

Melhorando o Desempenho de Algoritmos Evolutivos por Meio de Mineração de Dados: uma Aplicação na Área de Petróleo¹

L.S. OCHI, H.G. SANTOS, L.H.C. MERSCHMANN², Instituto de Computação, Universidade Federal Fluminense, 24210-240 Niterói, RJ, Brasil.

Resumo. A solução de problemas das classes NP-Completo e NP-*Hard* tem sido um grande desafio para pesquisadores da área de Otimização Combinatória. Um dos caminhos promissores tem sido reunir conceitos das áreas de Otimização e Inteligência Artificial (IA), procurando conjugar a rigidez dos métodos exatos de otimização com a flexibilidade dos métodos de busca em IA para desenvolver o que chamamos de técnicas inteligentemente flexíveis. As metaheurísticas são resultados desta união e podem ser vistas como heurísticas genéricas que procuram reduzir limitações históricas encontradas em heurísticas tradicionais, como por exemplo, a dificuldade em escapar de ótimos locais muitas vezes ainda distantes de um ótimo global. Neste trabalho são apresentadas alternativas para melhorar o desempenho de Algoritmos Evolutivos no que se refere à qualidade das soluções geradas para o Problema de Roteamento de Unidades Móveis de Pistoneio. São propostos módulos de busca local e mineração de dados para serem incorporados a um Algoritmo Genético (AG). Os resultados demonstraram uma melhora significativa com relação ao AG que continha somente os operadores genéticos.

1. Introdução

A bacia Potiguar, situada entre os estados do Rio Grande do Norte e do Ceará, é a maior produtora de petróleo em terra de nosso país, contendo cerca de 4000 poços de gás e petróleo [3]. Como nos campos petrolíferos desta região a maior parte dos poços são não surgentes, ou seja, não têm capacidade própria de elevação dos fluidos (óleo e água) até a superfície, faz-se necessário utilizar um método auxiliar artificial de elevação dos fluidos. Dentre esses métodos, os mais utilizados na região da bacia Potiguar são: bombeio mecânico (cavalo de pau), bombeio por cavidades progressivas (BCP) e bombeio por Unidade Móvel de Pistoneio (UMP). Os métodos de bombeio mecânico e bombeio por cavidades progressivas são operacionalizados com equipamentos permanentemente acoplados aos poços. Sendo assim, a utilização de tais métodos justifica-se somente quando a vazão do poço é tal que viabiliza financeiramente a operação. Para poços onde a vazão não justifica a utilização de

¹Conferência apresentada no XXVII CNMAC.

²{satoru,hsantos,lmerschman}@ic.uff.br

equipamentos permanentes, a extração dos fluidos por meio de uma UMP torna-se uma alternativa para mantê-los produzindo. Uma UMP é um equipamento acoplado a um caminhão ou trator, que percorre os poços extraindo os fluidos dos mesmos.

Todos os campos de produção de petróleo possuem uma unidade responsável pela separação do óleo da água. Esta unidade é chamada de Estação de Tratamento de Óleo (ETO) e corresponde ao ponto de partida e chegada da UMP a cada dia de trabalho. Além disso, todos os poços estão interconectados por estradas, ou seja, não existem poços isolados. Sendo assim, a UMP parte da ETO, percorre alguns poços extraindo os fluidos e retorna à ETO no final de sua jornada.

Para ilustrar o comportamento de um Algoritmo Evolutivo (AE) incluindo uma técnica de Mineração de Dados (MD), será considerado o problema de otimização denominado Problema de Roteamento de Unidades Móveis de Pistoneio (PRUMP). Este problema corresponde a encontrar uma rota para uma UMP que maximize o volume de óleo coletado e atenda a restrição de tempo, uma vez que as jornadas de trabalho diárias possuem uma duração máxima pré-determinada.

O problema citado anteriormente pode ser considerado como uma variação do Problema do Caixeiro Viajante (PCV), onde um único veículo deve percorrer um ciclo sobre uma rede de n clientes (poços) maximizando algum objetivo (volume de óleo coletado). Assim como o PCV, o PRUMP é classificado como NP-*Hard*. Deste modo, a utilização de métodos exatos para solucionar este problema torna-se limitada, justificando-se o emprego de algoritmos aproximados ou heurísticas como, por exemplo, Algoritmos Evolutivos.

O problema de roteamento de uma UMP pode ser modelado por um grafo não direcionado $G = (X, U)$, onde os vértices $x_i \in X$ correspondem aos poços e cada aresta $(i, j) \in U$ representa uma estrada interligando os poços i e j . A cada vértice $x_i \in X$ existe um valor v_i associado que corresponde à vazão daquele poço. Além disso, as arestas são valoradas para representar o tempo de percurso entre dois poços. A Figura 1 mostra uma rede com 12 poços e uma rota que percorre 7 poços coletando um volume de 206 unidades de medida e respeitando a restrição de 40 unidades de tempo.

2. Propostas para Melhorar o Desempenho dos Algoritmos Evolutivos

O objetivo deste trabalho é introduzir módulos aos AEs tradicionais para melhorar a qualidade da solução de problemas de elevada complexidade computacional. Os AEs e, em particular, seu representante mais popular, os Algoritmos Genéticos (AGs) [9, 8], estão inseridos numa área de pesquisa da Inteligência Artificial inspirada na Teoria da Evolução Natural e na Genética, conhecida como Computação Evolutiva ou Evolucionária. Tais algoritmos tentam simular algumas etapas do processo Darwiniano de seleção natural e têm sido muito utilizados para solucionar problemas considerados intratáveis (NP-Completo e NP-*Hard*) em diversas áreas, embora na sua forma tradicional, possam não demonstrar muita eficiência na resolução de alguns problemas de otimização combinatória. Na tentativa de melhorar o desempenho dos GAs, pesquisadores têm proposto variantes, tais como: Algoritmos

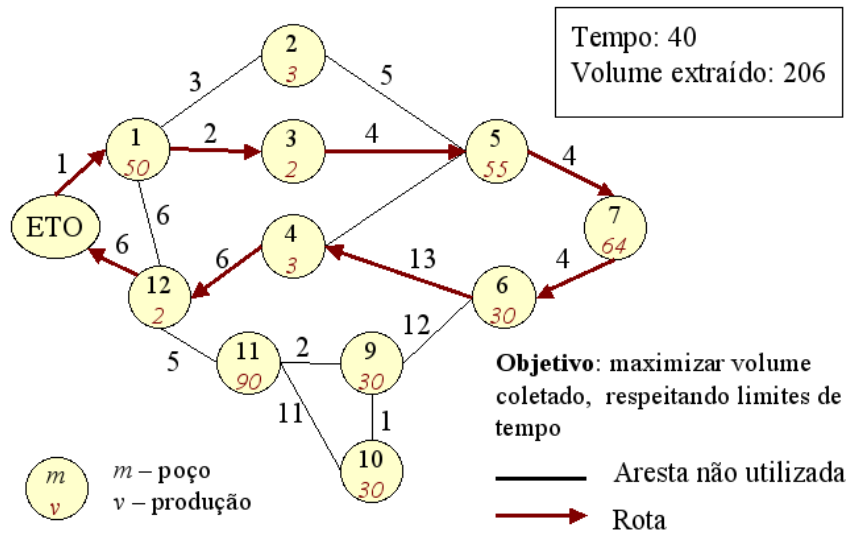


Figura 1: Exemplo de uma rota numa rede contendo 12 poços.

Meméticos [10], *Scatter Search* [7] e *Population Heuristics* [4].

O AG é um procedimento heurístico iterativo onde a cada iteração, por meio de operadores genéticos, novas populações de indivíduos são criadas. Usualmente, nestes operadores estão envolvidos, cruzamento e mutação, por meio dos quais o espaço de busca é explorado permitindo encontrar boas soluções para o problema em questão.

Neste trabalho são apresentadas propostas para melhorar o desempenho dos AGs para o PRUMP. Primeiramente construiu-se um AG que possui a fase de geração de novos indivíduos mais elaborada do que uma que utiliza operadores convencionais de cruzamento e mutação. Posteriormente, na tentativa de melhorar as soluções obtidas por meio do AG, dois módulos foram implementados. O primeiro corresponde a uma Busca Local e o segundo é um módulo de Mineração de Dados. Estes dois módulos têm como objetivo intensificar a busca em regiões do espaço de soluções que se mostraram promissoras durante a execução do AG. Apesar dos algoritmos serem desenvolvidos para o PRUMP, eles podem ser facilmente adaptados para resolver outros problemas de otimização combinatória.

2.1. Algoritmo Genético

Esta seção apresenta o AG implementado. Cada indivíduo da população é representado por um vetor de números inteiros, onde cada gene corresponde a um poço pertencente à rota que está codificada neste indivíduo. A última posição (gene) deste vetor sempre possui o valor 0 (ETO), que indica o final da rota (bem como a origem). Seja o indivíduo mostrado na Figura 2. Neste caso, a rota representada por esse indivíduo é a seguinte: ETO - poço 16 - poço 34 - poço 21 - poço 17 - poço

76 - poço 3 - poço 58 - poço 44 - ETO.

16	34	21	17	76	3	58	44	0
----	----	----	----	----	---	----	----	---

Figura 2: Exemplo de um indivíduo da população.

Na fase de construção da solução inicial os indivíduos são gerados utilizando-se um procedimento iterativo, guloso e randomizado, nos moldes do procedimento construtivo da metaheurística GRASP (*Greedy Randomized Adaptive Search Procedures*) [12]. A cada iteração utiliza-se um critério guloso para definir qual poço irá fazer parte da solução que está sendo construída. Este critério toma como base a razão $v_i/t_{j,i}$, onde v_i é a vazão do poço candidato i a entrar na solução e $t_{j,i}$ tempo de percurso gasto para se deslocar do poço candidato i até o último poço j inserido na solução parcial. Sendo assim, a cada iteração, escolhe-se aleatoriamente um poço que está contido numa lista de poços candidatos. Esta lista contém todos os poços que ainda não fazem parte da solução e que possuem a razão $v_i/t_{j,i}$ contida no intervalo $[r_{max}, r_{min}]$. Sendo \bar{r} o valor da maior razão $v_i/t_{j,i}$ dentre todos os poços que ainda não fazem parte da solução e \underline{r} o valor da menor razão dentre esses mesmos poços, $r_{max} = \bar{r}$ e $r_{min} = \bar{r} - (\bar{r} - \underline{r}) \cdot \alpha$. O grau de aleatoriedade na geração do indivíduo poderá ser definido pelo parâmetro α , que pode ser escolhido no intervalo $[0, 1]$. Este processo é executado enquanto o tempo total da rota que está sendo construída for menor ou igual ao tempo limite (restrição do PRUMP). A fase de geração de novos indivíduos do AG proposto neste trabalho utiliza um operador genético diferente dos operadores de cruzamento e mutação tradicionais. O processo de cruzamento para geração de um novo indivíduo começa com a seleção de $nPais$ indivíduos pais, onde $nPais$ pode variar de um até o tamanho da população. A escolha de cada indivíduo pai é realizada por meio de um torneio que consiste em selecionar o indivíduo (cromossomo) mais apto a partir de um conjunto de k indivíduos selecionados aleatoriamente na população (nos experimentos, utilizou-se torneio binário: $k = 2$). O operador de cruzamento proposto neste trabalho combina informações provenientes dos indivíduos pais e o critério heurístico utilizado na fase de construção da população inicial, como explicado a seguir. A cada iteração do processo de construção do indivíduo filho, um poço é selecionado para fazer parte da rota. Este poço é escolhido de forma aleatória numa lista de poços candidatos, de acordo com uma distribuição de probabilidade que privilegia os poços que possuem maior $\gamma_i = (1 + \omega_{j,i}) \cdot (v_i/t_{j,i})$, onde $\omega_{j,i}$ é a frequência com que o poço candidato i é o sucessor do poço j (último poço adicionado à solução parcial) nos indivíduos pais. A probabilidade de escolha de um determinado poço é proporcional a posição desse poço em uma lista ordenada de forma não crescente segundo o critério γ_i , conforme sugerido em [5]. Neste trabalho optou-se pela distribuição de probabilidade polinomial. O processo de construção do indivíduo filho continua enquanto o tempo total da rota que está sendo construída for menor ou igual ao tempo limite estabelecido no problema. É importante notar que o operador de cruzamento utilizado não trabalha somente com informações contidas nos indivíduos pais, permitindo a ocorrência de cromossomos filhos com código genético não existente nos pais. Sendo assim, não houve necessidade de se incorporar um

operador de mutação. Durante o processo de evolução a população permanece com tamanho fixo, ou seja, a inserção de β novos indivíduos ocasiona a saída dos β piores indivíduos da população.

2.2. Algoritmo Genético com Busca Local

O módulo de Busca Local corresponde a um procedimento que permite uma busca sistemática no espaço de soluções por meio de diferentes estruturas de vizinhança.

No caso deste trabalho, duas estruturas de vizinhança foram propostas com o objetivo de aumentar a quantidade de óleo coletada a partir de uma solução viável do problema. O pseudocódigo do Procedimento 1 demonstra como as estruturas de vizinhança são sistematicamente aplicadas. O processo começa pela primeira estrutura de vizinhança ($k = 1$) e sempre que um movimento de melhora é encontrado, a busca é reiniciada na primeira estrutura. Quando movimentos de melhora não mais são possíveis, a próxima estrutura de vizinhança é explorada ou a busca termina (se $k = 2$). Em ambas vizinhanças, uma lista L_r dos poços que ainda não pertencem à rota codificada no indivíduo é criada. Na primeira estrutura tenta-se inserir cada poço pertencente à lista L_r entre cada par de poços vizinhos da rota codificada no indivíduo que está sendo utilizado para a realização da busca local. Já na segunda estrutura de vizinhança tenta-se trocar cada poço que já pertence à rota por outro com vazão maior que ainda não faz parte da solução.

Procedimento 1 Busca Local

- 1: **Entrada:** indivíduo
 - 2: Selecione as estruturas de vizinhança $N_k, k = \{1, 2\}$;
 - 3: $s \leftarrow$ indivíduo; $k \leftarrow 1$;
 - 4: **enquanto** $k \leq 2$ **faça**
 - 5: $s' \leftarrow$ *PrimeiroMovimentoMelhora*($N_k(s)$);
 - 6: **se** $f(s') > f(s)$ **então**
 - 7: $s \leftarrow s'$;
 - 8: $k \leftarrow 1$;
 - 9: **senão**
 - 10: $k \leftarrow k + 1$;
 - 11: **fim se**
 - 12: **fim enquanto**
 - 13: **retorne** s .
-

2.3. Algoritmo Genético com Mineração de Dados

Com o objetivo de acelerar a ocorrência de soluções de alta qualidade na população, propomos a incorporação de um módulo de mineração de dados no AG. Este módulo tem como objetivo a descoberta de padrões (subrotas) que são frequentemente encontradas nas melhores soluções da população. Essa abordagem difere significativamente das aplicações correntes que tratam de AGs e mineração de dados, uma vez que até agora, a maioria dos esforços tratam do desenvolvimento de AGs como

métodos de otimização para a solução de problemas de mineração de dados [6], como a descoberta de regras de associação, regras classificação e agrupamento, que não é o caso da nossa proposta.

O processo inicia com a criação de um conjunto elite (CE) de soluções. Esse subconjunto da população é formado pelas s melhores soluções encontradas, sendo atualizado sempre que uma solução melhor que a pior solução do CE e diferente de todas as soluções desse conjunto é gerada. O CE será a base de dados na qual se tentará descobrir padrões relevantes. Para isso, desenvolveu-se uma versão ligeiramente modificada do algoritmo Apriori [1], a qual descobre seqüências freqüentes (ao invés de regras de associação). O algoritmo recebe o suporte mínimo como parâmetro, isto é, a significância estatística mínima que uma seqüência deve apresentar para ser considerada freqüente. Então, o algoritmo irá descobrir todas as seqüências (trechos de rotas encontrados no CE), de todos os tamanhos, que satisfazem o suporte mínimo. Por exemplo, um suporte de 0,5 indica que ao menos a metade das soluções do CE precisam conter a seqüência considerada. Uma vez que um conjunto de subseqüências freqüentes estiver disponível, este é usado para guiar a construção de indivíduos, do seguinte modo: novos indivíduos são construídos utilizando um algoritmo construtivo, como definido na seção 2.1., exceto pelo fato de que sempre que um poço é selecionado para ser adicionado à solução parcial, um conjunto de trechos válidos de rotas é construído. Esse conjunto contém apenas subseqüências freqüentes cujos poços não aparecem na solução parcial, iniciando com o poço selecionado. Se mais de uma seqüência é encontrada, seleciona-se de modo aleatório entre estas. A seqüência selecionada é incorporada na solução parcial. Uma vez que o limite de tempo pode ser facilmente violado através da adição de uma seqüência de poços, um operador de reparação é aplicado para remover os últimos poços, se necessário. Se nenhuma seqüência válida é encontrada nos resultados da mineração, somente o poço selecionado é adicionado. Toda vez que o módulo de mineração de dados é disparado, β novos indivíduos são gerados utilizando a informação de mineração de dados. Assim como no operador de cruzamento, a população permanece com o mesmo tamanho. Esse processo é aplicado em intervalos de μ gerações.

3. Algoritmo Genético com Mineração de Dados e Busca Local

Os procedimentos de mineração de dados podem ser utilizados juntamente com procedimentos de busca local, através da aplicação de busca local em novos indivíduos provenientes tanto dos operadores genéticos quanto do procedimento de geração de novos indivíduos que utiliza informações de mineração de dados. Isso configura a última versão do AG proposto neste trabalho: o Algoritmo Genético com Busca Local e Mineração de Dados - AGBLMD. O pseudo-código para esse algoritmo é apresentado no Procedimento 2. A população inicial é gerada utilizando o procedimento construtivo guloso e randômico (*CGA*, linha 2), o qual foi descrito na seção 2.1..

A função prole (linha 6) indica a aplicação do operador de cruzamento, utilizando $nPais$ soluções da população P para gerar cada um dos β novos indivíduos.

Para manter a população de tamanho fixo através das gerações, os β piores indivíduos de cada geração (função *pioresIndividuos*) são removidos sempre que β novos indivíduos são incluídos. Na aplicação de mineração de dados (linhas 14 - 20), β novos indivíduos são gerados pelo procedimento *CGAMD*, cujo funcionamento foi explicado na seção 2.3., utilizando as seqüências frequentes descobertas (*SEQ*). A melhor solução de todas as gerações (com maior valor da função de adaptação $f(x)$) é mantida e retornada pelo algoritmo. Versões mais simples desse algoritmo (AG sem busca local/mineração de dados) são exatamente como o algoritmo do Procedimento 2, exceto pela remoção dos módulos de mineração de dados (linhas 14-20) e/ou o busca local (Procedimento *BuscaLocal*).

Procedimento 2 AGLMD

```

1: Entrada: tamPop, nPaís,  $\beta, \alpha, sup, \mu$ 
2:  $P = CGA(\alpha, tamPop)$ ;
3:  $gen \leftarrow 1$ ;  $novosInd \leftarrow \emptyset$ ;  $conjuntoElite \leftarrow \emptyset$ ;
4:  $melhorInd \leftarrow \emptyset$ ;  $melhorIndGen \leftarrow \emptyset$ ;
5: enquanto critérioParadaNãoAtingido() faça
6:    $novosInd \leftarrow prole(P, \beta, nPaís)$ ;
7:   para todo  $ind \in novosInd$  faça
8:      $ind \leftarrow BuscaLocal(ind)$ ;
9:   fim para
10:   $P \leftarrow P \cup novosInd$ ;
11:   $P \leftarrow P - pioresIndividuos(P, \beta)$ ;
12:  atualize conjuntoElite usando  $P$ ;
13:  se  $(gen \bmod \mu = 0)$  e  $(gen > 0)$  então
14:    descubra seqüências SEQ com suporte mínimo sup no conjuntoElite;
15:     $novosInd \leftarrow CGAMD(SEQ, \beta)$ ;
16:    para todo  $ind \in novosInd$  faça
17:       $ind \leftarrow BuscaLocal(ind)$ ;
18:    fim para
19:     $P \leftarrow P \cup novosInd$ ;
20:     $P \leftarrow P - pioresIndividuos(P, \beta)$ ;
21:  fim se
22:   $melhorIndGen \leftarrow ind \in P | f(ind) \geq f(x) \forall x \in P$ ;
23:  se  $f(melhorIndGen) > f(melhorInd)$  então
24:     $melhorInd \leftarrow melhorIndGen$  ;
25:  fim se
26: fim enquanto
27: retorne  $melhorInd$ ;

```

4. Resultados Computacionais

De nosso conhecimento, não existe um conjunto de instâncias disponível para o PRUMP. Desse modo, foram geradas instâncias (Tabela 2) com diferentes carac-

Parâmetro	Valor
Tamanho da população ($tamPop$)	500
Pais para o cruzamento ($nPais$)	2
Novos indivíduos por geração (β)	50
Grau de aleatoriedade da pop. inic. (α)	0.5
Suporte mínimo (sup)	0.5
Concorrentes ao torneio binário	2
Intervalo de aplicação de mineração de dados (μ)	20

Tabela 1: Parâmetros dos algoritmo.

Problema	Tipo	Fonte TSP-Library	Vértices
1	EUC2D	rat99	99
2	MDS	–	300
3	EUC2D	pr439	439
4	EUC2D	d493	93
5	MDS	–	500
6	MDS	–	500
7	EUC2D	d657	657
8	MDS	–	1.000
9	MDS	–	1.000
10	EUC2D	pr1002	1.002

Tabela 2: Problemas considerados para testes - EUC2D: Coordenadas euclidianas no espaço 2-dimensional ; MDS:Matriz de distâncias simétricas, não euclidianas.

terísticas. Instâncias com coordenadas euclidianas no espaço 2-dimensional (tipo EUC2D) foram criadas a partir de problemas encontrados na *TSP-Library* [11]. Para esse tipo de problema, a produção dos poços foi gerada no intervalo [1,100000]. Em outro tipo de problema, considerou-se uma matriz simétrica de distâncias não-euclidianas (tipo MDS), com valores no intervalo [1,1000]. A produção dos poços nesses problemas foi gerada no intervalo [1,100]. Em todos os casos, a produção dos poços recebeu valores inteiros. A coluna vértices (Tabela 2) corresponde ao número de poços incluindo a ETO (o primeiro vértice). A restrição do limite de tempo para todas as instâncias foi definida de modo que não fosse possível a obtenção de soluções triviais contendo todos os poços.

Dois conjuntos de experimentos foram feitos para avaliação dos algoritmos propostos. Diferentes algoritmos foram construídos através da incorporação ao AG de módulos de Busca Local (AGBL), mineração de dados (AGMD) ou ambos (AG-BLMD).

Os parâmetros de execução (Tabela 1) foram definidos através de experimentos prévios, não detalhados nesse trabalho.

No primeiro conjunto de experimentos o objetivo foi verificar a valor médio de avaliação das soluções finais produzidas por cada algoritmo quando executado com

Problema	AG	AGBL	AGDM	AGBLMD
1	14,44	7,19	5,14	0,00
2	6,78	2,71	1,59	0,00
3	4,12	1,93	0,00	0,91
4	5,09	2,23	1,92	0,00
5	6,90	1,69	3,17	0,00
6	6,21	0,37	3,26	0,00
7	4,09	3,26	1,08	0,00
8	5,93	0,25	0,98	0,00
9	6,23	0,00	2,13	0,41
10	1,16	2,59	0,00	1,85

Tabela 3: Desvio médio da melhor solução conhecida.

limites de tempo. Os seguintes limites de tempo foram impostos: 1.800 segundos para as instâncias 1 e 2; 2.700 segundos para as instâncias 3-7 e 3.600 para as instâncias 8-10.

Resultados médios de 4 execuções para cada problema são apresentados na Tabela 3. Como pode ser visto, a versão básica (AG) produz resultados pobres. Embora AGBL e AGMD melhorem significativamente a qualidade das soluções, não existe uma dominância clara entre os dois métodos. Os melhores resultados ocorrem com o algoritmo AGBLMD.

Em outro conjunto de experimentos, o objetivo foi verificar a distribuição de probabilidade empírica de se atingir um dado valor de solução (isto é, encontrar uma solução tão boa quanto) em função do tempo em diferentes instâncias. Os valores sub-ótimos foram escolhidos de modo que o algoritmo mais lento pudesse terminar em um tempo razoável. Os tempos de execução de 100 execuções independentes para cada instância foram processados. O projeto do experimento segue a proposta de [2]. Os resultados de cada algoritmo foram plotados através da associação do i -ésimo menor tempo de execução t_i com a probabilidade $p_i = (i - 0,5)/100$, gerando os pontos $z_i = (t_i, p_i)$ para $i = 1, \dots, 100$. Embora os experimentos tenham sido executados para todas as instâncias, foram incluídos neste trabalho somente os resultados dos testes para um problema (Figura 3), cujos resultados ilustram os resultados típicos obtidos para todos os problemas. Uma análise simples dos resultados pode ser feita considerando-se o alinhamento das curvas: curvas alinhadas mais à esquerda indicam uma convergência mais rápida do algoritmo, enquanto curvas alinhadas mais à direita indicam algoritmos com convergência mais lenta. Os resultados mostram que o algoritmo mais simples (AG) leva consideravelmente mais tempo para alcançar altas probabilidades acumuladas ($> 50\%$) de se alcançar o valor alvo. Como exemplo, pode-se observar que existe uma probabilidade de 50% do AG atingir o valor alvo em 1.285 segundos, enquanto que para outros algoritmos isso leva aproximadamente 850 segundos. O resultado demonstra que, nos experimentos realizados, apesar de que a inclusão de módulos de Busca Local/Mineração de Dados aumentem o tempo necessário para a realização de cada geração, o algo-

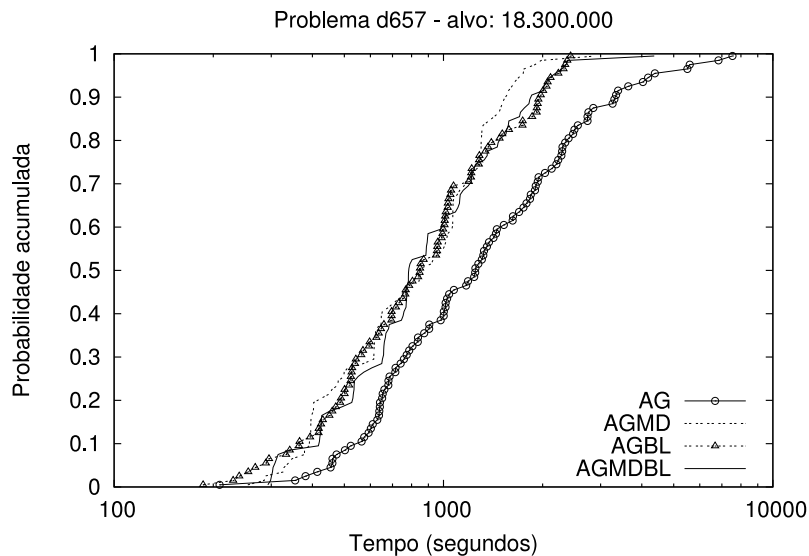


Figura 3: Distribuição empírica de probabilidade acumulada de se atingir o alvo

ritmo resultante da inclusão desses módulos encontra mais rapidamente as soluções com o padrão de qualidade considerado. Como pode ser visto na Figura 3, a superioridade do algoritmo AGBLMD sobre as outras versões não é tão explícita quanto os resultados mostrados na Tabela 3. Acredita-se que esse resultado ocorreu devido ao fato de que o valor de solução alvo foi mais fácil de se encontrar do que a solução média produzida pelo AGBLMD em experimentos com tempos fixos. A escolha por um valor alvo de solução fácil deveu-se ao fato de que alvos mais difíceis consumiriam muito tempo em experimentos com o AG.

5. Conclusões e Trabalhos Futuros

Neste trabalho apresentamos três versões melhoradas de um algoritmo evolucionário. Versões com a inclusão de módulos de busca local e mineração de dados foram apresentadas. Embora aplicações de algoritmos genéticos com busca local sejam abundantes na literatura, aplicações que utilizem mineração de dados para melhorar o desempenho de algoritmos evolucionários ainda são raras. O módulo de mineração de dados aqui proposto corresponde a uma estratégia de intensificação, uma vez que ele tenta descobrir características semelhantes nas melhores soluções encontradas. Os resultados mostram que as versões híbridas oferecem melhores resultados que o algoritmo genético puro. Na maioria dos casos, os melhores resultados foram encontrados com a aplicação de mineração de dados juntamente com o método de busca local.

Abstract. The aim of this work is to present some alternatives to improve the performance of an Evolutionary Algorithm applied to the problem known as Oil Collecting Vehicle Routing Problem. Some proposals based on the insertion of Local Search and Data Mining modules in a Genetic Algorithm are presented. Four algorithms were developed: a Genetic Algorithm, a Genetic Algorithm with a Local Search procedure, a Genetic Algorithm including a Data Mining module and a Genetic Algorithm including Local Search and Data Mining. The results demonstrate that the incorporation of Data Mining and Local Search modules in GA improved the solution quality produced by this method.

Referências

- [1] R. Agrawal, T. Imielinski e A. Swami, Mining association rules between sets of items in large databases, em “Proc. of the ACM SIGMOD Conf. on Management of Data”, Washington, DC, USA, pp. 207-216, 1993.
- [2] R. Aiex, M.G.C. Resende e C.C. Ribeiro, Probability distribution of solution time in GRASP: An experimental investigation, *J. of Heuristics*, **8** (2002), 343-373.
- [3] D.J. Aloise, J.A. Barros e M. Souza, A genetic algorithm for a Oil Retrieval System, (In Portuguese), em “Proc. of the XXXII Brazilian Symposium on Operations Research”, São Paulo, Brasil, 2000.
- [4] J. Beasley, “Population Heuristics”, Handbook of Applied Optimization, Oxford University Press, Oxford, 2002, pp. 138-157.
- [5] J.L. Bresina, Heuristic-biased stochastic sampling, em “Proc. of the Thirteenth National Conference on Artificial Intelligence”, Portland, pp. 271-278, 1996.
- [6] A.A. Freitas, A survey of evolutionary algorithms for data mining and knowledge discovery, em “Advances in Evolutionary Computation” (A. Ghosh e S. Tsutsui, eds.), Springer-Verlag, 2002.
- [7] F. Glover, M. Laguna e R. Marti, Fundamentals of Scatter Search and Path Relinking, *Control and Cybernetics*, **39**, No. 3 (2000), 653-684.
- [8] D.E. Goldberg, “Genetic Algorithms in Search Optimization & Machine Learning”, Addison-Wesley, Menlo Park, 1989.
- [9] J.H. Holland, “Adaptation in Natural and Artificial Systems”, University of Michigan Press, Ann Arbor, 1975.
- [10] P. Moscato, “On Evolution, Search, Optimization Algorithms and Martial Arts: Towards Memetic Algorithms”, Report 826, Caltech Concurrent Computation Program, California Institute Technology, 1989.
- [11] G. Reinelt, TSPLIB - A Traveling Salesman Problem Library, *ORSA J. Comput.*, **3** (1991), 376-384.

- [12] M.G.C. Resende e C.C. Ribeiro, Greedy randomized adaptive search procedures, em "Handbook of Metaheuristics" (F. Glover e G. Kochenberger, eds.), Kluwer, pp. 219-249, 2002.