Implementação da Metaheurística GRASP para o Problema do Caixeiro Viajante Simétrico

Ademir Roberto Freddo
Robison Cris Brito

Universidade Federal do Paraná
Tópicos em Inteligência Artificial
Drª Aurora Pozo

Resumo

Este trabalho apresenta a metaheurística GRASP para o problema do Caixeiro Viajante Simétrico. Propõe-se a hibridização do GRASP com um método de busca local *Hill-Climbing*, com as metaheurísticas *Simulated Annealing* e VNS Descendente. Essas hibridizações geraram três diferentes implementações que foram testadas e avaliadas em uma série de testes. O objetivo das implementações e dos testes é mostrar a importância da hibridização nas metaheurísticas como componente de equilíbrio na diversificação e intensificação da busca.

1. Introdução

Durante muitos anos têm-se estudado heurísticas para resolver problemas de otimização combinatória NP-difícil. As heurísticas são métodos de busca que tiram proveito de características e informações do próprio problema a ser explorado, facilitando o encontro de um mínimo global no espaço de busca [1]. As heurísticas são limitadas e fornecem sempre a mesma solução quando iniciadas de um mesmo ponto de partida. As metaheurísticas vêm suprir esta deficiência e tem como objetivo principal explorar um espaço de pesquisa de forma inteligente, ou seja, encontrar soluções de alta qualidade movendo-se para áreas não exploradas quando necessário.

Metaheurísticas são procedimentos de busca com capacidade de escapar de ótimos locais, focando na eficiência e uma maior exploração do espaço de busca, destinados a resolver problemas de otimização. As metaheurísticas possuem algumas características, entre elas [1]:

- utiliza estratégias para guiar o processo de busca;
- explora de maneira eficiente o espaço de busca, com o objetivo de encontrar uma solução ótima próxima ao ponto atual;
- utiliza técnicas de buscas locais a complexos processos de aprendizagem;
- possui mecanismos que evitam o aprisionamento dos mesmos em áreas restritas do espaço de busca;
- faz uso de um domínio específico de conhecimento com uma heurística para estratégias de busca;
- armazena experiências de buscas, utilizando-as para guiar o algoritmo nos futuros processos de buscas.

Há diversos métodos de metaheurísticas. Alguns podem ser vistos como extensões de algoritmos de busca local que procuram sair de regiões com poucas possibilidades de encontrar ótimas soluções e ir para locais onde as melhores soluções podem estar presentes. Isso é proposto por algoritmos como Busca Tabu [2], Busca Local Iterativa (ILS – *Iterated Local Search*) [3], Busca em

Estrutura de Vizinhança Variável (VNS – *Variable Neighborhood Search*) [4,5], *Simulated Annealing* [6] e GRASP (*Greedy Randomized Adaptive Search Procedures*) [7,8].

Há também outras técnicas inspiradas na capacidade da natureza de adaptação ao meio onde vivem, através da recombinação e mutação de indivíduos, mais precisamente, recombinar soluções atuais (soluções pai) para melhorar futuras soluções(soluções filho). Nessa classe estão algoritmos de computação evolutiva, como os Algoritmos Genéticos [9,10]. Outros algoritmos são inspirados no comportamento de indivíduos ao meio, como Colônia de Formigas [11] e Nuvem de Partículas [12]. A Colônia de Formigas é uma abordagem inspirada no comportamento das formigas para encontrar o menor caminho entre a origem do alimento e seu ninho, onde as mesmas depositam o feromônio para marcar a trajetória. A Nuvem de Partículas é motivada pela simulação do comportamento social existentes na natureza.

Para que uma metaheurística explore um espaço de busca de forma inteligente, obtenha soluções de ótima qualidade e consiga mover-se para áreas não exploradas quando necessário, os conceitos de intensificação e diversificação devem ser atingidos. A intensificação consiste em concentrar as buscas em regiões promissoras em torno de boas soluções. A diversificação corresponde em fazer buscas em regiões ainda não exploradas. Toda metaheurística deve possuir componentes de diversificação e intensificação. Esses dois componentes devem ser balanceados e bem utilizados. Para que isso ocorra, é necessário integrar metaheurísticas realizando uma hibridização. O resultado da hibridização pode resultar em melhores resultados do que uma metaheurística sozinha.

Este trabalho está focado na utilização da metaheurística GRASP e na hibridização de GRASP com outras metaheurísticas como SA e VNS para aplicação no problema do Caixeiro Viajante Simétrico.

O problema do Caixeiro Viajante é um problema clássico de otimização combinatória, possui variações generalizações e adaptações para problemas semelhantes. Trata-se, portanto de um problema de grande importância e interesse prático, uma vez que resultados satisfatórios encontrados no problema assimétrico podem ser aplicados em outros problemas.

Na Seção 2 é apresentado o problema do Caixeiro Viajante e exemplos de aplicações para o mesmo. A fundamentação das metaheurísticas utilizadas para a resolução do problema é apresentada na Seção 3. A Seção 4 apresenta a metaheurística GRASP, e suas hibridizações. Na Seção 5 são apresentados os experimentos, detalhes de implementação da hibridização e os resultados obtidos. Por último, na Seção 6 são relatadas as conclusões do trabalho.

2. Descrição do Problema Caixeiro Viajante

Suponha que um caixeiro tenha de visitar n cidades diferentes, iniciando e encerrando sua viagem na primeira cidade. A ordem de visita às cidades não importa e cada uma delas pode ir diretamente a qualquer outra. O problema do caixeiro viajante consiste em descobrir a rota que torna mínima a viagem total. Para exemplificar, têm-se quatro cidades A, B, C e D, uma rota que o caixeiro deve considerar poderia ser: saia de A e daí vá para B, dessa vá para C, e daí vá para D e então volte a A. Esta é uma possibilidade, mas existem outras seis rotas, citadas a seguir: ABCDA, ABDCA, ACBDA, ACDBA, ADBCA, ADCBA.

Em [13] há várias aplicações para o problema do caixeiro viajante. Um exemplo é o processo de perfuração de placas de circuitos impressos em sua confecção. Uma placa de circuito impresso pode conter centenas ou milhares de furos para soldagem de componentes eletrônicos. Como os furos podem ser de tamanhos diferentes, é necessária a troca da ferramenta. Essa troca de ferramenta é um processo lento. Para que a ferramenta não seja trocada várias vezes, é necessária uma

otimização perfurando primeiro todos os furos de mesmo diâmetro. Assim, esse processo pode ser visto como um problema do caixeiro viajante. Percorrendo-se o melhor caminho possível, economiza-se tempo, aumentando a produtividade do processo [14].

Uma segunda aplicação é a que ocorre na análise de estruturas de cristais na cristalografia por raios-x. A cristalografia analisa a disposição dos átomos em sólidos. Para isso um difratômetro obtêm informações da estrutura de um material cristalino medindo a intensidade dos raios-x refletidos do material, saindo de várias posições. Em alguns experimentos, o difratômetro deve realizar até 30.000 deslocamentos para fazer as medidas. Isso significa que pode haver um percurso com até 30.000 pontos de medida. Para minimizar o tempo gasto na mensuração, deve-se escolher um percurso mínimo entre os pontos de medida, o qual pode ser modelado através do problema do caixeiro viajante [14].

3. Fundamentação Teórica

Nesta seção são fundamentadas as tecnologias utilizadas no desenvolvimento do trabalho. Entre as tecnologias estão: método de busca local, metaheurísticas e estruturas de vizinhança utilizadas.

3.1 Hill Climbing

O método de busca local *Hill Climbing* (HC), é a base de muitos métodos heurísticos para problemas de otimização combinatória. Isoladamente, ele é um simples método iterativo de encontrar por boas soluções.

Considerado um algoritmo básico de busca local, o mesmo inicia com uma solução arbitrária e termina em um mínimo local onde não é mais possível qualquer tipo de melhoramento. Pode-se utilizar a busca local de *melhor movimento* que substitui a solução corrente por uma solução que melhora o custo após a busca em toda a vizinhança e a outra possibilidade é a busca local de *primeira melhora,* o qual aceita a primeira melhor solução quando ela é encontrada.

A complexidade computacional do HC depende do tamanho da vizinhança e também do tempo necessário para avaliar uma movimentação.

Os mínimos locais são os principais problemas na busca local. Entretanto estas soluções podem ser de boa qualidade, mas não necessariamente ótimas. Além disso, se a busca local acha um mínimo local não existe uma maneira obvia de melhorar a solução. As metaheurísticas estão tentando contornar isso através de critérios de diversificação.

3.2 Simulated Annealing

O Simulated Annealing (SA) [6] é uma metaheurística que simula um sistema de resfriamento aceitando movimentos de piora, segundo uma função probabilística.

O processo de busca da metaheurística SA é o resultado da combinação de duas etapas: caminhos aleatórios e melhoramento iterativo. Na primeira etapa, a tendência de melhorar o resultado é baixa, entretanto, faz com que o espaço de busca seja amplamente explorado. Esse comportamento é alterado, juntamente com a temperatura, fazendo o resultado convergir no final do processamento do algoritmo a um mínimo local. A escolha de um estado pior depende de duas variáveis: a diferença de custo entre o estado atual e o próximo estado e, a temperatura atual. Quanto maior a diferença de custo entre o estado atual e o próximo estado, menor é a possibilidade dele ser selecionado, e quanto maior a temperatura, maior a possibilidade de um estado pior ser escolhido [15,16].

3.2.1 Algoritmo

Na Figura 1 é apresento o algoritmo do SA.

```
Parâmetros:
         T0 (temperatura inicial),
         Tf (temperatura final),
         L(numero de iteracoes),
         α (redução de temperatura)

T ← T0 ; (temperatura atual recebe temperatura inicial)
S0 ← gera solução inicial; S ← S0; (solução atual)
S* ← S0; (melhor solução obtida até o momento)
enquanto T > Tf faça (temperatura alta)
       para cont ← 1 até L(T) faça (iterações para equilíbrio)
              S' ← seleciona uma solução vizinha de S
              Δcusto ← custo(S') -custo(S);
              se Δcusto < 0
                     S ← S';
              senão
                     Numero_Random ← Rand[0..1];
                     se Numero_Random < exp(-Δcusto/T)
                            S ← S';
              se (S < S*) S* ← S (guardar o melhor estado)
       fim para      T ← αT (reduz a temperatura)
fim-enquanto
Retorne S (melhor solução)
```

Figura 1 – Algoritmo SA

O algoritmo apresenta como valores iniciais os seguintes parâmetros: temperatura inicial, temperatura final, número de iterações para equilíbrio da temperatura e, taxa de redução de temperatura. A taxa de redução é aplicada à temperatura corrente após um número de iterações, ou seja, atingindo o equilíbrio na temperatura, aplica-se uma taxa de redução.
A escolha dos valores inicias como parâmetros são cruciais para o sucesso desse algoritmo. Essa escolha é feita através de testes de tentativa e erro observando-se o rendimento e comportamento do algoritmo. Se os valores iniciais estão bem calibrados aumentam as chances de localizar um estado ótimo global.

Após a configuração dos valores iniciais, o algoritmo gera um estado inicial (randomicamente ou heuristicamente construído).

O método inicia a partir de uma solução inicial e com temperatura alta, a qual é diminuída gradativamente até que alcance uma temperatura final baixa. A cada redução de temperatura é realizado um número fixo de iterações, que representa o número de iterações necessárias para o sistema atingir o equilíbrio térmico em uma determinada temperatura. Assim, a cada número de iterações a temperatura deve diminuir e uma solução S' vizinha a solução corrente S deve ser gerada e avaliada pelo valor da variação da função objetivo seguinte:

$$\Delta = f(S') - f(S)$$

Caso o movimento seja de melhora, então a solução S' torna-se a solução corrente. Caso contrário (delta <= 0) a solução S' possui um custo maior. Essa solução S' pode ser aceita através de um critério de probabilidade, onde T é a temperatura corrente:

$$p(\Delta E) = e^{-\Delta E/T}$$

Essa probabilidade permite movimentos ruins em altas temperaturas. A medida que a temperatura diminui, os movimentos ruins são reduzidos.

A temperatura T é diminuída a cada iteração do processo de busca, por isso, no início do processamento a possibilidade de serem escolhidos estados posteriores piores que o atual é grande,

no andamento da busca, as possibilidades vão diminuindo, com a tendência de terminar com buscas locais. Para o índice de resfriamento, são aconselhados valores entre 0 e 1, entretanto, os valores ideais para o índice pode modificar durante o processo de busca, com o objetivo de balancear entre a diversificação e a intensificação. Por exemplo, no inicio da busca, a Temperatura pode ser linearmente ou constantemente reduzido, convergindo para um mínimo local até o final da busca.

A metaheurística SA possui variantes que utilizam processos distintos de resfriamento e reaquecimento da temperatura, bem como a utilização de outras estruturas de vizinhança possibilitando constantes alternâncias entre diversificação e intensificação.

3.3 VNS

O VNS (*Variable Neighborhood Search*) [4,5] conhecido como método de busca em vizinhança variável consiste de uma metaheurística para resolver problemas de otimização global e combinatorial cuja idéia básica é troca sistemática da vizinhança dentro de uma busca local. O método procura explorar vizinhanças gradativamente mais distantes da solução corrente, priorizando a busca para uma nova solução somente se um movimento de melhora é realizado. O VNS não segue uma trajetória como os métodos de busca local, mas explora estruturas de vizinhanças a partir da solução corrente.

O VNS possui alguns princípios básicos, entre eles:

- Um ótimo local com relação a uma vizinhança não necessariamente corresponde a um ótimo local com relação a outra vizinhança;
- Um ótimo global corresponde a um ótimo local para todas as estruturas de vizinhança;
- Para muitos problemas, ótimos locais com relação a uma vizinhança são relativamente próximos.

O VNS pode ser utilizado de três diferentes maneiras: determinístico, estocástico e, ambos determinístico e estocástico. Nesse trabalho utilizo-se a versão determinística do VNS.

3.3.1 Determinístico

A maneira determinística do VNS é conhecida como Método de Descida em Vizinhança Variável (VND – *Variable Neighborhood Descent*). O VND é um método que explora o espaço de soluções através da troca sistemática de estruturas de vizinhança, aceitando apenas soluções de melhora da solução atual e retornando a primeira estrutura de quando uma solução melhor é encontrada. A Figura 2 apresenta o algoritmo para o VND.

```
s0 – uma solução inicial
r –  o número de estruturas diferentes de vizinhança;
s ← s0; {Solução corrente}
k ← 1; {Tipo de estrutura de vizinhança}
repita até nenhuma melhora
   enquanto (k ≤ r) faça
        Encontre o melhor vizinho s' ∈ N(k)(s);
        se f(s') < f(s)
               então
                      s ← s';
                      k ← 1;
               senão k ← k + 1;
        fim-se;
   fim-enquanto;
fim-repita;
Retorne s;
Fim VND;
```

Figura 2– Algoritmo VND

O algoritmo inicia-se a partir de uma solução inicial completa e um número de estruturas de vizinhança possível. Para cada estrutura de vizinhança é procurado o melhor vizinho. Se a solução encontrada for melhor a anterior, então o processo retorna a busca na primeira estrutura. Esse processo é interrompido quando for atingida a última estrutura de vizinhança e nenhuma melhora da solução corrente for possível.

3.4 Estruturas de Vizinhança

Uma estrutura de vizinhança pode ser definida pelos movimentos entre os componentes da solução, ou seja, as diferentes estruturas de vizinhança são obtidas a partir de uma operação chamada movimento. Seja N_k, (k= 1, ..., k_{max}), um conjunto finito de estruturas de vizinhanças e $N_k(x)$ o conjunto de soluções na k-ésima vizinhança da solução corrente x. Um método que utiliza estruturas de vizinhança inicializa com uma solução qualquer e a cada iteração gera um vizinho x' que é obtido a partir de uma operação chamada movimento.

Geralmente a busca local utiliza apenas uma estrutura de vizinhança (k_{max} = 1). Utilizando estruturas de vizinhança maiores que um, aumentam as chances de encontrar um local mínimo, do que utilizando apenas uma única estrutura.

A heurística k-opt [17] consiste na remoção de k-arcos de uma rota e substituí-los por outros k arcos, de modo a formar uma nova rota, com a finalidade de reduzir uma função objetivo (reduzir os custos). Movimentos k-opt somente são realizados se as novas ligações entre as cidades possuírem custo menor do que as removidas. Uma rota é k-ótima se não for mais possível efetuar trocas para melhorar a distância total viajada. Essa estrutura de vizinhança foi escolhida por ser indicada em problemas em que se deseja diminuir os custos em roteiros ou caminhos existentes e por ser popular na resolução do problema do caixeiro viajante. As Figuras 3 e 4, exemplificam respectivamente as heurísitcas 2-opt e 3-opt.

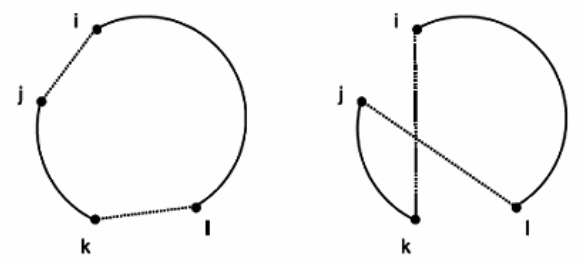

Figura 3 – Heurística 2-opt

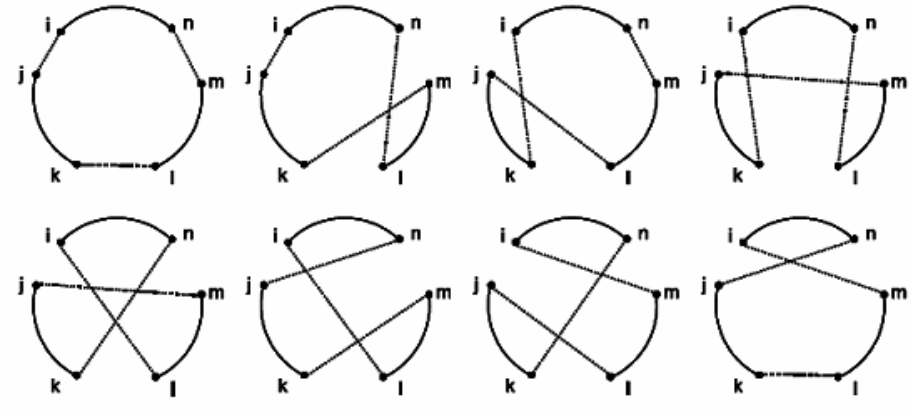

Figura 4 - Heurística 3-opt

De forma resumida, para k=2, o método testa trocas possíveis entre pares de arcos, refazendo as conexões quando houver uma melhoria na rota (Figura 3). A medida que aumenta o K, aumenta também o custo computacional. A Figura 5 mostra a aplicação do método 2-opt em uma solução inicial.

```
Solução Inicial: 1-6-4-2-3-5-1
Custo inicial = 20
Selecionar aresta: (4,2)
```

```
Soluções vizinhas 2-opt criadas:

1-) 1-6-4-3-2-5-1
(resulta da remoção das arestas (4,2) e (3,5) e da inclusão das arestas (3,4) e (2,5)) -
Custo = 19

2-) 1-4-2-6-3-2-5-1
(resulta da remoção das arestas (4,2) e (1,6) e da inclusão das arestas (1,4) e (2,5)) -
Custo = 19

3-) 1-6-4-5-3-2-1
(resulta da remoção das arestas (4,2) e (1,5) e da inclusão das arestas (4,5) e (1,2)) -
Custo = 20
```

```
Selecionar a solução criada com menor custo (solução 1 ou 2) e continuar o processo
escolhendo outras arestas até esgotar todas as possibilidades ....
```

Figura 5 – Exemplo da heurística 2-opt

4. GRASP

O GRASP (*Greedy Randomized Adaptive Search Procedures*) é uma metaheurística constituída por heurísticas construtivas e busca local [7,8]. Consiste de múltiplas aplicações de busca local, cada uma iniciando de uma solução diferente. As soluções iniciais são geradas por algum tipo de construção randômica gulosa ou algum esquema de perturbação.

A Figura 6 apresenta o algoritmo básico do GRASP. Observa-se no algoritmo que há duas fases: construção da solução e aplicação da busca local na solução construída.

Além disso, o algoritmo armazena a melhor ou as melhores soluções encontradas. Essa memória com as melhores soluções pode ser utilizada para encontrar outras soluções utilizando outras metaheurísticas como*: path-relink* [18] ou ILS.

```
Enquanto condiçãoTerminoNãoEncontrada faça
        s ← construçãoSolução();
        s'← aplicaBuscaLocal(s);
        memorizaMelhorSoluçãoEncontrada()
fim enquanto
```

Figura 6 – Algoritmo básico GRASP

4.1 Fase de Construção

Nessa fase uma solução é construída elemento a elemento. Inicialmente o elemento está em uma lista de candidatos (LC). Através de um fator α (*alfa*) é criada uma lista restrita de candidatos LCR que possui os melhores elementos de LC. O tamanho de LCR é determinado por:

$$CardinalidadeLCR = \alpha * CardinalidadeLC$$

Após a definição da LCR, seleciona-se um elemento da mesma para compor a solução. A seleção do elemento pode ser realizada aleatoriamente ou através de um critério guloso. Esses dois tipos de seleções provocam duas variações do GRASP conforme construção da solução: aleatório ou guloso. Após a adição do elemento na solução, o processo continua com a atualização de ambas as listas LC e LCR. O processo de construção é finalizado quando a cardinalidade de LC possuir valor zero.

O valor de α influencia na qualidade e diversidade da solução gerada na fase de construção. Valores baixos para α geram soluções gulosas de boa qualidade mas com pouca diversidade. Um valor alto para α, próximo a cardinalidade de LC leva a uma grande diversidade com soluções de qualidade inferior. Isso também influencia o processo de busca local, pois soluções de qualidade inferior tornam o processo de busca local mais lento [8]. Por exemplo com α valendo um, o melhor elemento da LCR seria adicionado, assim a fase de construção é uma heurística gulosa. Quando o valor de α é superior a 1 a construção é randômica. O valor de α pode ser constante, mas pode sofrer também alterações a cada iteração ou por meio de uma esquema aleatório ou adaptativo.

A Figura 7 apresenta o algoritmo para a fase de construção da solução inicial.

```
s ← ϕ  %  s que representa uma solução parcial nesse caso
α ← determinaTamanhoListaCandidatos
enquanto SoluçãoNãoCompleta faça
        RCLα ← gerarListaCandidatosRestritos(s)
        x ← selecionarElementoAleatório(RCLα)
        s ← s ∪ {x}
        atualizaFunçãoGulosa(s) % atualiza valores da heurística
fim enquanto
```

Figura 7 – Construção de Solução Randômica Gulosa

O algoritmo mostra que:

- a solução inicial é um conjunto vazio;
- o parâmetro α determina o tamanho da lista de candidatos;
- novos elementos são agregados a solução inicial;
- uma função gulosa avalia os elementos belo benefício imediato;
- melhores elementos formam uma lista de candidatos restritos em quantidade fixa ou de acordo com algum parâmetro α (aleatório ou guloso);
- um elemento da lista é escolhido aleatoriamente;
- a função gulosa é adaptada de acordo com a solução parcial;
- termina com uma solução aceitável ao problema.

4.2 Fase de Busca Local

A segunda fase do GRASP ou fase de busca local consiste em refinar a solução encontrada na primeira fase aplicando um método de busca local. Isso corresponde a uma intensificação na solução encontrada explorando regiões vizinhas através da busca local para encontrar um ótimo local.

A busca local pode ser sofisticada, entretanto não se deve esquecer do diferencial do GRASP que é amostrar o espaço com gerações rápidas. Quanto melhor for a qualidade da solução gerada na primeira fase, maior será a velocidade para encontrar um ótimo local pela fase de busca local.

Os algoritmos a serem utilizados nesta segunda fase podem ser básicos como o *Hill-Climbing* ou metaheurísticas mais avançadas como Tabu, SA e VND.

É importante salientar que o GRASP não faz uso de históricos no processo de busca. É possível armazenar uma ou várias soluções sendo as melhores até o momento. Entretanto, o GRASP é simples, rápido e pode ser facilmente integrado com outras técnicas de busca.

5. Experimentos e Resultados Obtidos

Neste capítulo apresenta-se as implementações do GRASP e resultados obtidos na resolução do problema do Caixeiro Viajante Simétrico.

5.1 Detalhes de Implementação

O GRASP é uma metaheurística com duas fases. Na primeira fase as soluções são construídas utilizando um procedimento aleatório guloso. Para a primeira fase é possível determinar a utilização de um procedimento mais aleatório ou guloso na geração das soluções iniciais determinando o tamanho da LCR e o parâmetro α. A implementação realizada possibilita a escolha do parâmetro α que influencia diretamente na LCR. O tamanho da LC foi determinado pelo número de cidades existentes na instância. Por exemplo na instancia dantzig42 há 42 cidades conectadas, portanto a LC possui tamanho 42.

Na segunda fase o GRASP procura refinar a solução inicialmente construída na primeira fase aplicando algum método de busca local. Assim o GRASP sofreu diversas hibridizações nesta fase, entre elas: GRASP com *Hill-Climbing* (GRASP/HC), GRASP com *Simulated Annealing* (GRAS/SA), e GRASP com VNS Descendente (GRASP/VND). Para as duas primeiras hibridizações (GRASP/HC e GRASP/SA) foi utilizada uma única estrutura de vizinhança através de trocas aleatórias (troca de cidades através de inserções e remoções aleatórias). Essa estrutura foi composta por movimentos aleatórios entre as cidades. Para a versão de hibridização do GRASP com VNS Descendente (GRASP/VND), utilizou-se duas outras estruturas de vizinhança denominadas de 2-opt e 3-opt, devido ao fato do VNS realizar trocas sistemáticas de vizinhança,

5.2 Instâncias utilizadas no experimentos

Para validar os métodos, foram utilizadas quatro instâncias encontradas e disponíveis em [19]. Essas instâncias correspondem a tabelas com distância simétrica entre as cidades. As instâncias são nomeadas na seqüência: Dantzig42 (42 cidades); Fri26 (26 cidades); gr48 (48 cidades) e hk48 (48 cidades). As melhores soluções para cada solução são: 699 para Dantzig42, 937 para Fri26, 5046 para gr48 e 11461 para hk48. Esses custos foram utilizados para comparar os resultados dos experimentos realizados com as diversas hibridizações.

5.3 Determinação dos parâmetros

O único parâmetro que deve ser calibrado no GRASP é o valor α. Esse valor foi calibrado buscando aplicar ao GRASP duas características na primeira fase: puramente guloso ou aleatório. No puramente guloso o α recebeu valor 1. No aleatório o valor de α utilizado corresponde a 40% do tamanho da LC. A LC corresponde ao número máximo de cidades (conexões entre as cidades). O α foi utilizado apenas para determinar o tamanho da LCR.

Na segunda fase há a definição de parâmetros quando da utilização do método de busca local SA. Os parâmetros de SA foram definidos através de várias execuções e verificando qual é o melhor valor para: temperatura inicial, temperatura final, número de iterações para equilíbrio da temperatura e taxa de redução da temperatura. Os seguintes valores para o SA foram utilizados:

- Temperatura Inicial: 30;
- Temperatura Final: 0.1;
- Taxa de Redução de Temperatura: 0,2%;
- Número de iterações para equilíbrio da temperatura: 40% do tamanho da matriz da instância.

Outro parâmetro utilizado nos teste foi o número de iterações ou número de execuções do GRASP. Esse parâmetro indica o número de soluções construídas e aplicação da busca local nas soluções.

Nesse trabalho foi utilizado o número de iterações como o número de cidades existentes nas instâncias. A justificativa foi construir *n* soluções cada uma iniciando com uma cidade diferente. Por exemplo, para a instância dantzig42 com 42 cidades conectadas, o GRASP executou 42 vezes para cada versão de hibridização e com dois valores distintos para alfa (aleatório ou puramente guloso).

5.4 Metodologia de Testes

Nos testes o objetivo foi tornar a fase inicial do GRASP puramente gulosa ou aleatória. Essa primeira fase possui portanto 2 variações.

Na segunda fase o GRASP recebeu diferentes métodos de busca local. Esses métodos deram origem a 3 diferentes hibridização: GRASP/HC, GRASP/SA, GRASP/VND.

Para cada hibridização o GRASP gerou *n* cidades, onde *n* corresponde ao número de cidades da instância.

Portanto foram executados os seguintes testes para as 4 instâncias:

GRASP/HC puramente guloso, GRASP/HC Aleatório, GRASP/SA puramente guloso, GRASP/SA aleatório, GRASP/VND puramente guloso e GRASP/VND aleatório.

Nesses testes foram calculados a média dos custos e tempo encontrados. Além disso, o melhor resultado foi comparado aos melhores resultados obtidos em [19].

5.5 Resultados obtidos

A seguir a apresentação dos resultados obtidos e comparação dos mesmos.

A Tabela 1 apresenta os mínimos locais para cada instância. Esses valores foram retirados de [19] e são utilizados para comparação com os resultados obtidos.

Instâncias	Melhor Custo
dantzig42	699
fri26	937
gr48	5046
hk48	11461

Tabela 1 – Mínimos Locais (Fonte: [19]).

A Tabela 2 apresenta os melhores custos encontrados na execução das 3 hibridizações onde a fase de construção da solução do GRASP foi totalmente gulosa (α = 1). A hibridização GRASP/VND conseguiu os melhores custos para dantzig42 e fri26. Para as demais instâncias os resultados ficaram próximos ao ótimo local.

GRASP/HC e GRASP/SA obtiveram resultados semelhantes, porém, a Tabela 3, que apresenta o tempo (em segundos) para obtenção dos resultados, mostrou que GRASP/SA possui uma complexidade computacional muito superior as outras hibridizações.

Instâncias	GRASP/HC	GRASP/SA	GRASP/VND
Dantzig 42	863	863	**699**
fri26	955	955	**937**
gr48	5781	5779	5093
hk48	12110	12006	11470

Tabela 2 – Melhores custos com fase de construção gulosa para o GRASP

Instâncias	GRASP/HC	GRASP/SA	GRASP/VND
dantzig42	0,01	1,94	1,42
fri26	0,001	2,65	0,10
gr48	0,01	12,52	2,85
hk48	0,02	36,40	1,60

Tabela 3 – Tempo em segundos com fase de construção puramente gulosa

Através da Tabela 3, observa-se também, que GRASP/HC foi a hibridização que atingiu o resultado mais rapidamente. Inclusive seus resultados são muito semelhantes ao GRASP/SA e um pouco inferiores a GRASP/VND.

Lembrando que para cada iteração, o GRASP iniciou com diferentes cidades, a Tabela 4 apresenta a média dos custos encontrados em todas as execuções. Mesmo iniciando com diferentes cidades, GRASP/VND conseguiu atingir os melhores custos em várias execuções. A média para todas as instâncias ficou muito próxima aos melhores custos.

Instâncias	GRASP/HC	GRASP/SA	GRASP/VND
Dantzig42	918	916	710
Fri26	1057	1059	954
Gr48	6228	6211	5147
Hk48	13263	12116	11571

Tabela 4 – Média dos custos com fase de construção puramente gulosa

A Tabela 5 mostra os resultados onde a fase de construção do GRASP é parcialmente aleatória, utilizando 40% das melhores opções de cidades da LCR. Em comparação com a Tabela 2, o GRASP/VND conseguiu o mesmo desempenho. Inclusive nesse caso, obteve os melhores custos para 3 instâncias e a quarta instância ficou muito próximo do ótimo global.

GRASP/HC e GRASP/SA obtiveram resultados inferiores em relação aos testes anteriores onde a fase de construção do GRASP é puramente gulosa.

Instâncias	GRASP/HC	GRASP/SA	GRASP/VND
Dantzig42	1027	938	699
Fri26	1345	1341	937
Gr48	7828	6712	5046
Hk48	17012	16504	11470

Tabela 5 – Melhores custos com fase de construção aleatória

Na Tabela 6 são apresentados os tempos para obtenção dos melhores resultados. Novamente o GRASP/SA teve o pior custo computacional e GRASP/HC a menor complexidade computacional.

Instâncias	GRASP/HC	GRASP/SA	GRASP/VND
Dantzig 42	0,001	3,30	2,94
fri26	0,001	1,81	0,17
gr48	0,01	32,60	6,24
hk48	0,01	52,34	5,34

Tabela 6 – Tempo em segundos com fase de construção aleatória

A Tabela 7 apresenta a média de todas as execuções realizadas para a fase de construção gulosa no GRASP. GRASP/VND manteve o bom desempenho encontrando em várias execuções os melhores custos globais.

	GRASP/HC	GRASP/SA	GRASP/VND
Dantzig 42	1919	1892	714
Fri26	1765	1862	958
Gr48	13771	11920	5151
Hk48	30164	28658	11785

Tabela 7 – Média dos custos com fase de construção aleatória

A Tabela 8 apresenta um resumo e uma comparação dos melhores custos encontrados para cada instância, comparando com a fase de construção do GRASP (Guloso/Aleatório). Para cada melhor resultado foi calculada a porcentagem de proximidade com o melhor custo. GRASP/VND conseguiu os melhores custos para três instâncias e para a instância hk48 ficou muito próximo (0,08%). A Tabela 8 também apresenta uma proximidade entre GRASP/HC e GRASP/AS, porém os melhores resultados são obtidos onde a primeira fase do GRASP é gulosa.

A Tabela 9 apresenta as mesmas informações apresentadas anteriormente, porém utilizando os valores médios

	GRASP/HC			GRASP/SA			GRASP/VND		
	Custo	**%**	**GRASP**	**Custo**	**%**	**GRASP**	**Custo**	**%**	**GRASP**
Dantzig 42	863	23,46%	G	863	23,46%	G	**699**	0,00%	G
fri26	955	1,92%	G	955	1,92%	G	**937**	0,00%	G
gr48	5781	14,57%	G	5779	14,53%	G	**5046**	0,00%	A
hk48	12110	5,66%	G	12006	4,76%	G	11470	0,08%	G/A

Tabela 8 – Comparativo com melhores resultados

	GRASP/HC			GRASP/SA			GRASP/VND		
	Custo	**%**	**GRASP**	**Custo**	**%**	**GRASP**	**Custo**	**%**	**GRASP**
Dantzig 42	918	31,33%	G	916	31,04%	G	710	1,57%	G
fri26	1057	12,81%	G	1059	13,02%	G	954	1,81%	G
gr48	6228	23,42%	G	6211	23,09%	G	5147	2,00%	G
hk48	13263	15,72%	G	12116	5,72%	G	11571	0,96%	G

Tabel 9 – Comparativo com valores médios de cada execução

6. Conclusões

O objetivo deste trabalho foi implementar o GRASP e hibridizar o mesmo com outras metaheurísticas na resolução do problema do caixeiro viajante simétrico. Conclui-se que a hibridização é um excelente mecanismo para equilibrar os componentes de diversificação e intensificação. Esse equilíbrio torna a metaheurística inteligente para conseguir explorar regiões ainda não visitadas e também para melhorar a pesquisa próxima a um ótimo local.

Os resultados obtidos mostraram que a hibridização do GRASP obteve ótimos resultados principalmente na utilização do VND devido ao mesmo utilizar estruturas de vizinhança k-opt (2-opt e 3-op) adequadas para a resolução do problema do caixeiro viajante. Além disso, o GRASP mostrou que a primeira fase é muito importante, pois mesmo o método HC obteve bons resultados a partir de soluções de qualidade encontradas na primeira fase quando a mesma é gulosa. Portanto, a primeira fase do GRASP deve ser capaz de testar regiões mais promissoras do espaço de busca.

Além da importância apresentada pela hibridização das metaheurísticas, relata-se a resolução do problema do caixeiro viajante que apesar de ser um problema simples comparado a outros, é muito utilizado em variações para a resolução de outros problemas que se assemelham ao caixeiro viajante.

Referências

[1] Blum C.; Roli A. (2003) **Metaheuristics in combinatorial optimization: Overview and conceptual comparison**. ACM Comput. Surv., Vol 35, N. 3, September , pp. 268-308.

[2] Glover. F. (1996) **Tabu search and adaptive memory programing – Advances, applications and challenges.** In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, Interfaces in Computer Science and Operations Research, pages 1–75. Kluwer.

[3] Lourenço H.R.; Martin O.C. and T. St¨utzle. (2002) **Iterated Local Search.** In F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics, pages 321–353. Kluwer, Boston.

[4] Mladenovic N. and Hansen. P. (1997) **Variable neighborhood search.** *Computers &Operations Research*, 24:1097–1100.

[5] Hansen P. and Mladenovi´c N. (1999) **An introduction to variable neighborhood search.**In S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 433– 458. Kluwer Academic Publishers, Boston, MA.

[6] Kirkpatrick, S., C. D. Gelatt Jr., M. P. Vecchi. (1983) **Optimization by Simulated Annealing**,*Science*, 220, 4598, 671-680.

[7] Feo T.A. and Resende M.G.C. (1989) **A probabilistic heuristic for a computationally difficult set covering problem.** Operations Research Letters, 8:67–71.

[8] Feo T.A. and Resende M.G.C. (1995) **Greedy randomized adaptive search procedures.** Journal of Global Optimization, 6:109–133.

[9] Holland, J. H. (1975) **Adaption in natural and artificial systems**. The University of Michigan Press, Ann Harbor, MI.

[10] Goldberg, D. E. (1989) **Genetic Algorithms in Search, Optimization and Machine Learning**. Addison Wesley, Reading, MA.

[11] Dorigo, M. (1992) **Optimization, learning and natural algorithms *(in italian)*.** Ph.D. thesis, DEI, Politecnico di Milano, Italy. pp. 140.

[12] Kennedy, J.; Eberhart, R. C. (1995). **Particle swarm optimization**. *Proceedings of the IEEE InternationalConference on Neural Networks IV*, Perth, Australia, pp. 1942-1948.

[13] Reinelt, G. (1994) **The Traveling Salesman: Computational Solutions for TSP Applications**, Lecture Notes in Computer Science, 840, Springer-Verlag.

[14] Bertini, L. Segundo Trabalho Prático de PAA: **O Problema do Caixeiro Viajante**. Disponível em: http://homepages.dcc.ufmg.br/~nivio/cursos/pa03/tp2/tp22/tp22.html. Acesso em: 30/05/2007.

[15] Aarts E. e Korst J. (1993) **Simulated Annealing and BoltzmannMachine**, John Wiley, 1989. DOWSLAND, K.A. *Simulated Annealing*, In Reeves, C.R. (ed), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, 20-69.

[16] Cerny, V. (1985) **Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm**", *J. Opt. Theory Appl.*, 45, 1, 41-51.

[17] Lin, S. e B. W. Kernighan (1973) **An Effective Heuristic Algorithm for the Traveling Salesman Problem.***Operations Research*, v. 21, p. 498-516.

[18] Resende M.G.C and Ribeiro C.C. (2005) **GRASP with path-relinking: Recent advances and applications.** In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, Metaheuristics: Progress as Real Problem Solvers, pages 29–63. Springer.

[19] TSPLIB. Disponível em: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html.Acesso em: 29/05/2007.