Agrupamento de Sistemas Orientados a Objetos com Metaheurísticas Evolutivas

André Luis Vasconcellos Botelho[1], Gustavo Silva Semaan[2], Luiz Satoru Ochi[2]

[1]Centro de Ensino Superior de Juiz de Fora (CESJF) – Juiz de Fora – MG – Brasil

[2] Instituto de Computação – Universidade Federal Fluminense (UFF) – Niterói – RJ – Brasil

alvbotelho@gmail.com, {gsemaan, satoru}@ic.uff.br

Abstract. *Several Software Engineering problems presenting high complexity and combinatorial possibilities and can be modeled as optimization problems. This paper proposes an evolutionary metaheuristic to solve a problem of clustering object-oriented information systems. The algorithms may use two versions of contructor procedures, a local search and path relinking. The computational results showed that the use of these algorithms is an efficient way to solve this problem, in which procedures achieve quality solutions in a reduced runtime.*

Resumo. *Diversos Problemas da Engenharia de Software, por possuírem elevadas complexidade e possibilidades combinatórias, podem ser modelados como problemas de otimização. Este trabalho apresenta uma proposta para a clusterização de sistemas orientados a objetos mediante a utilização de algoritmos evolutivos. Os algoritmos propostos podem utilizar duas versões de procedimentos construtivos, uma busca local e reconexão de caminhos. Os resultados computacionais mostraram que a utilização dos algoritmos propostos é uma alternativa eficiente para a resolução deste problema de clusterização, em que procedimentos alcançam soluções de qualidade em um reduzido tempo de execução.*

1. Introdução

Diversos Problemas da Engenharia de *Software* (ES), por possuírem elevadas complexidade e possibilidades combinatórias, podem ser modelados como problemas de otimização (PO). As resoluções desses problemas sugerem algoritmos metaheurísticos, uma vez que métodos exaustivos podem ser ineficientes. A modelagem de problemas de ES em PO deu origem a uma nova e promissora área de pesquisa, denominada *Search-based Software Engineering* (SBSE), que aborda temas da ES como: Engenharia de Requisitos, Teste, Manutenção e Estimativas de *Software*, Planejamento de Projeto, Otimização de Código-Fonte dentre outros (Freitas et al. 2009).

Um dos maiores desafios encontrados por uma equipe de desenvolvimento de sistemas é realizar manutenções em softwares complexos, principalmente em sistemas que não possuem documentação ou ela não é suficiente (Doval et al. 1999).

Nesse contexto, o presente trabalho propõe algoritmos metaheurísticos para a resolução de um problema de clusterização de sistemas de informação orientados a objetos. Tal problema consiste na Modularização do *Software*, baseada em sua

arquitetura. Assim, como diversos problemas de otimização, este problema de clusterização, também conhecido como problema de agrupamento, foi mapeado em um problema de particionamento de grafos em subgrafos disjuntos (*clusters*).

Durante a revisão da literatura observou-se que algoritmos foram propostos com base em sistemas de informação estruturados, em que Diagramas de Fluxo de Dados (DFDs) são representados por dígrafos sem a utilização de pesos nas arestas. Em Doval *et al.* (1999) foram apresentados algoritmos para modularizar sistemas de informação desenvolvidos utilizando o paradigma estruturado.

Dessa forma, após a divisão do sistema, suas partes seriam distribuídas entre equipes de desenvolvimento, descentralizando as atividades, reduzindo a comunicação entre equipes e aumentando a comunicação entre desenvolvedores em uma mesma equipe.

Conforme Berkhin (2002), um problema de clusterização é caracterizado por agrupar elementos de uma base dados em subconjuntos disjuntos (*clusters*) objetivando a maximização da similaridade dos elementos de um mesmo *cluster* e a minimização da similaridade entre os elementos de *clusters* diferentes. Objetivando a formalização do problema, pode-se descrevê-lo como, tendo um conjunto com n elementos $X = \{x_1,...,x_n\}$, encontrar partições desse conjunto (*Clusters* C_i) respeitando as seguintes condições:

$\bigcup_{i=1}^{k} C_i = X$	O conjunto submetido ao problema corresponde à união dos elementos dos *clusters*.
$C_i \bigcap C_j = \varnothing$ i,j=1, .., k e i $\neq$ j	Cada elemento pertence a somente um *cluster*.
$C_i \neq \varnothing$ i=1, .., k	Todos os *clusters* possuem ao menos um elemento.

Os problemas de clusterização podem ser reconhecidos em duas formas distintas. Na primeira, a quantidade de *clusters* da solução é conhecido, caracterizando um Problema de K-Clusterização ou apenas Problema de Clusterização (PC) (Semaan, et al., 2009). No segundo, a quantidade ideal de *clusters* não é conhecida, faz parte da solução, e denomina-se Problema de Clusterização Automática (PCA) (Berkhin, P., 2002; Dias, C. R., 2003). A quantidade de soluções viáveis em um PC pode ser definida pela Fórmula 1, em que n representa a quantidade de elementos e k a quantidade de *clusters*, enquanto em um PCA pode ser obtida pela Fórmula 2.

A fim de demonstrar o crescimento da quantidade de soluções possíveis em PC e PCA, Dias (2004) explica em seu trabalho que a primeira fórmula denomina que, para cada 10 elementos distribuídos em 2 *clusters* consegue-se 511 soluções diferentes. Para 100 elementos em 2 clusters, tem-se $6{,}33825 \cdot 10^{29}$. Para 100 elementos em 5 *clusters* encontra-se $6{,}57384 \cdot 10^{67}$. Já no caso de 100 elementos em 2 *clusters*, pode-se encontrar $5{,}3575 \cdot 10^{300}$ formas diferentes de resolver o problema. Na utilização da fórmula para o PCA, para um conjunto de 10 elementos, são encontradas 115.975 combinações diferentes e a quantidade de *clusters* varia de 1 a 10.

Esse trabalho apresenta novas heurísticas para agrupar unidades de sistemas de

informação orientados a objetos.

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

Fórmula 1: Problema de K-Clusterização.

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

Fórmula 2: Problema de Clusterização Automático

2. Particionamento de Grafos Ponderados

Um problema de particionamento de grafos consiste em, a partir de um grafo, agrupar os vértices em *clusters*, otimizando uma função-objetivo. Para a abordagem com *software* orientado a objetos, esse trabalho propõe uma modelagem que utiliza arestas não direcionadas e ponderadas.

Com o objetivo de resolver o problema abordado, foi utilizado um mapeamento em um grafo *G (V, E)*, em que: os vértices *V* de *G* representam unidades de *software* orientado a objetos, como: classes concretas, abstratas, *interfaces* e enumerados. Além disso, as arestas *E* de *G* possuem pesos que indicam diferentes níveis de relações entre as unidades do sistema, tais como: associações simples, composições, agregações, implementações e heranças. Desta forma, o relacionamento de herança seria mais forte, possuindo pesos maiores que associações simples. A Figura 1 apresenta um diagrama de classes e a Figura 2 o grafo construído com base no diagrama de classes da Figura 1.

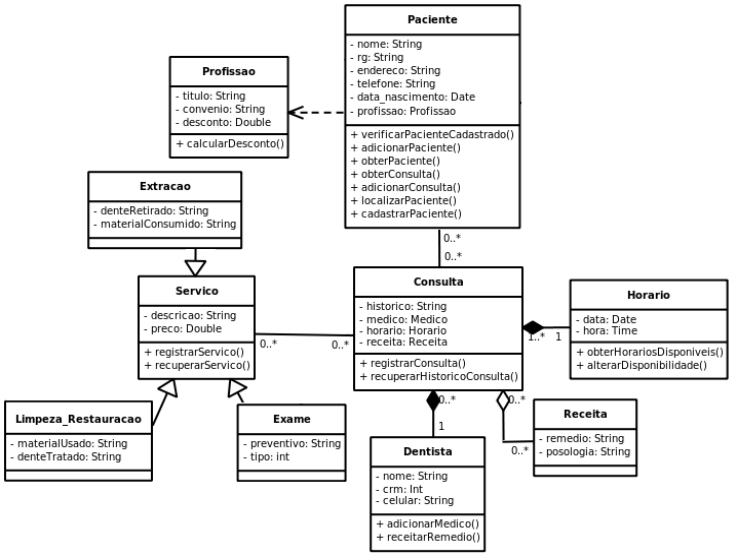

Figura 1: Um exemplo de diagrama de classes.

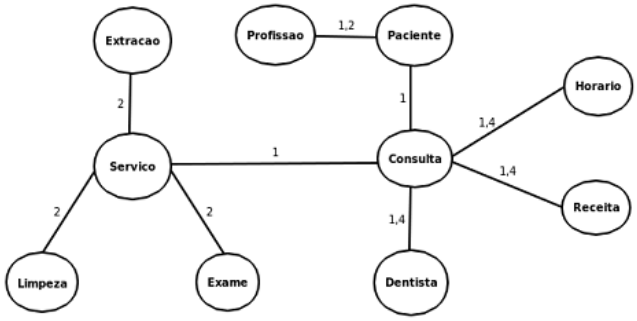

Figura 2: Grafo gerado a partir do diagrama de classes.

3. Algoritmos

De acordo com Glover et al.(2003) e Santos (2006), os algoritmos evolutivos (AE) e algoritmos genéticos (AG) são bastante utilizados na área de inteligência computacional e foram inspirados na teoria da evolução natural de Charles Darwin e nos princípios da genética de Gregor Mendel. Estes algoritmos trabalham na busca de soluções para problemas complexos utilizando técnicas inspiradas nos conceitos citados anteriormente.

3.1. Representação da Solução

Segundo Doval et al. (1999), uma boa representação da solução é extremamente importante, podendo ser um fator determinante na performance do algoritmo, ou seja, para a rápida convergência e qualidade das soluções obtidas. Neste trabalho foi utilizada a representação *group-number*, que utiliza um vetor onde seu índice representa o vértice do grafo. Já o valor apontado por este índice representa o *cluster* a que pertence a solução (Berkhin, P., 2002).

3.2. Procedimentos Construtores

Uma vez analisada e definida a representação que será utilizada, deve-se construir um conjunto de soluções iniciais necessárias ao funcionamento de um AG. Neste trabalho foram utilizados dois métodos construtores de soluções. A primeira versão constrói soluções de forma totalmente aleatória, enquanto a segunda versão utiliza uma heurística que objetiva a formação indivíduos de melhor qualidade. Para isto, de acordo com o peso das arestas seus vértices pertencerão a *clusters* diferentes ou ao mesmo *cluster*.

```
1. para i = 0 até i = total de vértices faça
2.         solucao[i] = número aleatório(total de vértices);
3. fim-para
```

Figura 3: Algoritmo referente ao primeiro método construtivo.

```
1. para i = 0 até i = total de vértices faça
2.         solucao[i] = número aleatório(total de vértices);
3. fim-para
4. para i = 0 até total_arestas faça
5.         i_auxiliar = buscar valor da aresta do vértice de saída;
6.         j_auxiliar = buscar valor da aresta do vértice de entrada;
7.         se ExisteAresta(i_auxiliar, j_auxiliar)então
8.                 se ConexaoForte(i_auxiliar, j_auxiliar) então
```

```
9.                          AgruparEmMesmoCluster(i_auxiliar, j_auxiliar)
10.                 senão
11.                         InserirClustersDistintos(i_auxiliar, j_auxiliar)
12.                 fim-se
13.         fim-se
14. fim-para
```

Figura 4: Algoritmo do segundo método construtor.

3.3. Seleção

A cada iteração de um AG, deve-se obter uma nova população de indivíduos onde serão aplicados os operadores genéticos. Entretanto, após a avaliação dos indivíduos (soluções), o algoritmo deve selecionar os melhores para que esses permaneçam na próxima geração.

Nesse trabalho foi utilizada a seleção por torneio de duas soluções, técnica baseada na seleção proporcional à aptidão criada por Holland (1975). Nela, o algoritmo seleciona duas soluções de forma aleatória e mantém a de melhor qualidade na próxima iteração. Também foi utilizado o elitismo, método que armazena o melhor indivíduo da população atual a fim de propagar suas características à geração seguinte (Glover *et al.*, 2003).

3.4. Operadores Genéticos

De acordo com Glover *et al.*(2003), Dias (2003); Santos *et al.* (2006), Holland (1975), os operadores genéticos são importantes métodos utilizados em AGs para evitar que os algoritmos fiquem estagnados em soluções ótimas locais. O operador de cruzamento é a principal força direcionadora em um AG tradicional, e realiza trocas entre partes de um par de soluções (Goldberg, D. E., 1989). Os algoritmos propostos nesse trabalho utilizaram o cruzamento de um ponto, conforme apresenta a Figura 5.

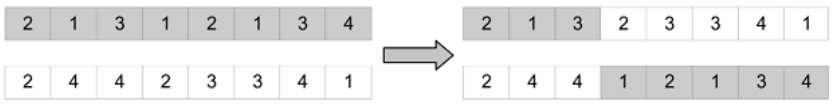

Figura 5: Operador genético de cruzamento de um ponto.

Já a mutação perturba a solução, alterando de forma aleatória valores atribuídos aos genes dos indivíduos. Na Figura 6 o vértice 6 que pertencia ao *cluster* 2 foi migrado para o *cluster* 3.

Figura 6: Operador genético de mutação.

3.5. Critério de Parada

Neste trabalho foram implementados diferentes critérios de parada dos algoritmos, utilizados conforme os experimentos: quantidades de iterações, tempo de execução ou alcance ao alvo, ou seja, valor da aptidão de uma solução na geração corrente ser no mínimo igual ao valor submetido como parâmetro ao algoritmo.

3.6. Busca Local

Conforme Glover et al. (2003), Semaan (2009), Dias (2003), Dias (2004) e Santos et al. (2006), a busca local avalia a vizinhança de uma solução com o objetivo de encontrar soluções de melhor qualidade.

```
1. para p = 0 até tamanho da população faça
2.         s = solução atual (p);
3.         para i = 0 até total de vértices do grafo faça
4.           enquanto aptidão de s > aptidão_aux faça
5.                     aptidão_aux = aptidão de s;
6.                     id = cluster na posição [i][i] da matriz de valores;
7.                     soma = somar arestas: vértice[i] e demais do mesmo cluster;
8.                     soma_aux = somar arestas: vértice[i] e demais dos outros cluster;
9.                     flag = 0;
10.                    para j = 0 ATÉ total de vértices do grafo FAÇA
11.                              se (soma < soma_aux[j]) ENTÃO
12.                                        soma = soma_aux;
13.                                        flag = j;
14.                              fim-se
15.                              se (id <> flag) então
16.                                        Altera o cluster de s;
17.                                        Calcula nova aptidão;
18.                              fim-se
19.                    fim-para
20.          fim-enquanto
21.        fim-para
22. fim-para
```

Figura 7: Algoritmo de Busca Local

3.7. Reconexão de Caminhos

A reconexão de caminhos é uma técnica que une intensificação e diversificação que atua na busca por soluções melhores (Glover, 1996; Glover *et al.*, 2000 e Glover *et al.* 2003). Nesta técnica, o algoritmo usa duas soluções para que, partindo dos dados encontrados na primeira, ele se recombine até que fique semelhante a segunda, avaliando todas as encontradas neste caminho. Assim, o algoritmo trabalha diversificando as soluções e buscando em espaços de soluções diferentes.

3.8. Função de Aptidão

Para a avaliação das soluções, a mesma função de aptidão dos trabalhos da revisão da literatura, denominada Qualidade de Modularização (*Modularization Quality* - MQ) foi utilizada. Houve, porém, a necessidade de adaptá-la para considerar os pesos associados às arestas. Essa função corresponde, basicamente, à diferença entre a conectividade interna e a externa dos elementos nos clusters.

Com o objetivo de avaliar a qualidade da solução, esta função premia *clusters* com alta intra-conectividade (conexões entre elementos de um mesmo *cluster*) e, ao mesmo tempo, penaliza a inter-conectividade (muitas conexões entre elementos de *clusters* distintos). Proposta por Doval et al. (1999), os valores dessa função pertencem ao intervalo [-1,1] e o objetivo desse problema é maximizá-la.

A Fórmula 3 é utilizada para calcular a intraconectividade de um *cluster i*, em que μi é o total de arcos que possuem em uma das extremidades algum vértice do

cluster i e *Ni* é a quantidade de vértices do *cluster*. Já a interconectividade mensura a conectividade entre vértices de *clusters* distintos. A Fórmula 4 é utilizada para calcular a interconectividade entre dois *clusters*, em que *Ni* é o total de vértices do *cluster i*, *Nj* é o total de vértices do *cluster j* e *eij* o total de arcos que possuem as extremidades em *i* e *j*.

$$A_i = \frac{\mu_i}{N_i^2}$$

Fórmula 3: Intraconectividade.

$$B_{i,j} = \begin{cases} 0 \text{ se i=j} \\ \dfrac{\varepsilon_{ij}}{2N_iN_j} \; se \text{ i} \neq \text{j} \end{cases}$$

Fórmula 4.: Interconectividade.

$$MQ = \begin{cases} \dfrac{\sum_{i=1}^{k} A_i}{k} - \dfrac{\sum_{i,j=1}^{k} B_{i,j}}{\frac{k(k-1)}{2}} \text{ para k>1} \\ Ai \text{ para k=1} \end{cases}$$

Fórmula 5: Qualidade da modularização.

4. Experimentos

Diversos experimentos foram realizados utilizando dados reais, por meio de códigos fonte de sistemas de informações mapeados, e artificiais, com a utilização de instâncias existentes na literatura adaptadas ao modelo de grafos ponderados.

Tabela 1: As instâncias utilizadas e suas características

Código	1	2	3	4	5	6	7	8	9	10
Vértices	10	20	40	60	80	100	100	100	10	25
Arestas	27	104	226	1063	737	921	788	825	9	29

Os experimentos consideraram diferentes versões de algoritmos evolutivos, desde um algoritmo genético tradicional até algoritmos evolutivos que utilizam procedimentos de busca local, reconexão de caminhos e heurísticas de construção de soluções. A tabela abaixo representa as versões dos algoritmos utilizadas durante a confecção deste trabalho.

Tabela 2: As versões dos algoritmos utilizados.

	Versão							
	AE1	AE2	AE3	AE4	AE5	AE6	AE7	AE8
Versão do Construtivo	1	1	1	1	2	2	2	2
Busca Local			X	X			X	X
Reconexão de Caminhos		X		X		X		X

O primeiro experimento consiste na execução de cada algoritmo proposto em cada instância 100 vezes considerando os parâmetros: critério de parada de 100 iterações considerando o tempo máximo de execução de 10 minutos, o tamanho da população corresponde a 10 vezes a quantidade de vértices da instância, probabilidade de mutação e de cruzamento de 10% e 20% respectivamente. Essas configurações foram baseadas em trabalhos da revisão da literatura (Doval et al. 1999; Dias et al. 2003) e ajustados de forma empírica em experimentos anteriores.

A Figura 8 apresenta os resultados obtidos no primeiro experimento, considerando a função de aptidão, tempo de execução e *Gap* em relação ao melhor valor de aptidão (*f(x)Best*) conhecido (Fórmula 6). Já a Figura 9 apresenta a média dos

melhores resultados obtidos.

$$Gap = 100 * \frac{f(x) - f(x)_{best}}{f(x)_{best}}$$

Fórmula 6: *Gap* ou hiato.

A partir das Figuras 11 e 12 verifica-se que os AEs 3, 4, 7 e 8 são os que apresentam os melhores resultados. Entretanto, os AEs 4 e 8 foram os algoritmos que mais consumiram tempo de execução e, inclusive, os algoritmos foram finalizados por atingir o tempo máximo estabelecido.

		AE1		AE2		AE3		AE4		AE5		AE6		AE7		AE8	
Instância	Best F(x)	Gap %	T(s)	Gap %	T(s)	Gap %	T(s)	Gap %	T(s)	Gap %	T(s)	Gap %	T(s)	Gap %	T(s)	Gap %	T(s)
1	0,524	15,90	1	14,58	1	0,00	1	0,00	1	11,92	1	9,32	1	0,00	1	0,00	1
2	0,589	20,70	1	20,70	42	0,00	1	0,00	30	20,70	1	20,70	39	0,00	1	0,00	30
3	0,505	24,19	8	24,19	600	0,00	8	0,00	600	24,02	9	24,27	600	0,00	8	0,00	600
4	0,767	26,68	39	26,68	600	1,39	34	0,08	600	33,11	39	26,68	600	0,00	36	1,91	600
5	0,665	33,02	104	33,02	600	0,00	120	0,00	600	34,15	118	34,16	600	0,00	122	0,00	600
6	0,648	35,66	242	35,39	600	0,00	227	1,17	600	34,09	250	34,07	600	1,51	223	0,84	600
7	0,612	33,06	253	33,06	600	0,00	191	0,00	600	31,84	234	32,98	600	0,26	194	0,00	600
8	0,549	30,13	245	30,13	600	2,38	223	2,51	600	28,01	245	28,10	600	0,00	215	1,76	600
9	0,256	2,93	1	6,71	1	0,00	1	0,00	1	3,97	1	4,93	0	0,00	1	0,00	1
10	0,338	19,44	2	20,09	600	0,00	1	0,00	600	13,72	2	13,16	600	0,00	1	0,00	600

Figura 8: Resultados gerais obtidos.

		AE1		AE2		AE3		AE4		AE5		AE6		AE7		AE8	
Instância	Best F(x)	Gap %	T(s)	Gap %	T(s)	Gap %	T(s)	Gap %	T(s)	Gap %	T(s)	Gap %	T(s)	Gap %	T(s)	Gap %	T(s)
1	0,524	51,55	1	48,40	1	0,00	1	0,00	1	48,85	1	42,63	1	0,00	1	0,00	1
2	0,589	65,62	1	65,35	42	5,45	1	5,17	30	64,76	1	65,89	41	6,89	1	5,17	29
3	0,505	79,19	8	80,50	600	0,00	8	0,93	600	77,37	8	77,03	600	0,00	8	1,05	600
4	0,767	77,73	38	79,64	600	4,48	34	3,99	600	81,14	39	76,84	600	3,53	34	4,41	600
5	0,665	87,23	126	89,03	600	0,65	115	0,88	600	87,88	122	87,95	600	0,20	115	2,08	600
6	0,648	91,57	245	91,64	600	8,28	229	8,60	600	89,95	244	89,25	600	7,63	224	9,61	600
7	0,612	90,71	243	90,43	600	0,91	192	2,07	600	86,98	242	88,29	600	1,88	187	1,64	600
8	0,549	89,35	251	90,40	600	11,35	284	9,77	600	81,92	257	82,70	600	8,37	245	10,16	600
9	0,256	41,50	1	50,71	1	0,00	1	0,00	1	40,04	1	46,44	1	0,00	1	0,00	1
10	0,338	82,96	2	84,09	600	12,55	1	9,34	600	61,51	2	62,08	600	4,55	1	6,95	600

Figura 9: Comparativo entre as médias e os melhores resultados obtidos.

A Figura 10 apresenta os valores das melhores soluções obtidas em cada instância para cada algoritmo, em que, para todas as instâncias, houve grande superioridade dos algoritmos que utilizam busca local e reconexão de caminhos.

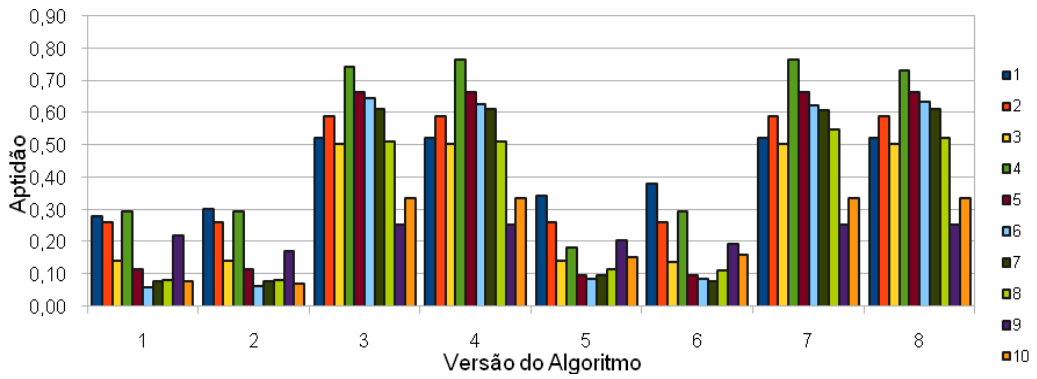

Figura 10: Variação dos resultados.

As Figuras 11 e 12 apresentam gráficos com os quantitativos das soluções obtidas por algoritmo, classificadas conforme sua qualidade para os melhores resultados e as médias dos resultados obtidos, considerando: *Best* (com *Gap* = 0%), Interessantes (com 0% < *Gap* < 10%) e Ruins (*Gap* > 50%). Além disso, foi considerado também o fim do tempo limite de execução. Uma vez que essas figuras apresentam os melhores resultados obtidos pelos algoritmos em cada instância, o valor máximo apresentado nessas figuras é a quantidade de instâncias (10 unidades).

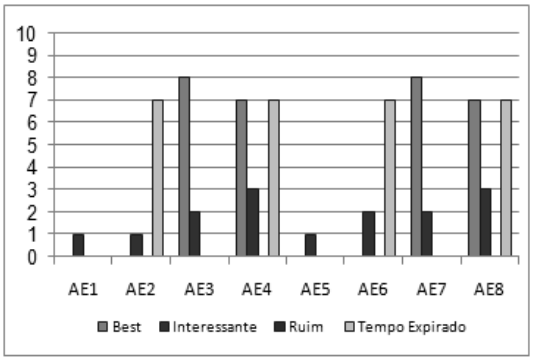

Figura 11: Quantitativos por Categorias - Melhores soluções por algoritmo.

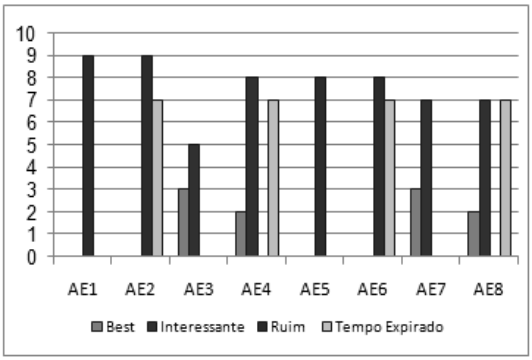

Figura 12: Quantitativos por Categoria - Média das soluções por algoritmo.

A Figura 13 demonstra, novamente, a superioridade dos algoritmos que utilizam as buscas locais e a reconexão de caminhos (AEs 3, 4, 7 e 8). Na Figura 14 é possível observar comportamento semelhante, porém nela os AEs 3 e 7 não expiraram o tempo de execução em nenhuma das execuções, o que ocorre em todos os algoritmos que utilizam reconexão de caminhos.

Com o objetivo de facilitar a visualização dos resultados, os dados foram agrupados conforme as técnicas utilizadas (procedimentos de construção, busca local e reconexão de caminhos). As Figuras 13 e 14 apresentam os resultados conforme a nova formatação.

Na Figura 14 observa-se que somente com a utilização da busca local foi possível a obtenção de soluções da categoria *Best*. Além disso, a reconexão de caminhos extrapola o tempo de execução limite submetido como parâmetro e o algoritmo termina sua execução. Verifica-se ainda, que a utilização de procedimentos construtores sem busca local e reconexão de caminhos não ultrapassam o tempo de execução, porém obtém poucas soluções da categoria interessante e nenhuma da categoria *Best*.

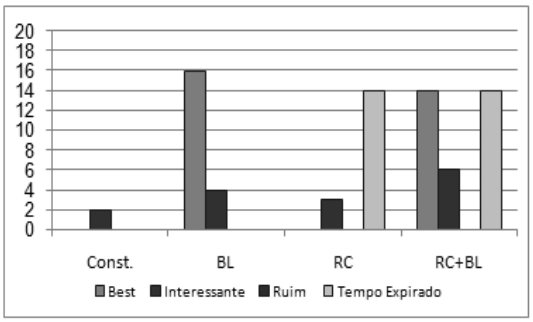

Figura 13: Quantitativos por técnicas - Melhores soluções.

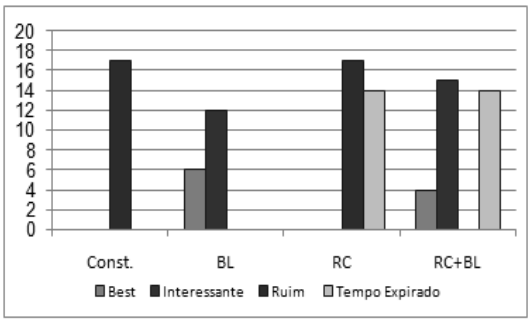

Figura 14: Quantitativos por Técnicas - Média das soluções.

Na Figura 14, referente as médias dos melhores resultados obtidos, pode-se verificar que os algoritmos que utilizaram apenas os construtores não alcançaram soluções de boa qualidade. Já os algoritmos que utilizaram a busca local foram os que conseguiram os melhores resultados, independente da utilização da reconexão de caminhos. Nos experimentos que utilizaram a reconexão de caminhos, entretanto, ocorreram em muitas execuções o fim de tempo máximo de execução. Além disso, a utilização da reconexão de caminhos sem a busca local, além do grande consumo de tempo não alcançou soluções de boa qualidade.

Para analisar de forma mais detalhada a robustez dos algoritmos propostos, as melhores versões foram submetidas a experimentos de análise de probabilidade empírica, e foram executadas 200 vezes em duas das maiores instâncias utilizadas nesse trabalho. Devido ao grande consumo de tempo de execução e resultados observados no experimento anterior, os algoritmos que utilizam reconexão de caminhos não foram utilizados.

Com os resultados obtidos, por meio de gráficos TTTPlot (*Time-To-Target Plots* (Aiex et al. 2009)), é possível verificar a probabilidade que cada algoritmo possui de alcançar os alvos (solução média e melhor solução obtida para cada instância) em função do tempo de execução.

As Figuras 15 e 16 apresentam os gráficos TTTPlot com resultados dos algoritmos AE1, AE3, AE5 e AE7 na busca pelos alvos difícil (melhor solução conhecida) e médio (média dos melhores resultados de todos os algoritmos) das instâncias 7 e 8, respectivamente. Na busca pelos alvos difíceis, para ambas as instâncias, os algoritmos AE1 e AE5 não registraram nenhuma ocorrência.

Na Figura 15 observa-se que, ao término do tempo de execução de (3000 segundos), os algoritmos AE1, AE3, AE5 e AE7 possuíram probabilidades de alcançar o alvo médio de respectivamente 20%, 100%, 44% e 100%. Em relação ao alvo difícil os algoritmos AE3 e AE7 alcançaram probabilidades de 85% e 80%, respectivamente.

Na Figura 16 ao término do tempo de execução de os algoritmos AE1, AE3, AE5 e AE7 resultaram probabilidades de alcançar o alvo médio de respectivamente 18%, 100%, 78% e 100%. Em relação ao alvo difícil os algoritmos AE3 e AE7 alcançaram probabilidades de 76% e 69%, respectivamente.

Com base nos experimentos realizados, os procedimentos construtivos e a reconexão de caminhos não foram determinantes para a obtenção dos melhores resultados. Em contrapartida, a utilização da busca local foi fundamental para a obtenção de soluções de boa qualidade, representadas pelas categorias *Best* e Interessante. Além disso, o experimento para alcance dos alvos confirmou a superioridade da busca local, em que somente os algoritmos que a utilizam foram capazes de alcançar os alvos difíceis.

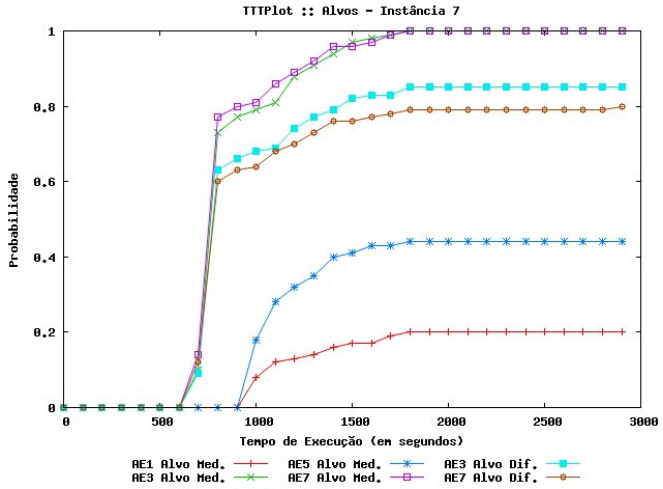

Figura 15: TTTPlot para as instâncias com alvos médio e difícil.

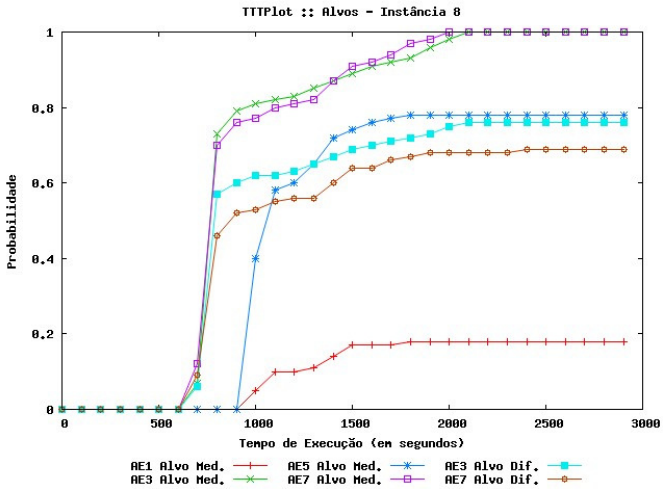

Figura 16: TTTPlot para as instâncias 8 com alvos médio e difícil.

5. Conclusões

Os resultados computacionais mostraram que a utilização dos algoritmos propostos é uma alternativa eficiente para a resolução deste problema de clusterização, em que os novos procedimentos alcançam soluções de melhor qualidade em um reduzido tempo de execução, sobretudo com a utilização da busca local proposta.

Pode-se concluir ainda que o algoritmo de reconexão de caminhos utilizado não foi uma boa solução para o problema apresentado, pois consumiu muito tempo e não conseguiu resultados satisfatórios.

O mesmo não pode ser dito sobre a busca local, pois foi o algoritmo que melhor encontrou os melhores resultados em um tempo computacional relativamente baixo.

Na Figura 15 pode-se verificar uma das soluções encontradas mais adequadas ao propósito deste trabalho. Como se trata de um problema de maximização, essa foi uma das soluções cuja função de aptidão obteve um dos maiores valores.

Nota-se que o algoritmo agrupou classes que possuem relacionamentos mais fortes, como as heranças predominantes no *cluster* 1. Por ser um problema de PCA, a melhor solução obtida possui apenas dois grupos, mas poderia possuir mais grupos, uma vez que obter a quantidade ideal desses faz parte da solução do problema.

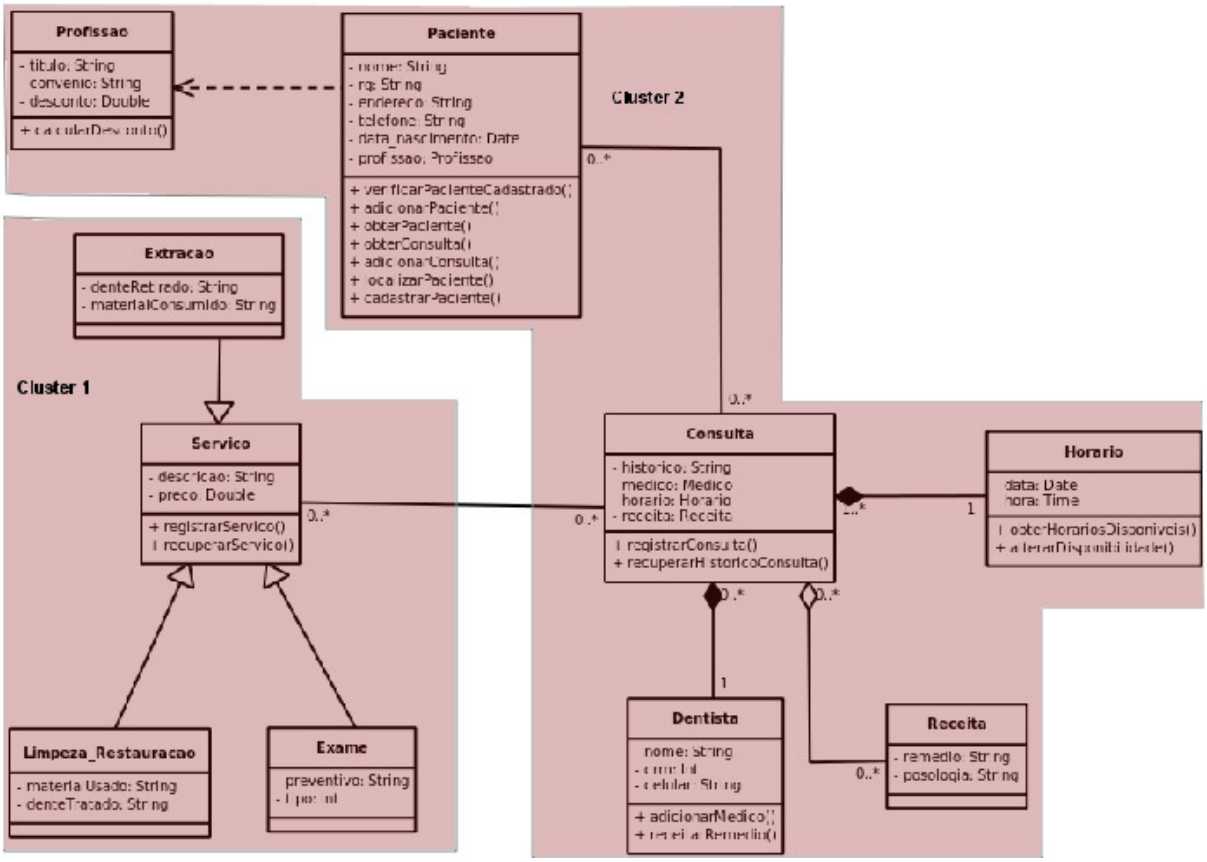

Figura 15: Representação de uma das melhores soluções encontradas.

6. Trabalhos Futuros

Durante a pesquisa e na realização de experimentos foram observados novos caminhos para trabalhos futuros, em que podemos destacar:

- Implementação de outras metaheurísticas, tais como Busca Tabu, ILS (*Iterated Local Search*) e VNS (*Variable Neighborhood Search*).
- Utilização de métodos exatos, como Programação Linear Inteira na busca por soluções ótimas, inclusive com algoritmos híbridos.
- Realização de experimentos com instâncias maiores, tanto artificiais quanto modeladas com base em códigos fontes de sistemas de Informações.

Referências

Aiex, R. M.; Resende, M. G. C.; Ribeiro, C. C. TTTPLOTS: A Perl program to create time-to-target plots, Optimization Letters 1, 2007.

Berkhin, P. Survey of Clustering Data Mining Techniques. Accrue Software, 2002.

Dias, C. R.; & Ochi, L. S.. Efficient Evolutionary Algorithms for the Clustering Problems in Directed Graphs. Proc. of the IEEE Congress on Evolutionary Computation (IEEE-CEC), 983-988. Canberra, Austrália, 2003.

Dias, C. R. Algoritmos Evolutivos para o Problema de Clusterização de Grafos Orientados: desenvolvimento e análise experimental. 2004. 129 f. Dissertação de Mestrado em Computação, Universidade Federal Fluminense, Niterói, 2004.

Doval, D., Mancoridis, S. and Mitchell, B. S. Automatic Clustering of Software Systems using a Genetic Algorithm. Proc. of the Int. Conf. on Software Tools and Engineering Practice, pp. 73-81, 1999.

Freitas F. G.; Maia, C. L. B; Coutinho, D. P.; Campos, G. A. L.; Souza, J. T., Aplicação de Metaheurísticas em Problemas da Engenharia de Software: Revisão de Literatura, Anais do II Congresso Tecnológico InfoBrasil (InfoBrasil´2009), 2009.

Glover, F. Tabu search and adaptive memory programing: advances, applications and challenges. Interfaces in Computer Science and Operations Research, pp. 1-75, 1996.

Glover, F.; Laguna, M.; Mart, R. Fundamentals of scatter search and pathrelinking. Control Cybernetics, pp. 653-684, 2000.

Glover, F.; Kochenberger, G. A. Handbook of Metaheuristics. Kluwer Academic Publishers, 2003.

Goldberg, D. E. Genetic Algorithms in search, optimization and machine learning. Tuscaloosa: Addison-Wesley, 1989.

Han, J., e Kamber, M., Data Mining: Concepts and Techniques, 2 ed., Morgan Kaufmann, USA, 2005.

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

Santos, H. G., Ochi, L. S., Marinho, E. H., Drummond, L. M. A. Combining an Evolutionary Algorithm with Data Mining to solve a Vehicle Routing Problem. NEUROCOMPUTING Journal - ELSEVIER, volume 70 (1-3), pp. 70-77, 2006.

Semaan, G. S., Ochi, L.S., Brito, J. A. M. Um Algoritmo Evolutivo Híbrido Aplicado ao Problema de Clusterização em Grafos com Restrições de Capacidade e Conexidade. IX Congresso Brasileiro de Redes Neurais /Inteligência Computacional (IX CBRN), Ouro Preto, 2009.