Apresentado e Publicado nos Anais do XXXV Simpósio Brasileiro de Pesquisa Operacional - 2003

MELHORANDO O DESEMPENHO DE METAHEURÍSTICAS GRASP E ALGORITMOS EVOLUTIVOS : UMA APLICAÇÃO PARA O PROBLEMA DE ÁRVORE DE CUSTO MÍNIMO COM GRUPAMENTOS

Bruno Bastos Lima, Fernando Lourenço Pinho Costa, Luiz Satoru Ochi

Instituto de Computação, Universidade Federal Fluminense
Rua Passo da Pátria, 156, Bloco E, 3° andar
Niterói, Rio de Janeiro, 24210-240

Resumo

Metaheurísticas tem se mostrado técnicas muito eficientes na solução aproximada de vários problemas de elevada complexidade. Entre as metaheurísticas, os Algoritmos Genéticos (AGs) e o Greedy Randomized Adaptive Search Procedures (GRASP) têm obtido muito sucesso na solução de problemas de otimização combinatória. Neste trabalho, propomos metaheuristicas baseadas em conceitos de AGs e GRASP para a solução de uma variante do Problema de Árvore Geradora de Custo Mínimo (AGM) denominada Árvore de Custo Mínimo com Grupamentos (AMG). Neste problema, o conjunto de vértices do grafo associado, é particionado em um conjunto de clusters. Uma solução ótima da AMG, é uma AGM de um subgrafo do grafo original que contenha ao menos um vértice de cada cluster. Este problema é classificado como NP-COMPLETO limitando com isso o uso exclusivo de métodos exatos para a sua solução. O objetivo deste trabalho é mostrar como a inclusão de módulos adicionais podem melhorar o desempenho do AG e do GRASP. Para isso são propostos dois novos AGs para o problema AMG. O primeiro é uma versão básica (AG-B) e o segundo incluindo módulos de busca local e diversificação (AG-BLD). Adicionalmente propomos dois algoritmos GRASP; o primeiro é um modelo básico (GRASP-B) e o segundo incorporando procedimentos de perturbação nos pesos das arestas do problema e um filtro na etapa de construção do GRASP (GRASP-F). Resultados computacionais mostram um bom desempenho dos algoritmos propostos, principalmente do AG-BLD e do GRASP-F, quando comparados com suas versões básicas e com um AG e uma heurística existentes na literatura.

Palavras-chave: metaheuristicas, algoritmos evolutivos, GRASP

Abstract

Metaheuristics have proven to be very effective approaches to solving hard problems. Among them, Genetic Algorithms (GAs) and Greedy Randomized Adaptive Search Procedures (GRASP) have been used successfully in combinatorial optimization problems. In this paper, we propose metaheuristics based on concepts of the GAs and GRASP to solve a variation of the Minimal Spanning Tree (MST) Problem known in literature as The Clustered Minimal Tree (CMT). In this problem, the set of nodes of the associated graph is partitioned in a set of clusters. An optimal solution of CMT is a minimum tree, which include at least a node of each cluster. It is proved that this problem is classified as NP-COMPLETE so the exclusive use the exact methods have been restricted only to small instances. The objective of this work is to show as the inclusion of additional procedures can improve the performance of GA and GRASP algorithms. For this we propose two new GAs and GRASP for the CMT problem. The first one GA is a basic version (AG-B) and another including local search and diversification procedures (AG-BLD). We also propose two GRASP algorithms. The first one is a basic version of GRASP and a second, including a weight perturbation strategy and a filter procedure in the constructive phase (GRASP-F). Computational results show a good performance of the proposed algorithms, mainly of the AG-BLD and GRASP-F, when compared with their basic versions and with an existing GA and a heuristic in the literature.

Keywords: metaheuristics, evolutionary algorithms, GRASP

Introdução

Este trabalho tem como objetivo propor mecanismos que procuram melhorar o desempenho dos Algoritmos Genéticos (AGs) e *Greedy Randomized Adaptive Search Procedure* (GRASP). As propostas são aplicadas neste trabalho na solução de uma generalização do clássico problema da árvore geradora mínima (AGM) aqui denotada por Problema da Árvore Mínima com Grupamentos (AMG). Contudo as propostas deste trabalho podem ser facilmente adaptadas na solução de outros problemas de otimização. A variante do Problema da Árvore Geradora Mínima (AGM) estudada neste artigo é quando os vértices do grafo associado estão particionadas em conjuntos disjuntos e a árvore resultante não deve ligar necessariamente todos os pontos e sim todos os conjuntos. Esta variante da AGM é denotada por Árvore Mínima com Grupamentos (AMG) [1].

Dado um grafo não direcionado, G = (V, E), onde V representa um conjunto de nós (vértices), e E o conjunto de arestas que ligam dois vértices de V. Neste artigo, abordamos uma variante da AGM conhecido na literatura como Árvore Mínima com Grupamentos (AMG). Na AMG, o conjunto de vértices V é particionado em k grupamentos, $V = V_1 \cup V_2 \cup ..... \cup V_k$, podendo estes serem disjuntos entre si ou não. O objetivo do AMG é a de encontrar uma AGM de um subgrafo de G tal que passe por pelo menos um vértice de cada grupamento.

Uma aplicação da AMG envolve sistemas de irrigação de agricultura em ambientes desérticos [1]. Considere k grupamentos (regiões) necessitando de irrigação. Todas estas regiões dividem a mesma fonte de água. Cada região j representa um polígono com alguns vértices v_j. O vértice v_0 é o principal, pois é nele que existe uma fonte de água. O problema consiste em desenvolver a rede de irrigação de menor comprimento, onde cada região ou grupamento deve ter no mínimo um de seus vértices ligado a rede de irrigação. Cada região deve ter um ou mais vértices com pontos de irrigação. A rede não pode cruzar as regiões podendo apenas passar pelas suas fronteiras (arestas). Observe que nesta aplicação, os grupamentos podem não ser disjuntos, isto é, um grupamento pode ter vértices em comum com outros grupamentos como é mostrado na Figura1.

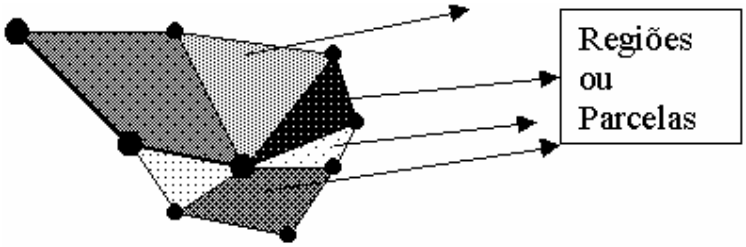

Figura 1. Exemplo de uma aplicação de AMG

Este problema da irrigação então pode ser modelado com uma AMG. Consideramos um Grafo G=(V,E). Os vértices de V representam os vértices das fronteiras das regiões. O peso de cada aresta é associado a distância entre os seus vértices. O problema desta aplicação ter vértices comuns a dois ou mais grupamentos pode ser resolvido através da replicação de parte dos vértices, criando assim um grafo transformado satisfazendo a condição de permitir uma partição em grupos disjuntos. Para isto, podemos usar um novo Grafo G' que é obtido substituindo cada nó que pertence à duas regiões por dois nós, cada um pertencendo somente à uma região, e uma aresta fictícia com peso zero ligando os nós replicados. Desta forma, todo problema de AMG que contiver vértices em dois ou mais grupos pode ser representado na sua definição inicial com grupos disjuntos. Os grupos devem conter pelo menos um nó na solução.

O problema da AMG ao contrário da AGM, é classificado como um problema NP-COMPLETO [1]. Somente para alguns casos particulares, este problema possui complexidade polinomial. Por exemplo, quando o número de grupamentos for igual a um. Neste caso, a solução é

formada por um único vértice. Quando tivermos exatamente dois grupos, a solução será uma aresta de menor peso conectando estes dois grupos. Um outro caso particular simples de ser resolvido, é quando cada grupamento possui um único elemento. Neste caso, a AMG se reduzirá ao problema da AGM.

A literatura apresenta um grande número de trabalhos e aplicações para o problema da arvore geradora de custo mínimo (AGM), contudo pouco existe de nosso conhecimento sobre AMG. Contudo para o problema da AMG, a única referência que dispomos é um trabalho de M. Dror, M. Haouari, J. Chaouachi [1], onde os autores apresentam: uma formulação matemática para o Problema da AMG descrito como um problema de programação linear inteira; quatro heurísticas tradicionais de construção ou de busca local e um Algoritmo Genético (AG) tradicional (sem incorporar etapas de busca local). Alem das heurísticas descritas em [1] é proposto um Algoritmo Genético tradicional aqui denotado por AG-DHC. Neste algoritmo, a representação de uma solução é através de uma lista (zero, um) indicando se um ponto está (um) ou não (zero) numa solução. O nível de aptidão é determinado pelo inverso do comprimento total da arvore solução. A população inicial é obtida através do algoritmo para problema de Steiner em Grafos proposto por Takashashi e Matsumyama [8]. A fase de reprodução é feita através de operadores tradicionais do tipo indivíduo e mutação. O operador crossover proposto é o crossover de um ponto. Este ponto de corte do indivíduo é obtido aleatoriamente. A mutação proposta ocorre com probabilidade de 0,01.

Procedimentos Propostos

A maneira imediata de resolver problemas de otimização combinatória seria simplesmente listar todas as possíveis soluções e selecionar a melhor. Porém com a complexidade e dinamismo dos problemas práticos atuais possuindo normalmente um grande número de soluções candidatas, esta listagem exaustiva se torna inviável para a maioria dos casos. Esta dificuldade tem incentivado o desenvolvimento de métodos que forneçam ao seu final uma solução de boa qualidade, mas não necessariamente garantindo ser a melhor solução existente (solução ótima em um problema de otimização). A característica principal das heurísticas, é a de fornecer uma solução próxima a uma solução ótima em problemas de otimização e num tempo computacional suportável.

Outra facilidade encontrada nos algoritmos heurísticos, é a sua flexibilidade em incorporar novos dados, facilitando assim a sua adaptabilidade na solução de problemas com características dinâmicas.

Contudo, as heurísticas tradicionais possuem também os seus gargalos, e um deles, é a dificuldade histórica destes métodos em superar as armadilhas de um ótimo local ainda distante de um ótimo global em problemas de otimização. Em outras palavras, normalmente as heurísticas quando chegam a um ótimo local tem dificuldades em prosseguir sua busca por soluções de melhor qualidade.

Mais recentemente, a partir do ano de 1985, começaram a surgir na literatura as chamadas metaheurísticas ou heurísticas inteligentes, cuja característica principal, são ferramentas que possibilitam estes de escapar de ótimos locais ainda distantes de um ótimo global.

Dentre as inúmeras metaheurísticas existentes, podemos destacar as que tem produzido as melhores soluções em problemas de otimização combinatória, entre elas: os algoritmos genéticos (AGs), *Tabu Search* (TS), e mais recentemente, GRASP e VNS.

Neste artigo nós propomos a resolução do problema da AMG utilizando conceitos do Algoritmo Genético (AG) e o GRASP) [7].

AGs são membros de uma classe de algoritmos de busca baseados em princípios da evolução natural, que simulam mecanismos de populações genéticas e regras de sobrevivência buscando idéias de adaptação. As buscas por AGs têm se mostrado eficiente em muitos problemas de otimização combinatória. Esta eficiência indica uma robustez do método de busca por AG e a sua flexibilidade. A estrutura inicial de um AG tem sofrido uma gama de variações no sentido de se tentar melhorar o seu desempenho frente a outras metaheurísticas. Um dos principais pontos, é a inclusão de fases de busca local e diversificação nos AGs [2, 3, 4, 5, 9, 10, 11].

O GRASP é uma metaheurística desenvolvida em 1985 por Tom Feo e Maurício Resende [7]. Sua metodologia consiste em um processo iterativo onde cada iteração possui duas fases: uma fase de construção, na qual uma solução viável é construída, seguida de uma fase de melhorias, cujo objetivo é encontrar um ótimo local.

A fase de construção do GRASP é iterativa, adaptativa, randômica e pode ser gulosa. Ela é iterativa porque a solução inicial é construída elemento a elemento. é adaptativa porque os benefícios associados a cada elemento são levados de uma iteração para outra, refletindo as mudanças ocasionadas pela seleção prévia de outros elementos e pode ser gulosa porque a adição de cada elemento pode estar restrita a uma lista de apenas um candidato. Nesta fase, a escolha do próximo elemento a ser adicionado é determinado pela ordenação de todos os elementos candidatos em uma lista de candidatos LC. Esta lista é construída de acordo com uma função gulosa g: C→R, que mede o benefício de se selecionar cada elemento. O componente probabilístico do GRASP é caracterizado pela escolha aleatória de um dos melhores elementos de uma lista de candidatos restritos (LCR) composto dos *k* melhores candidatos da LC.

Algoritmos Propostos

Neste trabalho mostramos como o desempenho dos modelos básicos de AGs e GRASP podem ser melhorados com a introdução de módulos adicionais nestes métodos. Para isso, iniciamos com versões básicas das metaheurísticas AG e GRASP denotados respectivamente por AG-B e GRASP-B. Em seguida propomos a inclusão de módulos adicionais nestes métodos, incluindo respectivamente módulos de busca local e uma nova técnica de diversificação para os AGs (AG-BLD) e uma técnica de perturbação dos dados de entrada e um método de filtro na etapa de construção do GRASP (GRASP-F).

Um Algoritmo Genético Básico (AG-B)

Apresentamos uma descrição das principais ferramentas e representações do AG proposto para resolução do problema da AMG:

Representação dos Indivíduos

Indivíduos são as representações das soluções para o problema. Em nossa representação um indivíduo é um vetor binário (0,1) de m posições, onde m representa o número de nós do grafo associado. Cada nó da AMG está representado por uma posição no indivíduo. Se a posição de um determinado nó está com valor "0", então ele não pertence a solução. Se em sua posição está um "1", então ele está na solução. No nosso AG, um gene tem uma definição diferente do que normalmente é usada nos AGs. Na maioria dos AGs, um gene eqüivale a uma posição do indivíduo. Mas na nossa proposta, um gene representa um grupo (cluster). Portanto o número de genes de um indivíduo está associado ao número de grupos da AMG. Cada indivíduo deverá ter pelo menos uma posição com valor "1" em cada gene, ou ao menos um nó visitado em cada grupo. No exemplo da Figura 2, o indivíduo terá 6 posições e 3 genes. Um gene com 3 posições, um com 2 e outro com apenas uma posição, conforme a Figura 3.

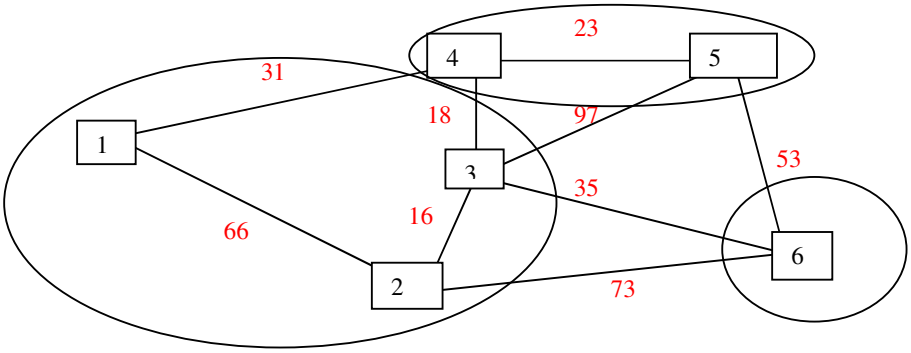

Figura 2. Exemplo de um indivíduo com três genes.

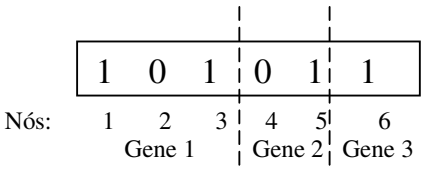

Figura 3. Estrutura de um indivíduo .

População

A população é um conjunto de indivíduos ou soluções. Esta população contém os indivíduos que sobrevivem a cada iteração e participam da evolução genética por mutação ou recombinação para formação da próxima geração. Foi adotado um número fixo para o tamanho da população. Adicionalmente foi feito um estudo dos melhores indivíduos de uma população. Chamamos a melhor solução ou o melhor indivíduo da população de *indivíduo campeão*. O algoritmo é inicializado com uma população de soluções gerada aleatoriamente. Partindo destas soluções o algoritmo busca evoluir e chegar a melhor solução ao seu final. Mesmo tendo uma inicialização aleatória é obedecido o critério de que toda solução deve conter pelo menos um nó de cada grupo. Diferentemente da proposta de *Dror, M., Haouari M., Chaouachi*, J. [1], que usa uma penalidade para soluções não válidas por terem grupos vazios.

Avaliação das Soluções

Para avaliar cada solução (indivíduo), lembrando que buscamos os menores pesos totais possíveis para que estas sejam de custo mínimo, devemos determinar o peso (custo) de cada indivíduo. Como já sabemos os nós que compõe cada solução, devemos agora determinar seu peso ou custo. Para avaliar cada indivíduo é aplicado o Algoritmo Prim [6]. Este algoritmo irá produzir para cada solução uma AMG necessariamente envolvendo todos os nós da solução. A soma de todos os pesos de cada arco ou aresta da AMG construída é dito o peso ou custo da solução. Portanto é guardado o peso de cada indivíduo e então as soluções são ordenadas de forma crescente de seus pesos. Lembrando que a melhor solução é aquela que possui o peso menor. A solução é dita campeã se ela é a melhor solução da população atual.

Seleção dos indivíduos pais

Depois de avaliada cada solução devemos agora escolher os pais que participaram da próxima etapa de reprodução (através dos operadores crossover ou mutação). Isto é, devemos selecionar os indivíduos da presente população que gerarão filhos. Esta escolha é feita pelo método da roleta. Neste método cada elemento recebe uma fatia da roleta, proporcional a sua aptidão (inverso do seu custo), assim quanto melhor for a solução, maior será a fatia associada, e portanto, maior a chance dela ser escolhida. Porém este método possibilita a escolha de qualquer elemento da população, não só as melhores. Isso é feito para dar prioridade às melhores soluções, porém dando oportunidades para todos os indivíduos, garantido assim uma maior variedade genética na população.

Reprodução

A reprodução é feita por um operador que é uma variante do crossover tradicional. Ou seja, são escolhidos alguns indivíduos pais (geralmente dois) e estes são combinados de forma que cada um passe informações (genes) para os filhos gerados (geralmente um) de forma trocada. O indivíduo filho é a união de metade de cada indivíduo pai. Nós propomos a reprodução entre indivíduos com a passagem de grupos inteiros (genes) dos indivíduos pais para o filho, e este filho será composto de metade por grupos de um pai e a outra metade por grupos do outro pai. Diferentemente do crossover

tradicional, onde são trocadas partes aleatórias, determinadas pelo ponto de corte. Considere a ilustração a seguir da Figura 4.

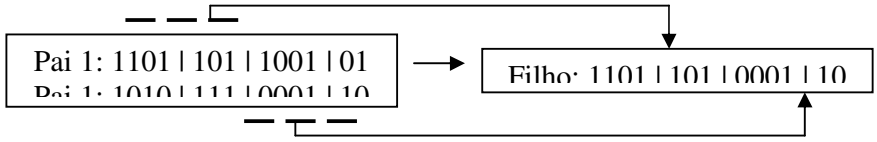

Figura 4. Exemplo da reprodução.

Mutação

A mutação é um operador que é aplicado aos próprios indivíduos, sem que haja cruzamento entre eles. Os indivíduos que sofrerão mutação podem ser escolhido pelo método da roleta. A mutação é um modificação feita no indivíduo afim de melhorá-lo, ou seja, torna-lo mais apto ou com o intuito de efetuar uma diversificação na atual população. Evitando com isso uma parada prematura do AG num ótimo local, às vezes distante de um ótimo global. Mesmo com as modificações nas soluções, devemos garantir que elas continuem viáveis. A mutação é feita nos indivíduos com a abertura de uma janela de posição inicial e final aleatória. Dentro desta geramos novamente cada posição aleatoriamente.

Algoritmo Genético com Busca Local e Diversificação (AG-BLD)

Esta versão é baseada no AG-B descrito anteriormente, mas incorporando conceitos de busca local e uma nova técnica de diversificação nos AGs, descritos a seguir.

Diversificação

A diversificação na população atual deve ser efetuada se o AG ficar por muitas iterações consecutivas com um nível de atualizações abaixo de um nível mínimo. Isso é um sintoma de que estamos num ótimo local as vezes distante de um ótimo global. A idéia da diversificação de uma população é a de permitir que o AG busque novas regiões ainda não exploradas ou pouco exploradas para tentar obter soluções de melhor qualidade. Para diversificar uma população, substituimos a atual população, por outra feita com replicações dos k melhores indivíduos gerado até o momento pelo AG. Para cada cópia, aplicamos uma mutação como descrito anteriormente. Este método visa buscar pequenas modificações nas melhores soluções ou na melhor, de uma população.

Também propomos uma diversificação em soluções que são encontradas duplicadas dentro da população. A cada nova população temos uma identificação de soluções iguais. Nunca queremos ter soluções idênticas em uma mesma população, pois isto não representa nenhum ganho. Para não se perder a característica da solução original, é feita a modificação em um grupo apenas. Este grupo, sorteado aleatoriamente, terá seus nós todos zerados e apenas um será sorteado para ficar ativo, contendo o valor "1". Com isso, transformamos soluções iguais em duas que diferem em apenas de um gene.

Busca Local

Para que seja feita a busca local, primeiro é definido a vizinhança de um ponto e um ponto de partida da busca local. O ponto de partida será a solução construída inicialmente. A busca foi definida de duas formas: na primeira a vizinhança de soluções está apenas a uma mudança na solução inicial, e sempre retornamos a solução inicial independentemente se a solução vizinha foi melhor, esta versão é conhecida como *best improvement*. Na segunda versão, a solução inicial (semente) é atualizada a cada melhora encontrada (*first improvement*).

A busca local proposta é um método exaustivo onde são testadas cada uma das soluções que diferem em apenas um nó da solução de partida inicial. Para isso, cada posição de nó é percorrida e mudada o seu valor, esta é testada e é guardada. O valor é restaurado. Então passamos a próxima

posição, mudando o valor e testando. Isso até o fim do indivíduo. Como mudar o valor, entendemos passar o nó de "0" para "1", ou seja, de inválido para válido ou vice-versa. Propomos um filtro na busca local para tornar o processo mais dinâmico. Para não testar soluções que tem grandes indicativos de serem ruins, durante o processo fazemos este filtro com alguns critérios: se o grupo ficar sem nó ativo ("1"), não testamos (solução é inválida por definição). Se o grupo já tem um determinado número de nós ativos ("1"), não ativamos mais nenhum até que outro seja desativado (em sua maioria, grupos com muitos nós ativos não são bons).

Dinâmica da População

Nós propomos neste artigo uma dinâmica da população de indivíduo de forma que a variação genética seja a maior possível e mesmo assim não se perca boas cargas genéticas. Como definimos anteriormente, uma população é o conjunto de n indivíduos que a cada iteração do algoritmo participará das variações genéticas (mutação, diversificação ou busca local) ou combinações genéticas para compor a população da próxima iteração. Quem estará na próxima população é definido pelo critério de sobrevivência.

Nós definimos uma iteração como a criação de uma nova população. Como vimos, existem várias maneiras de se gerar novos indivíduos e elas são usadas dinamicamente dentro da rotina. A rotina começa com uma população de n indivíduos gerados aleatoriamente, como visto na inicialização. A partir daí são feitas inúmeras iterações com apenas as gerações de soluções com o cruzamento entre indivíduos (reprodução). Para que o algoritmo fosse adaptativo, este número de iterações de reprodução não é fixo e depende diretamente da taxa de mudança desta população e a taxa de atualização do campeão. A partir do momento em que estas taxas ficam abaixo do limite, a população entra para fazer uma variação genética com a mutação. E novamente o que define o número de iterações de mutações são as taxas que medem a renovação da população e a atualização do indivíduo campeão. Para a mutação estas taxas devem ser maiores que limites determinados, e estes limites são mais rígidos do que na reprodução. As diversificações e a busca local entram como alternativa para se tentar alcançar soluções melhores. Estas serão executadas sempre que as taxas de renovação de uma população estiverem baixas. Em um determinado número de iterações também é identificado se o campeão foi atualizado, neste caso, nele será feita a busca local ao seu redor.

Algoritmos GRASP

Além dos AGs, propomos dois algoritmos GRASP. O primeiro utiliza conceitos descritos em modelos tradicionais GRASP e o segundo incorpora conceitos de filtro na fase de construção e utiliza um procedimento de perturbação nos dados de entrada. Este último procedimento é mais uma alternativa para que metaheurísticas melhorem a sua capacidade de busca sem alterar a estrutura do algoritmo. Ou seja, com a perturbação nos dados de entrada, espera-se que determinados critérios de prioridade sejam alterados e assim propiciar uma busca mais ampla do algoritmo.

GRASP Básico (GRASP-B)

No GRASP-B, os dois módulos (construção e busca local descritos a seguir), são repetidos um determinado número de vezes e a melhor solução é então a solução do algoritmo.

Etapa de Construção

Na fase de construção da solução, foram usadas duas variantes da Heurística1 proposto em[1]. Esta heurística percorre cada cluster a fim de achar a menor ligação existente entre um cluster e a árvore parcial construída. Ele começa colocando todos os clusters em uma lista e procurando a aresta de menor custo que liga dois clusters distintos. Assim esta aresta é a primeira aresta da árvore parcial, e estes dois clusters que já estão ligados, são eliminados da lista. A seguir, a cada passo é escolhido a aresta de menor custo entre a árvore parcial construída e um cluster ainda não ligado ou que ainda está

na lista. O procedimento continua até que não exista nenhum cluster sem conexão. Chamamos este algoritmo de H1.

No nosso GRASP, duas variantes deste H1 são usadas na etapa de construção. A primeira variante ao invés de escolher sempre a melhor aresta no grafo para começar a construção, é sorteado aleatoriamente um entre os 5 melhores candidatos, que formam uma lista de candidatos restrita (LCR), chamamos esta versão H1-1. É importante lembrar que este H1-1 só difere do original na ligação da primeira aresta, que é feita de forma aleatória nesta LCR.

Numa outra versão é usado ainda mais aleatoriedade. A cada vez que uma aresta for conectado (do primeiro ao último) é construído a sua LCR e selecionado uma aresta desta lista aleatoriamente. Chamamos esta versão de H1-2.

Busca Local

Para que seja feita a busca local, primeiro é definido a vizinhança de um ponto e um ponto de partida da busca local. O ponto de partida será a solução construída inicialmente. A busca foi definida de duas formas: na primeira a vizinhança de soluções está apenas a uma mudança na solução inicial, e sempre retornamos a solução inicial independentemente se a solução vizinha foi melhor, esta versão é conhecida como *best improvement*. Na segunda versão, a solução inicial (semente) é atualizada a cada melhora encontrada (*first improvement*).

A busca local é feita percorrendo os clusters. Em cada cluster identificamos qual o nó que esta presente na atual solução e o retiramos da solução. Então percorremos todos os outros nós ligando um de cada vez a atual solução e testando. As melhores soluções são guardadas numa *população elite*. Neste caso é importante salientar que a vizinhança não muda, pois só testamos e guardamos as soluções geradas, mas a busca local é sempre feita sobre a solução gerada na fase de construção. Numa versão mais econômica, a cada iteração são analisadas apenas uma certa percentagem dos clusters para agilizar o processo.

GRASP com perturbação e Filtro (GRASP-F)

Esta versão é baseada no GRASP-B, incorporando procedimentos de: diversificação, filtro na etapa de construção e usando perturbação nos dados de entrada. No entanto, para que os tempos computacionais desta versão não sejam elevados em relação aos exigidos pela versão básica, os módulos adicionais de diversificação quando ativados substituem os mecanismos de reprodução.

Diversificação

Quando já temos uma *população elite* completa com as m melhores soluções geradas até o momento (m é um parâmetro de entrada), será feita uma diversificação no GRASP. Nesta fase percorremos todos as arestas da árvore solução e buscamos um caminho mínimo que ligue cada dois nós adjacentes da solução, substituindo posteriormente se for vantajoso, o caminho mínimo pela aresta até então utilizada. A troca valerá a pena sempre que o custo do caminho mínimo for menor que o peso da aresta utilizada na atual solução. Recordemos que neste problema estamos usando grafos onde os pesos de cada aresta são gerados aleatoriamente, não satisfazendo necessariamente a desigualdade triangular. Este caminho mínimo é obtido através do algoritmo de caminho mínimo de Dijkstra.

Perturbação nos dados de entrada

É proposto também uma função de perturbação da matriz de entrada. Este é um método que busca gerar soluções iniciais bastante diversificadas, pois a cada iteração a matriz de custos (distâncias) apresenta valores ligeiramente distintos do original. Esta perturbação é utilizada apenas para a construção inicial e logo após uma solução for construída com base na matriz perturbada, a matriz original é carregada e esta mesma solução tem seu peso reavaliado corretamente.

Filtro

O filtro é um método para que a solução construída na etapa de construção já seja um pouco otimizada. Ao invés de construir apenas uma solução, são construídas p soluções e só a melhor é avaliada para fazer a busca local em cada iteração do GRASP.

Implementações e Resultados Computacionais

Foram implementados os seguintes algoritmos: dois algoritmos existentes na literatura, dois Algoritmos Genéticos e dois GRASP propostos neste trabalho. Os primeiros dois são propostos no trabalho de M. Dror, M. Haouari, J. Chaouachi [1], já descritos anteriormente neste artigo: Heu-1 e um AG que contém operações básicas dos modelos tradicionais de AGs e será denotada por AG-DHC. Nós propomos inicialmente dois novos AGs. O primeiro é também uma versão tradicional de AG, sem as etapas de diversificação e busca local denominado AG-B**.** O AG-B embora básico difere do AG-DHC em alguns aspectos. O segundo AG proposto contem todas as funcionalidades propostas e é denominado AG-BLD. Estas duas diferentes implementações permitem avaliar o custo x benefício dos módulos propostos.

Também foram implementados as duas versões do GRASP propostos: GRASP-B e GRASP-F. O primeiro possui a estrutura de um GRASP padrão. Não possui Filtro nem Perturbação da matriz de entrada. Na construção inicial utiliza o H1-1 anteriormente descrito. O segundo algoritmo GRASP foi feito com intuito de utilizar todos os módulos possíveis. Assim ele utiliza o algoritmo H1-2 na fase de construção, uma perturbação da matriz de entrada a cada iteração do GRASP e um filtro, que constrói várias soluções, mas apenas a melhor é analisada na etapa de busca local do GRASP. Chamaremos esta versão de *GRASP-F.* A implementação dos algoritmos propostos foram todos feitos na linguagem C de programação. Devido a inexistência de nosso conhecimento de instância do problema em bibliotecas públicas, geramos uma bateria de problemas testes onde os grafos (matrizes) iniciais são em parte grafos completos e outros esparsos (com uma percentagem de arcos inexistentes). Os parâmetros dos testes podem ser vistos nas Tabelas 1 e 2:

Parâmetros:	
Sistema Operacional	Linux
Velocidade	1800 MHz
Memória	256 MB
Valores dos pesos das arestas	entre 1 e 100
Tamanho da População:	250
Número de Iterações (AGs e GRASP)	1000

Tabela 1: Parâmetros dos testes realizados.

N°	Total de Nós	# arestas	Tam. do Grupo Médio	Variação do Tam. do Grupo
1	20	85%(161)	10%(2)	50%(1)
2	40	85%(663)	10%(4)	50%(2)
3	60	85%(1504)	10%(6)	50%(3)
4	80	85%(2686)	10%(8)	50%(4)
5	100	85%(4207)	10%(10)	50%(5)
6	120	85%(6069)	10%(12)	50%(6)
7	140	85%(8270)	10%(14)	50%(7)
8	160	85%(10812)	10%(16)	50%(8)
9	180	85%(13693)	10%(18)	50%(9)

N°	Total de Nós	# arestas	Tam. do Grupo Médio	Variação do Tam. do Grupo
10	200	85%(16915)	10%(20)	50%(10)
11	400	75%(59850)	1%(4)	40%(2)
12	600	75%(134775)	1%(6)	40%(3)
13	800	75%(239700)	1%(8)	40%(4)
14	1000	75%(374625)	1%(10)	40%(4)
15	1200	75%(539550)	1%(12)	40%(5)
16	1400	75%(734475)	1%(14)	40%(6)
17	1600	75%(959400)	1%(16)	40%(7)
18	1800	75%(1214325)	1%(18)	40%(8)

Tabela 2: Configurações dos Testes

Na Tabela 2 são descritos os parâmetros usados. A coluna 1 (Nº) identifica a instância, a coluna 2 (Total de Nós) representa o número de vértices do problema, a coluna 3 (# arestas) representa o número de arestas conectadas a cada vértice, a coluna 4 (Tam. do Grupo Médio) representa o número

médio de vértices que pertence a cada grupo e a coluna 5 (Variação do Tam. do Grupo) mostra o quanto os grupos podem variar para mais ou para menos de seu tamanho médio.

	Heu-1		**AG-B**		**AG-DHC**		**AG-BLD**		**Grasp-B**		**GRASP-F**	
N°	custo	tempo	custo	tempo	custo	tempo	custo	tempo	custo	tempo	custo	tempo
1	44	1	**31**	95	**31**	188	**31**	181	**31**	1	**31**	1
2	33	1	**33**	104	33	148	**33**	149	36	1	**33**	2
3	23	1	**17**	113	18	198	21	146	24	1	22	3
4	26	1	**11**	132	12	170	20	149	12	2	13	5
5	29	1	**13**	148	16	210	**13**	181	**13**	1	**13**	7
6	29	1	25	189	20	217	19	201	**18**	2	19	8
7	31	1	21	160	13	183	**9**	174	12	2	10	10
8	25	1	19	162	17	191	13	182	15	3	**12**	13
9	42	1	26	170	10	204	**9**	188	12	4	**9**	15
10	38	1	23	175	10	209	**10**	199	14	4	**10**	19
11	145	1	142	600	105	680	**103**	985	140	195	110	550
12	154	1	141	856	99	1164	**94**	1113	130	259	**94**	787
13	153	1	148	1250	101	1475	**99**	1440	133	369	**99**	1077
14	149	1	147	1524	103	1831	**100**	1800	131	478	**100**	1277
15	148	1	146	2014	100	2097	**98**	2178	122	567	**98**	1726
16	158	1	153	2412	101	2486	**98**	2625	130	668	**98**	1829
17	160	1	155	3005	103	3036	**100**	3152	125	796	**100**	2190
18	155	1	153	3555	103	3596	**100**	3747	129	924	**100**	2378

Tabela 3: Resultados computacionais dos algoritmos

Na Tabela 3, a coluna 1 (Nº) apresenta o número do teste. As colunas 2, 4, 6, 8, 10 e 12 mostram o valor das soluções encontradas por cada algoritmo e as colunas 3, 5, 7, 9, 11 e 13 mostram os tempos em segundos gastos por cada algoritmo.

N°	**Heu-1**	**AG-B**	**AG-DHC**	**AG-BLD**	**GRASP-B**	**GRASP-F**
1	0,42	x	x	x	x	x
2	x	x	x	x	0,09	x
3	0,35	x	0,06	0,24	0,41	0,29
4	1,36	x	0,09	0,82	0,09	0,18
5	1,23	x	0,23	x	x	x
6	0,61	0,39	0,11	0,06	x	0,06
7	2,44	1,33	0,44	x	0,33	0,11
8	1,08	0,58	0,42	0,08	0,25	x
9	3,67	1,89	0,11	x	0,33	x
10	2,80	1,30	x	x	0,40	x
11	0,41	0,38	0,02	x	0,36	0,07
12	0,64	0,50	0,05	x	0,38	x
13	0,55	0,49	0,02	x	0,34	x
14	0,49	0,47	0,03	x	0,31	x
15	0,51	0,49	0,02	x	0,24	x
16	0,61	0,56	0,03	x	0,33	x
17	0,60	0,55	0,03	x	0,25	x
18	0,55	0,53	0,03	x	0,29	x
DM	*1,01*	*0,52*	*0,09*	*0,06*	*0,24*	*0,04*

Tabela 4: Resultados comparativos dos custos, onde DM = desvio médio

Na Tabela 4 as colunas de 2 a 7 mostram o desvio de cada solução em relação ao *best(i),* i =1...18, representado por um "x" nesta tabela. O valor *best(i)* representa a melhor solução comparando os resultados de todos os algoritmos aqui testados para cada instância i = 1...18. Nesta tabela, a última linha mostra o desvio médio DM (*soma de todos os desvios dividido pelo número de instâncias (18))* encontrado em cada algoritmo, comparando seus resultados com *best(i),* para cada instância i.

N°	AG-B	AG-DHC	AG-BLD	GRASP-B	GRASP-F
1	94,00	187,00	180,00	x	x
2	103,00	147,00	148,00	x	1,00
3	112,00	197,00	145,00	x	2,00
4	65,00	84,00	73,50	x	1,50
5	147,00	209,00	180,00	x	6,00
6	93,50	107,50	99,50	x	3,00
7	79,00	90,50	86,00	x	4,00
8	53,00	62,67	59,67	x	3,33
9	41,50	50,00	46,00	x	2,75
10	42,75	51,25	48,75	x	3,75
11	2,08	2,49	4,05	x	1,82
12	2,31	3,49	3,30	x	2,04
13	2,39	3,00	2,90	x	1,92
14	2,19	2,83	2,77	x	1,67
15	2,55	2,70	2,84	x	2,04
16	2,61	2,72	2,93	x	1,74
17	2,78	2,81	2,96	x	1,75
18	2,85	2,89	3,06	x	1,57

Tabela 5: Resultados comparativos dos tempos

Na Tabela 5 as colunas de 2 a 6 mostram o desvio de tempo acima da solução *best tempo(i),* i = 1...18, representado por um "x" nesta tabela. O valor *best tempo(i*) representa o menor tempo comparando os tempos de execução de todos os algoritmos aqui testados para cada instância i = 1...18.

Observamos pelos resultados da tabela 3 e 4, que as versões básicas do AG e GRASP foram superadas pelas versões AG-BLD e GRASP-F respectivamente. O pior desempenho dentre as metaheurísticas foram o AG proposto em [1] (AG-DHC) e o GRASP-B que obteram somente a melhor solução em três problemas testes, seguido do AG-B que obteve cinco melhores soluções, o GRASP-F veio em seguida com 13 melhores e o melhor desempenho foi do AG-BLD com 14 melhores soluções. O desempenho dos algoritmos pode ser medido também em função dos desvios médios (DM) dado na última linha da tabela 4. Por este critério, a classificação do melhor para o pior fica sendo: GRASP-F, AG-BLD, AG-DHC, GRASP-B, AG-B, e Heu-1. Um fato importante é que a medida em que as dimensões crescem (problema 9 em diante), as versões AG-BLD e GRASP-F aqui propostos sempre obtiveram as melhores soluções, colocando-os por conseguinte como os melhores algoritmos para problemas de elevadas dimensões. Finalmente em relação aos tempos computacionais exigidos, a tabela 5 mostra que a versão mais rápida é o GRASP-B seguido pelo GRASP-F. Os AGs, apresentam tempos bem maiores que as duas versões GRASP, principalmente em instâncias onde o número de vértices em cada cluster é elevado (instâncias de 1 a 10). Em suma, levando em conta a qualidade das soluções e o tempo exigido, as nossas simulações mostram um melhor desempenho do GRASP-F, principalmente em instâncias de elevadas dimensões. Outro aspecto interessante e importante é que AG-BLD melhora consideravelmente a qualidade das soluções em relação a AG-B e AG-DHC exigindo tempos computacionais muito similares. A questão dos tempos similares ocorre como já explicado, devido ao fato de no AG-BLD, muitas populações serem criadas pelo mecanismo de diversificação que *substitui* os operadores tradicionais, isto é, o módulo de diversificação *não é um módulo adicional,* portanto não onera os tempos finais do algoritmo.

Conclusões

Neste trabalho apresentamos propostas para melhorar o desempenho das metaheurísticas Algoritmos Evolutivos e GRASP na solução de problemas de otimização combinatória de elevada complexidade computacional. Para avaliar os algoritmos propostos estes foram utilizados na solução aproximada de uma extensão do Problema da Árvore Geradora de custo mínimo, denotada por Árvore Mínima com Grupamentos (AMG). Este problema está classificado como um problema NP-Completo o que limita o uso exclusivo de técnicas exatas para a sua solução. Os resultados computacionais obtidos mostram um futuro muito promissor para as propostas aqui sugeridas para obter versões mais eficientes de AGs e GRASP, através do uso de módulos de busca local, diversificação, filtro e perturbação nos dados de entrada.

Bibliografia

[01] M. Dror, M. Haouari, and J. Chaouachi, "Generalized Spanning Trees", European Journal of . Operational Research,120, 583-592, 2000.

[02] L. M. A. Drummond, D. S. Vianna and L. S. Ochi, "A Parallel Genetic Algorithm for the Vehicle Routing Problems", In Future Generations on Computer Systems Journal - Elsevier, vol. 14(5-6), 285-292,1998.

[03] L.M.A.Drummond, L. S. Ochi and R. M.V.Figueiredo, "Design and Implementation of a Parallel Genetic Algorithm for the Travelling Purchaser Problem", In APPLIED COMPUTING'97/ACM, 257-263, 1997, (capitulo de livro), ACM, Association for Comp. Machinery,Inc., NY.

[04] L. S. Ochi and M. L. Rocha, "A new hybrid evolutionary algorithm for the vehicle routing and scheduling problems", In Proc. Of the Ninth International Conference on Intelligence Systems: Artificial Intelligence Applications for the New Millennium, 135-140, 2000, Louisville, USA, International Society for Computer & Their Applications – ISCA.

[05] L. S. Ochi and P. W. P. Vieyra, "A Hybrid Metaheuristic Using Genetic Algorithm and Ant Systems for the Clustered Traveling Salesman Problem", In Proc. of the III Metaheuristic International Conference (MIC'99), 365-371, Angra dos Reis, RJ, 1999.

[06] C. Prim, "Shortest connection networks and some generalization", Bell System Technological Journal 36, 1389-1401, 1957.

[07] M.G.C. Resende, and T. A. Feo, "Greedy Randomized Adaptative Search Procedures", Journal of Global Optimization 1995,1-27

[08] H. Takahashi, and A. Matsumyama, "An Aproximation solution for the Steiner Problem in graphs", Math Jpn. 24, 1980.

[09] D. S. Vianna, L. M. A. Drummond and L. S. Ochi, "A New Parallel Hybrid Evolutionary Metaheuristic for the Vehicle Routing Problems", In Lecture Notes in Computer Science, LNCS, volume 1586, 183-192, Editors**:** J. Rolim et al.; Springer, 1999.

[10] D. S. Vianna, L. M. A. Drummond, and L. S. Ochi, "An Asynchronous Parallel Metaheuristic for the Periodic Vehicle Routing Problems", In Future Generations on Computer Systems Journal, 17(4), 379-386 - ELSEVIER, 2001.

[11] D.Vianna, Luiz Satoru Ochi and L.M.A.Drummond, "Design and Implementation of an Improved Parallel Evolutionary Algorithm for the Vehicle Routing and Scheduling Problems", In Proc. of the III Metaheuristic International Conference (MIC'99), 193-199, Angra dos Reis, RJ, 1999.