clusters, conforme a equação (5.3.10), onde podemos ver a existência de um parâmetro de entrada *w*.

$$f(s) = \sum_{i=1}^{k} D_{\text{int } er}(C_i) * w - D_{\text{int } ra}(C_i) \qquad (5.3.10)$$

O CLUSTERING recebe como parâmetros de entrada dois valores reais w_s e w_l que definem o intervalo $[w_s,w_l]$ em que o parâmetro *w* da equação (5.3.10) assumirá na busca da melhor clusterização, ou seja, aquela com o número ideal de clusters. Na heurística de busca do melhor valor para o parâmetro *w* da equação (5.3.10), o módulo genético é chamado inicialmente para $w = w_s$, depois para $w = w_l$ e posteriormente para $w = w_m = (w_s + w_l) / 2$. Os parâmetros usados do CLUSTERING foram os mesmos sugeridos pelos autores, ou seja, tamanho da população igual a 50 indivíduos, número de gerações igual a 100, taxa de crossover igual a 85% e taxa de mutação igual a 5%. No caso do AEC e do AEC-RC, apenas o número de indivíduos foi diferente do CLUSTERING, tendo sido fixado em 10 indivíduos. Utilizamos como medida de qualidade da solução o valor apresentado pela solução de cada algoritmo expresso pela função de aptidão utilizada pelo AGS, já que a solução expressa por esta função está tão próxima da solução ideal quanto seu valor se aproximar de 1. Desta forma, uma vez que a função de aptidão usada pelo algoritmo da literatura (CLUSTERING) na mesma função usada pelo AGS, após o mesmo ter encontrado a solução, esta é avaliada para que seu valor seja expresso com base na equação (5.3.8). Os resultados apresentados na tabela (5.3.4) apresentam a média dos valores das 100 soluções obtidas por cada algoritmo. Foram utilizadas 17 instâncias para verificar a eficiência do AGS, deste total, quatro instâncias são da literatura e 13 foram geradas artificialmente de forma a gerar clusters com formatos variados e com densidades variadas. Das quatro instâncias da literatura, listadas em negrito na tabela 1, duas são artificiais no espaço R^2 a saber: Ruspini e 200p Kaufman (1989) [22]. A base Ruspini é composta de 75 pontos distribuídos em quatro clusters. Já a base de dados 200p consiste de uma distribuição normal onde cada ponto p_i é um vetor (μ_x, μ_y) com desvio padrão ρ para *x* e *y*. A distribuição gera três grupos conforme segue.

Grupo 1: 120 pontos	μ_x, = 0	μ_y = 10	ρ 1.7
Grupo 2: 60 pontos	μ_x, = 20	μ_y = 12	ρ 0.7
Grupo 3: 20 pontos	μ_x, = 10	μ_y = 20	ρ 1.0

As outras duas instâncias da literatura são bases reais. A instância Winconsin Breast Cancer Database (Mangasarian (1990)) [veja [38]] é uma base de dados composta de 699 pontos formados por 9 atributos que caracterizam tumores no tratamento de câncer. Dos 699 pontos, 16 foram excluídos da base por apresentarem atributos ausentes. Desta forma, o conjunto de pontos efetivamente avaliado é composto de 683 elementos. A última instância da literatura é a Iris Plants Database (R. A. Fisher (1936)) (veja [38], composta de 150 pontos no espaço R^4 divididos em três grupos de 50 pontos. As demais instâncias da tabela 5.3.4, foram geradas aleatoriamente. Para efeito de comparação dos resultados, a tabela 5.3.4 apresenta desempenho médio do AEC e do AEC-RC além do desempenho do CLUSTERING. Nesta análise, cada algoritmo executou 100 vezes e a média foi calculada para gerar a tabela. Os valores em negrito indicam o melhor desempenho, já que a função objetivo é de maximização.

Instância	**AEC-RC**	**AEC**	**CLUSTERING**
200p	**0.823**	0.823	0.741
CancerData	**0.596**	0.592	0.374
IrisData	**0.686**	**0.686**	0.651
RuspiniData	**0.737**	**0.737**	0.683
1000p6c	**0.727**	0.708	0.367
157p	**0.667**	0.666	0.657
2000p11c	**0.611**	0.602	0.287
2face	**0.666**	**0.666**	0.513
350p5c	**0.758**	**0.758**	0.568
3dens	**0.762**	**0.762**	0.742
97p	**0.710**	**0.710**	0.706
Convdensity	**0.854**	**0.854**	0.818
Covexo	**0.667**	**0.667**	0.618
Face	**0.511**	**0.511**	0.402
Moreshapes	**0.725**	0.720	0.436
Numbers	**0.542**	0.540	0.417
Numbers2	**0.565**	0.562	0.527

Tabela 5.3.4: Desempenho médio dos algoritmos avaliados.

Na análise dos resultados obtidos, observamos a eficiência do AEC e principalmente do AEC-RC em termos de encontrar o número ideal de clusters, a sua robustez e a

qualidade da solução gerada. Podemos ver que tanto o AEC como o AEC-RC apresentaram desempenho bem superior ao algoritmo da literatura. Além disso, podemos ver que a RC incorporado pelo AEC-RC implicou em melhora do desempenho em 8 das 17 instâncias testadas. Para efeito de verificação da robustez do algoritmo proposto, efetuamos também uma análise probabilística sugerida em Aiex et al. (2003)[1]. Esta análise se justifica por uma das limitações clássicas de heurísticas e algumas metaheurísticas que é a sua grande sensibilidade com os parâmetros de entrada do problema. Ou seja, o desempenho destes algoritmos pode variar muito de uma instancia a outra. Para verificarmos a robustez dos métodos aqui propostos, colocamos como critério de parada para todos os algoritmos, um valor alvo (valor de MQ), obtido de simulações preliminares destes métodos para cada instância. O alvo pode ser uma média dos valores obtidos, ou um dos melhores valores, se queremos alvos mais difíceis. Definido o alvo, cada algoritmo é executado m vezes para cada instância selecionada. A cada execução i, armazena-se o tempo de cpu ti, e uma probabilidade $pi = (i - ½).m$ (no nosso caso m = 100). Com isso geramos pontos no R^2 da forma zi = (ti,pi). Como forma de avaliar a eficiência do algoritmo proposto, a figura 1 mostra a análise probabilística do AEC, do AEC-RC e do CLUSTERING, além das versões AEC-SC e AEC-RCSC, que são o AEC e AEC-RC sem a heurística de construção.

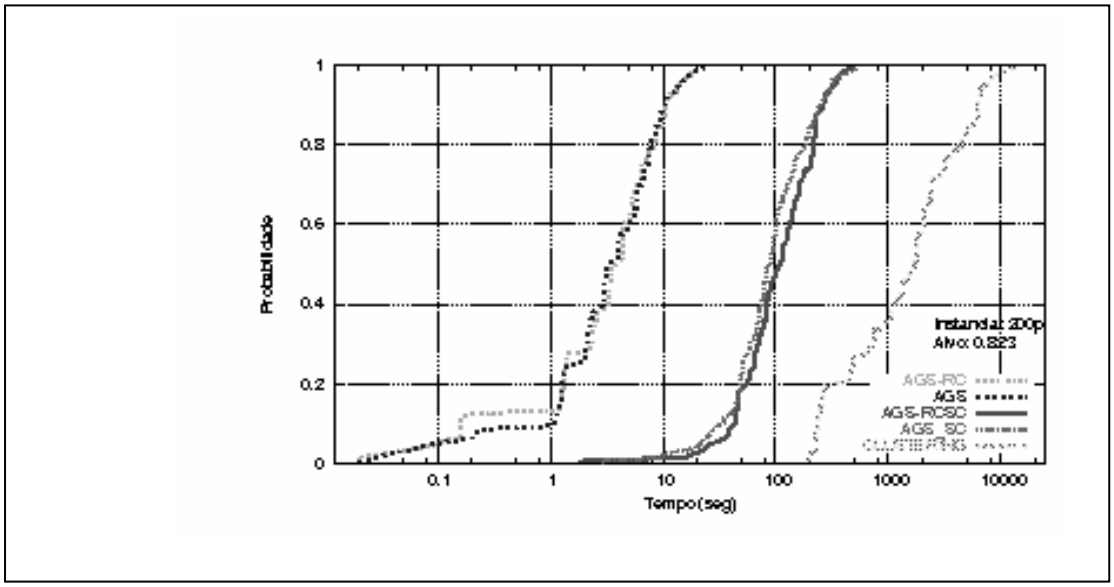

Figura 5.3.1: Análise probabilístico para instância 200p.

Podemos ver pela figura 5.3.1 que o AEC e o AEC-RC apresentam 100% de chance de chegar ao alvo antes dos primeiros 100 segundos de processamento (as duas curvas mais a esquerda). Já as versões sem a heurística de construção AEC-SC e AEC-RCSC para esta probabilidade precisam de mais de 500 segundos de processamento (duas curvas intermediárias). Ainda assim, as versões sem construção são melhores que o algoritmo da literatura, que para esta probabilidade precisa de mais de 10.000 segundos (curva mais a direita). Isto reflete a robustez dos algoritmos propostos.

Conclusões

Nesta seção apresentamos um algoritmo de clusterização hierárquica para o PCA que, para todas as instâncias submetidas, apresentou resultados de boa qualidade em termos de otimização da função objetivo, do número ideal de clusters e tempo de processamento. Além disso, nossa abordagem enquanto reduz a cardinalidade do conjunto de entrada, tira proveito desta redução para abreviar o processo de busca, fato este comprovado pela capacidade do algoritmo de encontrar a solução nas primeiras gerações mesmo numa população pequena de indivíduos.

6. Conclusões Gerais

Apresentamos neste trabalho, uma introdução aos problemas de clusterização na Mineração de Dados (MD). Iniciamos com uma breve descrição e classificação básica do mesmo, seguido de algumas aplicações de nosso interesse. Para algumas destas aplicações, mostramos como obter soluções aproximadas de boa qualidade usando conceitos da metaheurística Evolutiva. Maiores informações sobre este tema pode ser obtido pelos interessados (as) através de msgs para um dos autores ou na página: http://www.ic.uff.br/LabIC.

7. Bibliografia

[1] Aiex, R., Binato, S., e Resende (2003), M. Parallel GRASP with path-relinking for job shop scheduling. Parallel Computing 29, 393 - 430.

[2] N. Aljaber, W. Baek and C.-L. Chen (1997), A Tabu Search Approach to the Cell Formation Problem, in Computers and Industrial Engineering, Vol. 32(1), 169-185.

[3] Bäck, T. Optimization by Means of Genetic Algorithms. In *36th Int. Scientific Colloquium*, Technical University of Ilmenau, ed. E. Köhler, pp. 163-169, 1991.

[4] Battiti, R. e Bertossi, A. Greedy, Prohibition, and Reactive Heuristics for Graph Partitioning. *IEEE Transactions on Computers*, 1999.

[5] Battiti, R., Bertossi, A. e Cappelletti, A. Multilevel Reactive Tabu Search for Graph Partitioning. *Preprint UTM 554, Dip. Mat.*, Univ. Trento, Itália, 1999.

[6] Beasley, D., Bull, D. R. e Martin R. R. An Overview of Genetic Algorithms: Part I, Fundamentals. *University Computing*, vol. 15, no. 2, pp. 58-69, 1993.

[7] Ben-Dor, A., Shamir, R. e Yakhini, Z. Clustering gene expression patterns. *Journal of Computational Biology*, vol. 6, pp. 281-297, 1998.

[8] Berkhin, P. *Survey of Clustering Data Mining Techniques*. Accrue Software, 2002.

[9] Chiun, Y. e Lan, L. W. Genetic Clustering Algorithms. *European Journal of Operational Research* (135) 2, pp. 413-427, 2001.

[10] Dias, C. R. and Ochi, L. S. Improving the performance of Evolutionary Algorithms for the Directed Graph Partitioning. In Proc. of the XII CLAIO (em CD-ROM), Havana, CUBA, 2004.

[11] Dias, C. R. and Ochi, L. S. Efficient Evolutionary Algorithms for the Clustring Problem. In Proc. of the 2003 IEEE Congress on Evolutionary Computation (IEEE-CEC 2003) (em CD-ROM), Camberra, Australia.

[12] Dias, C. R. (Dissertação de Mestrado) Algoritmos Evolutivos para o Problema de Clusterização em Grafos Orientados: Desenvolvimento e Análise Experimental. Orientador: Luiz Satoru Ochi. Programa de Pós Grad. em Computação, IC/UFF, 2004.

[13] Doval, D., Mancoridis, S. e Mitchell, B. S. Automatic Clustering of Software Systems using a Genetic Algorithm. In *1999 International Conference on Software Tools and Engineering Practice (STEP '99)*, 1999.

[14] Drummond, L. M. A., Ochi, L. S., e Figueiredo, R. M. V. Design and Implementation of a Parallel Genetic Algorithm for the Travelling Purchaser Problem. In *Applied Computing/ACM*, pp. 257-263, 1997.

[15] Fasulo, D. An Analysis of Recent Work on Clustering Algorithms. Technical Report, Dept. of Computer Science and Engineering, Univ. of Washington, 1999.

[16] Glover, F. e Laguna, M. Fundamentals of Scatter Search and Path Relinking. *Control and Cybernetics*, vol 29, no. 3, pp. 653-684, 2000.

[17] Glover, F., Laguna, M. e Martí, R. *Scatter Search*. Springer-Verlag New York, Inc. New York, NY, USA, 2003.

[18] Gonçalves, J. F. and Resende, M. G. C. A hybrid genetic algorithm for manufacturing cell formation, Tec. Report, Fac. de Eng. do Porto, Portugal. (submitted to Comp. & Industrial Eng), 2002.

[19] Joines, J. A., Culbreth, C. T., and King, R. E. Manufacturing Cell Design: An Integer Programming Model Employing Genetic Algorithms. Tec. Report, Department of Industrial Engineering, North Caroline State University, NC 27695-7906, 1996.

[20] Goldberg, D. E. *Genetic Algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.

[21] Holland, J. H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[22] Kaufman, L. and Rousseuw , P. J. Finding Groups in Data - An Introduction to Cluster Analyses, Wiley Series in Probability and Mathematical Statistics, John Wiley and Sons, 1989.

[23] Koza, J. R. Hierarchical Genetic Algorithms Operating on Populations of Computer Programs. In *11th International Joint Conference on Artificial Inteligence*, ed. N. S. Sridharan, pp. 768-774, 1989.

[24] Laguna, M. Scatter Search. In *Handbook of Applied Optimization*, eds. P. M. Pardalos e M. G. C. Resende, Oxford University Press, pp. 183-193, 2002.

[25] Lorena, L. A. N. e Furtado, J. C. Constructive Genetic Algorithm for Clustering Problems. *Evolutionary Computation*, vol. 9, no. 3, pp. 309-327, 2001.

[26] Maini, H. S., Mehrotra, K. G., Mohan, C. K. e Ranka, S. Genetic Algorithms for Graph Partitioning and Incremental Graph Partitioning. In *Proceedings of the 1994 Conference on Supercomputing*, pp. 449-457, 1994.

[27] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1992.

[28] Michalewicz, Z., Xiao, J. e Trojanowski, K. Evolutionary Computation: One Project, Many Directions. In *Proceedings of the 9th International Symposium (ISMIS '96)*, pp.189-201, 1996.

[29] Moscato, P. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report, Caltech Concurrent Computation Program, California Institute of Technology, 1989.

[30] Ochi, L. S. and Vieyra, P. W. P. A Hybrid Metaheuristic using Genetic Algorithm and Ant Colony Systems for the Clustered Traveling Salesman Problem. In Proc. of the III Metaheuristic Int. Conference (III MIC), Angra dos Reis, RJ, 1999.

[31] Ochi, L. S., Vianna, D. S., and Drummond, L. M. A., A Parallel Hybrid Evolutionary Algorithm for the Vehicle Routing Problem. In Lectures Notes in Computer Science (LNCS), vol. 1586, pp. 183-192, Springer, 1999.

[32] Ochi, L. S., Vianna, D. S., and Drummond, L. M. A., A Parallel Evolutionary Algorithm for the Vehicle Routing Problem with Heterogeneous Fleet. In Lectures Notes in Computer Science (LNCS), vol. 1388, pp. 216-225, Springer, 1998.

[33] Ochi, L. S., Dalboni, F. L. ., and Drummond, L. M. A., On Improving Evolutionary Algorithms by Data Mining for the Oil Collector Vehicle Routing Problem. In Proc. of the Int. Network Optimization Conference (INOC 2003), pp. 182-188, Evry, Paris, INFORMS, 2003.

[34] Ochi, L. S., and Rocha, M. L. A New Evolutionary Algorithm for the Vehicle Routing and Scheduling Problems. In Ninth Int. Conference on Intelligence Systems, Louisville, Kentucky, 2002..

[35] Ochi, L. S., Drummond, L. M. A., Vianna, D. S., "An asynchronous parallel metaheuristic for the period vehicle routing problem", Future Generations on Comp. Systems, Elsevier, pp. 379-386, vol 17(4), 2001.

[36] Radcliffe, N. J. e Surry, P. D. Formal Memetic Algorithms. In *Evolutionary Computing: AISB Workshop, Lecture Notes in Computer Science*, vol 865, pp. 1-16, Springer-Verlag, 1994.

[37] Santos, H. G., Merschmann, L. H., Ochi, L. S., and Drummond, L. M. A., An Improving Evolutionary Algorithm with Data Mining for a Vehicle Routing Problem. In Proc. Of the VIII Brazilian Symposium on Neural Networks (SBRN) (IEEE Press, Em CD-ROM), São Luis, MA, 2004.

[38] Soares, S. S. F. (Dissertação de Mestrado) Metaheurísticas para o Problema de Clusterização Automática . Orientador: Luiz Satoru Ochi. Programa de Pós Grad. em Computação, IC/UFF, 2004.

[39] Trindade, A. R., and Ochi, L. S. An efficient evolutionary algorithm for the manufacturing cell design problem. In Proc. (CD-ROM) of the XII CLAIO, Havana, Cuba, 2004.

[40] Trindade, A. R. (Dissertação de Mestrado) Metaheuristicas para o Problema de Clusterização de Células de Manufatura . Orientador: Luiz Satoru Ochi. Programa de Pós Grad. em Computação, IC/UFF, 2004.

[41] Toussaint, G., The relative neighborhood graph of a finite planar set, *Patt. Recognition,* 12, 261 – 268, 1980.

[42] Vianna, L. S., (Dissertação de Mestrado) Metaheuristicas Paralelas para os Problemas de Escalonamento de Tarefas e Roteamento de Ve´ciulos. Orientadores: Luiz Satoru Ochi e L. Drummond. Programa de Pós Grad. em Computação, IC/UFF, 2002

[43] Wang, J. A Linear Assignment Algorithm for Formation of Machine Cells and Parts Families in Cellular Manufacturing, in Computers Ind. Engineering, vol 35(1-2), 81-84, 1998.