HEURÍSTICA BASEADA EM BUSCA LOCAL ITERADA PARA A RESOLUÇÃO DO PROBLEMA DE AGRUPAMENTO DE SISTEMAS ORIENTADOS A OBJETOS

Gustavo Silva Semaan
Instituto de Computação – Universidade Federal Fluminense (IC-UFF)
Rua Passo da Pátria 156 - Bloco E – 3° andar, São Domingos, CEP: 24210-240, Niterói, RJ
gsemaan@ic.uff.br

André Luis Vasconcellos Botelho
Universidade Federal de Juiz de Fora – (UFJF)
Rua José Lourenço Kelmer, s/n, Bairro São Pedro, CEP: 36036-900, Juiz de Fora, MG
alvbotelho@gmail.com

Luiz Satoru Ochi
Instituto de Computação – Universidade Federal Fluminense (IC-UFF)
Rua Passo da Pátria 156 - Bloco E – 3° andar, São Domingos, CEP: 24210-240, Niterói, RJ
satoru@ic.uff.br

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 uma heurística baseada na metaheurística Busca Local Iterada. O algoritmo proposto utiliza três tipos de perturbações e uma busca local. Os resultados computacionais mostraram que a utilização do algoritmo proposto é uma alternativa eficiente para a resolução deste problema de clusterização, em que a heurística alcança soluções de qualidade em um reduzido tempo de execução.

PALAVRAS CHAVE: busca local iterada, metaheurísticas, engenharia de software, problemas de agrupamento automático

ABSTRACT

Several problems of Software Engineering presenting high complexity and combinatorial possibilities and can be modeled as optimization problems. This paper proposes a heuristic algorithm based on iterated local search metaheuristic to solve a problem of clustering object-oriented information systems. The algorithm uses three versions of perturbation procedures and one local search. The computational results showed that the use of this algorithm is an efficient alternative to solve this clustering problem, in which the heuristic achieve quality solutions in a reduced runtime.

KEYWORDS: iterated local search, metaheuristics, software engineering, automatic clustering problems

1. Introdução

Conforme Berkhin (2002), um problema de clusterização, tem como meta agrupar elementos de uma base dados em subconjuntos disjuntos (*clusters*) buscando a maximização da similaridade dos elementos de um mesmo *cluster* e a minimização da similaridade entre os elementos de *clusters* distintos. Esse problema é definido à partir de um conjunto com *n* elementos $X = \{x_1,...,x_n\}$, buscando encontrar partições desse conjunto em *clusters Ci* respeitando as seguintes condições:

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

Os problemas de clusterização podem ser classificados em duas formas: Na primeira, a quantidade ideal de *clusters* é conhecida préviamente, caracterizando um Problema de *K-Clusterização* ou apenas *Problema de Clusterização* (PC). Na segunda a quantidade ideal de *clusters* não faz parte dos dados de entrada do problema, exigindo que o algoritmo descubra este número, esta versão denomina-se *Problema de Clusterização Automática (PCA)* (Berkhin, P., 2002; Dias and Ochi, 2003; Cruz and Ochi, 2009; Dib and Ochi, 2011). Ambos os problemas pertencem a classe NP-Completo, o que de certa forma limita o uso exclusivo de métodos exatos para a sua resolução. A quantidade de soluções viáveis em um PC pode ser obtida pela Fórmula 1, em que *n* representa a quantidade de elementos e *k* a quantidade de *clusters*, enquanto para no PCA pode ser obtida pela Fórmula 2.

A fim de ilustrar o crescimento da quantidade de soluções possíveis no PC e PCA, Dias (2004) exemplifica que no caso do PC para um conjunto X com 10 elementos distribuídos em 2 *clusters* existem 511 soluções diferentes. Para 100 elementos em 2 *clusters*, este número cresce para 6,33 . 10^{29}. Para 100 elementos em 5 *clusters* encontra-se 6,57 . 10^{67}. Já no caso de 100 elementos em 2 *clusters*, pode-se encontrar 5,35 . 10^{300} soluções diferentes. Na utilização da fórmula para o PCA, esse crescimento é ainda mais rápido como pode ser visto na Fórmula 2.

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

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 Automática

Diversos Problemas da Engenharia de *Software* (ES), por possuírem elevada complexidade e possibilidades combinatórias, podem ser modelados como problemas de otimização (PO). As resoluções desses problemas sugerem algoritmos metaheurísticos eficientes, uma vez que métodos exaustivos e/ou exatos podem ser inviáveis nestas situações. 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 é insuficiente (Doval *et al.* 1999). Neste contexto, como um estudo de caso, o presente trabalho propõe um algoritmo heurístico baseado na metaheurística Busca Local Iterada (*Iterated Local Search* - ILS) para a resolução de um problema de agrupamento de sistemas de informação orientados a objetos. Tal problema consiste na Modularização do *Software* com base em sua arquitetura. A motivação para a escolha desta técnica está baseada na experiência prévia dos autores deste trabalho na resolução de diferentes problemas de otimização combinatória (Brito et. al. 2010; Silva *et at*. 2010; Souza *et al*. 2010; Subramanian *et al*. 2010; Gonçalves *et al*. 2011; Mine *et.al*. 2011; Ribas *et al*. 2011; Silva and Ochi 2011).

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.

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) e Semaan e Ochi (2007) foram apresentados algoritmos para modularizar sistemas de informação desenvolvidos utilizando o paradigma estruturado. Em Botelho *et al.* (2011) foi apresentada uma heurística evolutiva para a resolução desse problema considerando a mesma modelagem utilizada no presente trabalho.

O objetivo do problema abordado é propor a divisão automática de sistemas de informação considerando os relacionamentos existentes entre suas unidades. Os grupos formados na divisão serão distribuídos entre equipes de desenvolvimento, descentralizando as atividades, reduzindo a comunicação entre equipes e aumentando a comunicação entre os desenvolvedores em uma mesma equipe.

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 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 é mais forte, possuindo pesos maiores que associações simples. A Figura 1 apresenta um diagrama de classes e seu grafo correspondente. As arestas que representam relacionamentos mais fortes estão em destaque, com linhas mais largas.

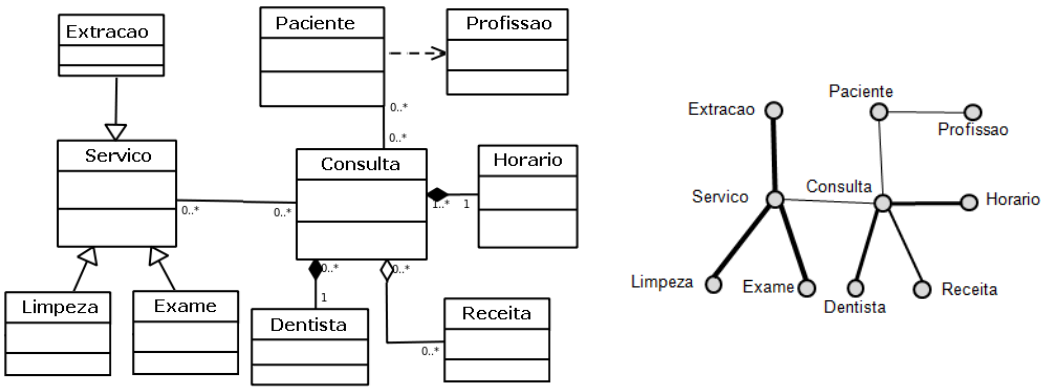

Figura 1: Um exemplo de diagrama de classes e seu grafo correspondente.

3. Algoritmo Proposto

De acordo com Lourenço et al. (2003), o ILS é um método de busca local que não visa especificar a busca no espaço de soluções completo, e sim em um pequeno subespaço, definido por soluções ótimas locais de determinados procedimentos de otimização. A idéia básica é que com a perturbação de uma solução ótima local e a realização de novas buscas locais é possível encontrar soluções de melhor qualidade. A Figura 2 apresenta o pseudo-código do algoritmo ILS na sua forma tradicional.

```
Procedimento ILS()
    S0 ← GerarSoluçãoinicial();
    S ← BuscaLocal (S0);
    Enquanto não(critério de parada)faça
        S' ← Perturbação (S);
        S" ← BuscaLocal (S0);
        S  ← CritérioAceitação(S, S");
    Fim-Enquanto;
    Retorne S;
Fim ILS();
```

Figura 2: Algoritmo ILS Tradicional

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 uma 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 e o valor alocado representa o *cluster* a que pertence o vértice (Berkhin, 2002). Para a construção de soluções iniciais foi utilizada uma abordagem aleatória.

3.2. Heurísticas de Refinamento

Também conhecidas na literatura como técnicas de Busca Local, têm como finalidade melhorar a qualidade das soluções. Conforme Glover *et al*. (2003), Semaan (2009), Dias (2003), Dias (2004) e Santos *et al*. (2006), Gonçalves *et al.* (2011), Silva *et al*. (2011), as buscas locais quando bem utilizadas tendem a melhorar consideravelmente a qualidade das soluções de um algoritmo heurístico,

A busca local utilizada neste trabalho foi apresentada por Botelho *et. al.* (2011) e atua na migração de vértices entre *clusters*, considerando uma *força de conexão.* Basicamente, para cada vértice do grafo, verifica-se a intensidade de conexão com todos os *clusters* da solução. Essa intensidade é obtida com o somatório dos pesos das arestas entre o vértice selecionado e seus vértices adjacentes, pertencente a um outro *cluster* específico. A Figura 3 ilustra uma solução que possui 3 *clusters* e 11 vértices. Com base nessa figura, é possível observar que:

- A intensidade da conexão do vértice *f* em relação ao *cluster* 1 é 4, uma vez que ele está conectado apenas ao vértice *g* alocado ao *cluster 1* e a aresta possui peso 4. Em relação ao *cluster 3* esse vértice possui intensidade 2, pois está conectado aos vértices *e* e *d*, ambos com arestas de pesos 1. Já sua intensidade em relação ao cluster 2 é 0, uma vez que não existe conexão entre ele e nenhum vértice do *cluster 2*. A busca local, nesse cenário, não migra o vértice *f*, pois ele já está associado ao *cluster* que possui a maior intensidade.
- Seguindo a mesma idéia, a intensidade da conexão do vértice *d* em relação ao *cluster* 1 é 1 e, em relação aos demais *clusters* é 0. Neste caso, o algoritmo migra o vértice *d* para o *cluster 1*.

- O vértice *a* possui intensidade 0, 2 e 3 em relação aos *clusters* 1,2 e 3, respectivamente. Esse vértice, então, é migrado para o *cluster* 3. É importante observar que a quantidade de arestas conectadas a esse vértice no *cluster 2* é maior, porém a intensidade de conexão é menor que a existente para o *cluster 3*.

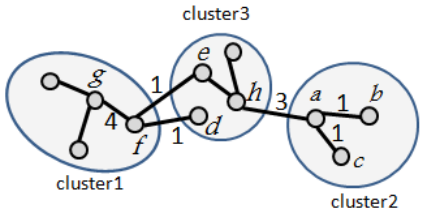

Figura 3: Visualização de uma solução.

A Figura 4 apresenta a solução da Figura após a execução da busca local.

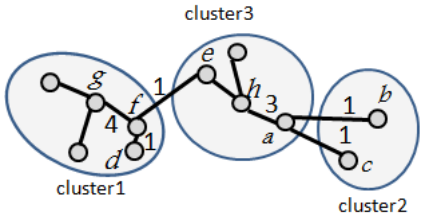

Figura 4: Visualização da solução após a aplicação da busca local.

A busca local é executada até que a solução esteja estável, ou seja, todos os vértices estão alocados nos *clusters* que possuem maior intensidade.

3.3. Perturbações

O módulo de perturbação ou diversificação é responsável por gerar uma nova solução, modificando uma solução corrente. Ela deve ser suficientemente forte para permitir que a busca local explore novas regiões promissoras, mas também não tão drástica para não se tornar um processo de um reinício aleatório. Nesse trabalho foram implementados três procedimentos de perturbações que são ativadas de forma aleatória a cada iteração da heurística proposta.

- ***Migração aleatória:*** foi apresentada por Botelho et. al. (2011) como o operador de mutação do algoritmo evolutivo. Cada vértice possui uma probabilidade de migrar para outro cluster. O destino pode ser qualquer cluster da solução, ou ele pode formar sozinho um novo cluster.
- ***Explosão:*** um dos clusters da solução é selecionado de forma aleatória e cada vértice pertencente a ele formará sozinho um novo cluster.
- ***Divisão:*** um dos clusters da solução é selecionado de forma aleatória e é dividido em dois clusters. O critério de seleção dos vértices que serão alocados em cada cluster também é aleatório.

3.4. 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. Porém, assim como em Botelho et. al. (2011), houve 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 (alta densidade de conexões entre elementos de um mesmo *cluster*) e, ao

mesmo tempo, penaliza a inter-conectividade (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 intra-conectividade de um *cluster i*, em que μ_i 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 inter-conectividade mensura a conectividade entre vértices de *clusters* distintos. A Fórmula 4 é utilizada para calcular a inter-conectividade entre dois *clusters*, em que *Ni* é o total de vértices do *cluster i*, *Nj* é o total de vértices do *cluster j* e e_{ij} o total de arcos que possuem as extremidades em *i* e *j*.

$$A_i = \frac{\mu_i}{N_i^2} \qquad B_{i,j} = \begin{cases} 0 \text{ se i=j} \\ \dfrac{\varepsilon_{ij}}{2N_iN_j} \text{ se i} \neq \text{j} \end{cases} \qquad MQ = \begin{cases} \dfrac{\sum_{i=1}^{k} A_i}{k} - \dfrac{\sum_{i,j=1}^{k} B_{i,j}}{\frac{k(k-1)}{2}} \text{ para k>1} \\ Ai \text{ para k=1} \end{cases}$$

Fórmula 3: Intraconectividade.

Fórmula 4.: Interconectividade.

Fórmula 5: Qualidade da modularização.

4. Experimentos

Os algoritmos propostos nesse trabalho foram implementados em linguagem JAVA e o computador utilizado nos experimentos possui processador Core 2 Quad 64 bits, 2,5 GHz, com 4 Gb de memória RAM e sistema operacional Ubuntu 10.10.

Diversos experimentos foram realizados utilizando dados reais, por meio de códigos-fonte de sistemas de informação mapeados, e artificiais, com a utilização de instâncias existentes na literatura (Botelho *et. al.* ,2011, Semaan and Ochi, 2007) ou adaptadas ao modelo de grafos ponderados (Dias and Ochi (2003)). Tais instâncias encontram-se disponíveis em ***http://labic.ic.uff.br***.

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

O primeiro experimento consiste na execução do algoritmo proposto em cada uma das 10 instâncias 10 vezes considerando o critério de parada por tempo máximo de execução de 10 minutos.

A Tabela 2 apresenta os resultados obtidos no primeiro experimento, considerando a função de aptidão, tempo de execução e *gap* (Fórmula 6) em relação ao melhor valor de aptidão (*f(x)Best*) encontrado pelos algoritmos evolutivos em Botelho *et al.* (2011).

Com base nessa tabela, é possível observar que todos os resultados foram melhorados em ao menos 0,06%. A média da melhoria obtida pelo nosso trabalho quando confrontado com os melhores resultados da literatura foi em média 5,6%. Além disso, a média das médias dos resultados por instância da heurística proposta é menor que 0,5%, enquanto para o algoritmo evolutivo da literatura é superior a 9,8%.

Em relação ao tempo de execução, embora a heurística proposta não tenha para todas as instâncias o tempo médio menor que os obtidos pelo algoritmo evolutivo da literatura, a média dos tempos médios é de apenas 20,3 segundos, contra 80,7 segundos. Além disso, é importante destacar que os tempos do algoritmo evolutivo referem-se a obtenção de soluções de pior

qualidade, uma vez que a heurística proposta neste trabalho melhorou os resultados para todas as instâncias.

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

Fórmula 6: *Gap*.

Tabela 2: Resultados colhidos no primeiro experimento.

		Algoritmo Evolutivo			ILS Proposto		
		Gap %		Tempo	*Gap %*		Tempo
Instância	F(x)	Menor	Médio	(seg)	Menor	Médio	(seg)
1	**0,674**	22,16	22,16	1	0,00	0,00	0
2	**0,614**	2,60	7,90	1	0,00	0,00	3
3	**0,543**	5,76	5,76	8	0,00	0,33	79
4	**0,811**	5,44	9,68	34	0,00	1,13	5
5	**0,679**	2,00	2,64	120	0,00	0,92	64
6	**0,649**	0,06	8,34	227	0,00	0,00	2
7	**0,630**	2,86	3,74	191	0,00	0,48	37
8	**0,566**	2,95	13,96	223	0,00	1,50	12
9	**0,281**	9,10	9,10	1	0,00	0,00	0
10	**0,349**	3,10	15,26	1	0,00	0,00	1
	Média	5,60	9,86	80,7	0,00	0,44	20,3

Para analisar de forma mais detalhada a robustez do ILS proposto nesse trabalho e do melhor algoritmo evolutivo proposto por *Botelho et. al.* (2011), foram realizados experimentos de análise de probabilidade empírica, em que cada algoritmo foi executado 200 vezes em duas das maiores instâncias, considerando o critério de parada um tempo máximo de 3000 segundos.

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 em função do tempo de execução.

A Figura 5 apresenta um comparativo do melhor algoritmo evolutivo da literatura e a heurística proposta para o alcance a alvos médios (média dos resultados da literatura) e difíceis (melhor resultado obtido na literatura) para a instância 7. É possível observar a superioridade da heurística proposta em que, em menos de um segundo, o algoritmo alcançou o alvo médio em todas as execuções, o que somente ocorreu ao algoritmo evolutivo com cerca de 1700 segundos de execução. Em relação ao alvo difícil, enquanto o algoritmo evolutivo em cerca de 3000 segundos teve pouco mais de 80% de convergência ao alvo estipulado, a heurística ILS em cerca de 100 segundos obteve sucesso em todas as execuções. Resultados semelhantes ocorreram à instância 8, conforme ilustra a Figura 6.

A Figura 7 ilustra um resultado encontrado pelo ILS para a instância 9. É possível observar que o algoritmo respeitou os relacionamentos mais fortes entre as entidades do software em questão. Ele reuniu em um mesmo cluster as classes que possuíam agregação e herança, separando apenas as classes que possuíam um relacionamento simples.

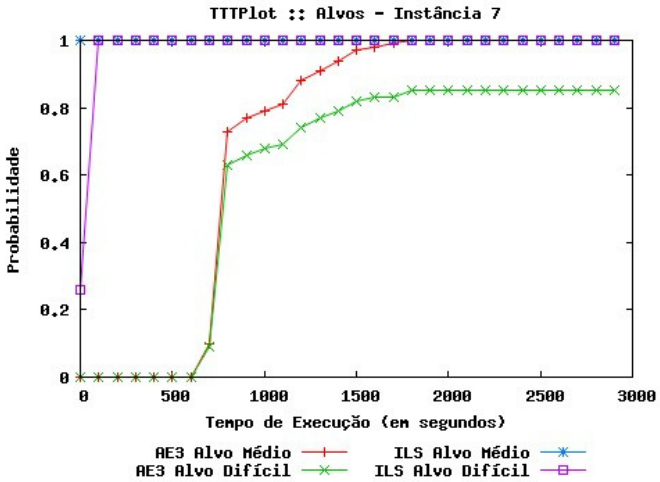

Figura 5: Resultados por tempo de execução para a instância 7.

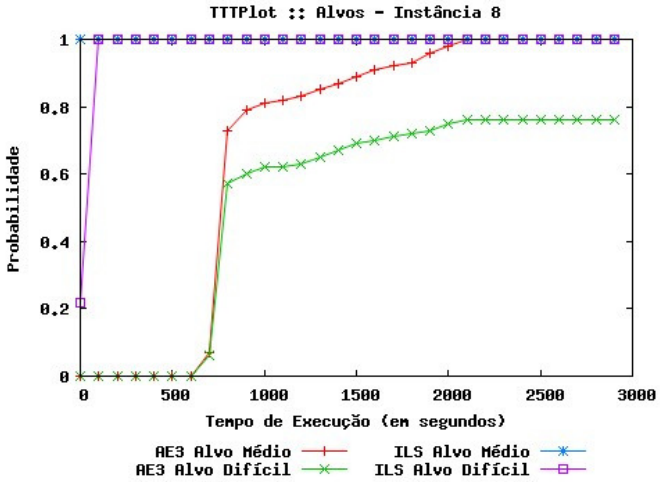

Figura 6: Resultados por tempo de execução para a instância 8.

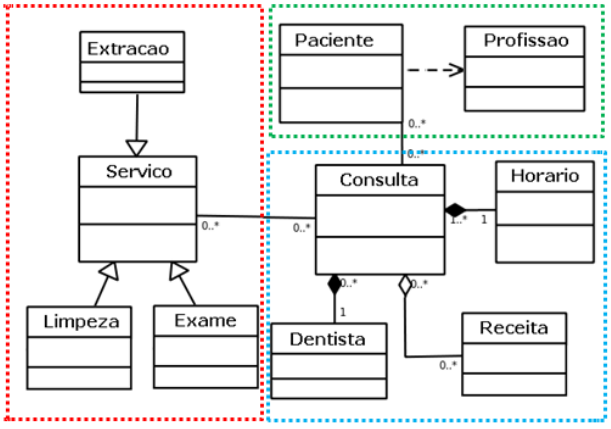

Figura 7: Um resultado para a instância 9.

5. Conclusões

Os resultados computacionais mesmo de forma empírica, mostraram que a utilização do algoritmo proposto é uma alternativa eficiente para a resolução do problema de clusterização automática, onde a heurística ILS proposta alcançou resultados superiores e exigiu um reduzido tempo de execução.

No contexto da qualidade das soluções obtidas, a heurística proposta melhorou todas as soluções conhecidas, melhorando os resultados existentes entre 0,06% e 22%. Com isso, a média dos *gaps* dos melhores resultados da literatura foi de 5,6% em relação aos nossos resultados. Já em relação ao tempo de execução, a média dos tempos médios do ILS foi de apenas 20,3 segundos, contra 80,7 segundos do melhor algoritmo evolutivo da literatura, além do ILS nestes tempos alcançar as soluções de melhor qualidade.

Durante as pesquisas e na realização dos experimentos foram identificadas novas oportunidades para trabalhos futuros, incluindo:

- Adaptar o ILS proposto ao problema de *K-Clusterização*.
- Adaptar o ILS proposto ao problema de *K-Clusterização* Capacitado, ou seja, além da quantidade de grupos conhecida préviamente, existe como dados de entrada, uma capacidade mínima/máxima nos clusters.
- Gerar versões paralelas do ILS proposto em ambientes *multi-core* e/ou programação em GPU (Cuda).

Agradecimentos

A todos os professores e alunos do Instituto de Computação da Universidade Federal Fluminense, da Universidade Federal de Juiz de Fora e à CAPES (*www.**capes**.gov.br*) pelo apoio financeiro.

Referências

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

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

Botelho, A. L. V. E Semaan, G. S., **Ochi, L. S**. (2011). Agrupamentos de Sistemas Orientados a Objetos com Metaheurísticas Evolutivas, A ser publicado nos *Anais do Simpósio Brasileiro de Sistemas de Informação*, Salvador - BA.

Brito, J. André M., Ochi, Luiz Satoru, Montenegro, Flávio, and Maculan, Nelson. (2010). An ILS Approach Applied to the Optimal Stratification Problem. In International Transaction in Operational Research (ITOR), Volume 17, pp. 753-764 - Wiley-Blackwell.

Cruz, M. D. and Ochi, L. S. (2009). A hybrid method using evolutionary algorithm and a linear integer model to solve the automatic clustering problem. Proc. of the ICEC 2009 - International Conference on Evolutionary Computation (CD-ROM), Portugal.

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

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

Dib, C. M. e Ochi, L. S. (2011). Um algoritmo evolutivo com memória adaptativa para o problema de clusterização automática. *Learning & NonLinear Models*, Volume 8(4), pp. 227-239

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

Freitas F. G., Maia, C. L. B., Coutinho, D. P., Campos, G. A. L. e Souza, J. T. (2009).

Aplicação de Metaheurísticas em Problemas da Engenharia de Software: Revisão de Literatura, Anais do II Congresso Tecnológico InfoBrasil (InfoBrasil´2009).

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

Gonçalves, Luciana B., Martins, Simone de L,, Subramanina, Anand and Ochi, Luiz Satoru (2011). Exact and Heuristic Approach for the set cover with pairs problem. To appear in Optimization Letters. *(Já disponivel em versão online).*

Lourenço, H. R.; Martin, O. e Stuetzle, T. *Iterated local search. Handbook of Metaheuristics,* p. 321-353, MA, Kluwer Academic Publishers, 2003.

Mine, M., Silva, M. S. A., Subramanian, A., Ochi, L. S. and Souza, M. J. F. (2011). *A hybrid heuristic based on iterated local search and genius, for the vehicle routing problem with simultaneous pickup and delivery. To appear in International Journal of Logistics Systems Management* (IJLSM) – *Inderscience Publishers.*

Ribas, S., Subramanian, A., Coelho, I. M. C., Ochi, L. S. and Souza, M. J. F. A (2011). A hybrid algorithm for the vehicle routing problem with time-windows. Proc. of the International Conference on Industrial Engineering and Systems Management - IESM'2011 - May, 25-27, Metz, France.

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

Semaan, G. S. e Ochi, L. S. (2007). Algoritmo Evolutivo para o Problema de Clusterização em Grafos Orientados. Anais do X Simpósio de Pesquisa Operacional e Logística da Marinha *(CD-ROM)*, Rio de Janeiro, Brasil. X SPOLM.

Semaan, G. S., Ochi, L.S. e Brito, J. A. M. (2009).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 (CD-ROM), Ouro Preto.

Silva, G. C., Bahiense, L, Ochi, L. S., and Boaventura-Netto, P. O. (2011). Dynamic Space Allocation Problem: Applying Hybrid GRASP and Tabu Search metaheuristics. To appear in Computers & Operations Research - *ELSEVIER*.

Silva, André R. V., and Ochi, Luiz Satoru. (2011). The dynamic resource constrained project scheduling problem. To appear in International Journal of Data Mining, Modelling and Management (IJDMMM) - InderScience Publishers.

Subramanian, A., Drummond, L. M. A., Ochi, L. S., Bentes, C., and Farias, Ricardo. (2010). A Parallel Hetaheuristic for the Vehicle Routing Problem with Simultaneous Pickup and Delivery. Computers & Operations Research – ELSEVIER , Volume 37(11), p. 1899-1911.