# UMA METAHEURÍSTICA BASEADA EM CONSTRUÇÃO DE VOCÁBULOS PARA O PROBLEMA DO CAIXEIRO VIAJANTE ASSIMÉTRICO

**Allison da Costa Batista Guedes**
Departamento de Informática e Matemática Aplicada – Programa de Educação Tutorial
Universidade Federal do Rio Grande do Norte
Campus Universitário, s/n, Natal/RN, CEP: 59072-970
allison@lcc.ufrn.br

**Dario José Aloise**
Departamento de Informática e Matemática Aplicada – Programa de Educação Tutorial
Universidade Federal do Rio Grande do Norte
Campus Universitário, s/n, Natal/RN, CEP: 59072-970
dario@dimap.ufrn.br

**Resumo**

O presente trabalho desenvolve duas implementações distintas da técnica de otimização conhecida como Vocabulary Building, idealizada por Fred Glover no início da década de 90, e propõe um algoritmo memético que as utiliza para a resolução do Problema do Caixeiro Viajante Assimétrico. Desde a sua apresentação, a técnica de Vocabulary Building tem sido mencionada em alguns artigos, mas ainda é um conceito pouco explorado. O procedimento híbrido desenvolvido mostrou-se, experimentalmente, muito interessante na obtenção de bons resultados nas instâncias assimétricas da TSPLIB.

**Palavras-Chaves:** Metaheurística; Vocabulary Building; Problema do Caixeiro Viajante Assimétrico.

**Abstract**

The present work develops two distinct implementations of the optimization technique known as Vocabulary Building, idealized by Fred Glover in the early 90's, and proposes a memetic algorithm that uses them for the resolution of the Asymmetric Traveling Salesman Problem. Since its presentation, the Vocabulary Building technique has been mentioned in some papers, but it is still a concept that has been little explored. The hybrid procedure developed has experimentally shown to be very interesting in obtaining good results on asymmetric instances from the TSPLIB.

**Keywords:** Metaheuristic; Vocabulary Building; Asymmetric Traveling Salesman Problem.

## 1. INTRODUÇÃO

O Problema do Caixeiro Viajante (PCV) é um dos mais conhecidos na área de otimização combinatória ([15]; [21]). Pela sua importância prática e dificuldade de solução, ele tem atraído a atenção de muitos pesquisadores das áreas da matemática, da engenharia e da ciência da computação ([22]; [20]; [13]).

Embora possa ser descrito de uma maneira simples, o PCV pertence à classe dos problemas NP-difíceis ([2]), ou seja, não pode ser resolvido por um algoritmo polinomial, a menos que a classe P = NP, tornando-o intratável para obtenção da solução por métodos exatos para problemas de grande porte. Com isso, abordagens exatas e aproximativas (heurísticas/metaheurísticas) têm sido buscadas para a resolução de instâncias de grande porte desse problema. Como referências para avanços na abordagem das metaheurísticas,

destacamos as referências: [24], [18], [8], [14], [3] e [16].

O presente trabalho apresenta duas propostas de aplicações da técnica conhecida como *Construção de Vocábulos* (Vocabulary Building – VB) em metaheurísticas populacionais para a resolução de problemas de otimização combinatória NP-difícil. Especificamente, essas propostas foram inseridas na metaheurística Algoritmo Memético e implementadas para o Problema do Caixeiro Viajante Assimétrico.

## 2. VOCABULARY BUILDING

### 2.1. HISTÓRICO

A idéia de utilização de Vocabulary Building foi idealizada por Fred Glover, dentro do contexto de Path Relinking ([4]). Em [5], essa técnica foi proposta como uma estratégia a ser implementada dentro da Busca Tabu. Em 1995, surgiram dois trabalhos cujos conteúdos possuem ligação com as idéias relativas ao Vocabulary Building ([24] e [19]) aplicadas a problemas de roteamento de veículos. Em 1997, a técnica é mais uma vez mencionada em um trabalho que trata de *Scatter Search* e *Path Relinking* ([6]). A partir de 1997, surgiram outras publicações que abrangem o Vocabulary Building, sendo quase todas elas do próprio Fred Glover, como: [7], [25], [8], [9] e [10].

Dentre os trabalhos citados acima, o único que trata de uma aplicação prática real com implementação computacional da técnica Vocabulary Building é [25], o qual aborda o Problema de Seqüenciamento de Linhas de Montagem Multi-Modelo. O Vocabulary Building é, portanto, uma técnica de otimização ainda pouco explorada. Isso leva a crer que qualquer utilização dessa técnica produzirá resultados originais, o que motivou a pesquisa realizada nesse trabalho.

### 2.2. IDÉIA GERAL

O Vocabulary Building pode ser considerado uma variação de *Path Relinking*, tendo em vista que também utiliza o conceito de "vizinhanças construtivas" para gerar novas soluções a partir de diversas outras já existentes, tal como explicado em [9]. A diferença está na possibilidade de se utilizar também soluções parciais em conjunto com soluções completas.

A idéia central da técnica consiste em tentar identificar boas soluções parciais, ou seja, que possam originar soluções completas de alta qualidade (*fitness*). Através dessas configurações parciais, pode-se prosseguir com a busca pelo ótimo global. Uma boa solução parcial pode ser, por exemplo, uma configuração que esteja presente em várias soluções-elite.

Para que seja possível identificar boas configurações parciais, é preciso que se encontrem bons elementos dessas. Dá-se o nome de "Vocábulo" a um elemento identificado como sendo importante para a obtenção de soluções de boa qualidade (por exemplo, uma determinada aresta da rota do Caixeiro Viajante). A partir desses vocábulos, é possível criar combinações mais complexas e úteis.

É interessante citar que o Vocabulary Building opera tanto sobre elementos primitivos como elementos compostos ([9]). Exemplificando para o caso do PCV, a técnica trabalharia tanto sobre cidades (elementos primitivos) como sobre trechos contendo várias cidades (elementos compostos - vocábulos).

É necessária a utilização de memória para que o processo de busca esteja informado a respeito de configurações parciais que tenham se mostrado interessantes. Para isso, o método necessita de uma estrutura para armazenar os vocábulos identificados como úteis. Esse conjunto de vocábulos recebe o nome de "Pool de vocábulos". O *Pool de vocábulos* é uma forma de memória adaptativa, pois é dinâmico, ou seja, ele se modifica no decorrer da busca. A cada momento, soluções parciais distintas podem ser de diferente atratividade. Com isso, é interessante que o *Pool* mantenha somente as que forem mais promissoras.

A idéia de Vocabulary Building é bem geral e pode ser utilizada para diversos

problemas e de diversos modos. Neste trabalho, idealizou-se duas formas de implementações das idéias do Vocabulary Building para o PCVA. Na primeira delas, é utilizado um pool de vocábulos para manter trechos de rotas que são avaliados de acordo com a sua boa ou má influência durante a busca. Na segunda forma, verificam-se as arestas presentes em todas as rotas de um determinado conjunto de soluções e faz-se a condensação de trechos contíguos. A seguir, serão vistos os detalhes de implementação dos procedimentos utilizados nesse trabalho.

## 3. IMPLEMENTAÇÕES

### 3.1. IMPLEMENTAÇÃO 1

Nesta primeira forma de Vocabulary Building implementada, trabalha-se com um pool de vocábulos e um conjunto de soluções completas. Esse pool contém os vocábulos que serão utilizados (e modificados) durante o processo de busca. Nas figuras 1 e 2, tem-se uma ilustração de uma possível solução completa e de um possível vocábulo.


Figura 1: representação de uma solução completa

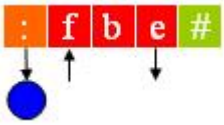


Figura 2: representação de um vocábulo

O trecho marcado por {f, b, e} indica uma seqüência de cidades que será utilizada para tentar melhorar alguma solução completa. O campo "#" indica uma espécie de *fitness* do vocábulo, que indica o quão bem ele vem atuando na busca. A partir desse valor são tomadas decisões sobre o que fazer com o vocábulo. O campo marcado com ":" indica o histórico de construção do vocábulo; o círculo apontado por esse campo foi denominado de "árvore de quebra". Essa estrutura é necessária devido ao fato de que os vocábulos podem ser combinados entre si, na tentativa de gerar trechos mais úteis. A idéia básica é iniciar com um pool de vocábulos gerado de forma aleatória. A partir daí, passa-se a realizar a aplicação do pool nas soluções-elite e vai se verificando quais trechos produzem melhoras e quais não produzem. A intenção é manter os vocábulos de alta qualidade e alterar/descartar os de baixa qualidade.

#### 3.1.1. Aplicação do Pool de Vocábulos

A aplicação do Pool de Vocábulos é realizada sobre uma solução completa. Essa operação tem por objetivo verificar se é possível melhorar a solução em questão, ao se colocar nela um dos trechos disponíveis no pool. Abaixo, segue o algoritmo desse procedimento.

```
procedimento aplicar_pool_em_solucao(P, S);
        Entrada: Pool de vocábulos P, Solução completa S;
        Saída: Melhor solução encontrada, das inserções realizadas;
1.      melhor.fitness ← ∞;
2.      para cada vocábulo V em P faça
3.              auxiliar ← S;
4.              atual ← inserir_vocabulo_em_solucao(V, auxiliar, P);
```

Figura 3: procedimento de aplicação de um pool de vocábulos em uma solução completa

No algoritmo acima, faz-se a inserção de cada vocábulo do Pool, separadamente, na solução dada como entrada. Retorna-se a melhor solução encontrada como resultado de uma inserção. É interessante observar que a inserção é realizada e mantida na solução, mesmo que esta não tenha uma melhora em sua *fitness*.

Abaixo, tem-se o algoritmo do mecanismo de inserção de um vocábulo em uma solução.

```
procedimento inserir_vocabulo_em_solucao(V, S, P);
        Entrada: Vocábulo V, Solução completa S, Pool de vocábulos P;
        Saída: Solução completa após V ser inserido em S;
1.      res ← transcrever(V, S);
2.      se res.fitness < S.fitness então
3.              V.contador ← V.contador + 1;
4.      senão
5.              V.contador ← V.contador – 1;
6.      fim-se;
7.      se V.contador ≤ 0 então
8.              se V.arvore_quebra = NULO então // Vocábulo elementar
9.                      se rand() < PROB então // rand(), PROB ∈ [0..1]
10.                             alterar_vocabulo(V, S);
11.                             V.contador ← CONT_INI;
12.                     senão
13.                             V ← novo_vocabulo_aleatorio();
14.                             V.contador ← 0;
15.                     fim-se;
16.             senão
17.                     desnaturar_vocabulo(V, X, Y);
18.                     X.contador ← CONT_MAX;
19.                     Y.contador ← CONT_MAX;
20.                     P ← P – {V};
21.                     P ← P ∪ {X} ∪ {Y};
22.             fim-se;
23.     fim-se;
24.     se V.contador ≥ CONT_MAX então
25.             combinar_vocabulo(V, P);
26.             V.contador ← CONT_INI;
27.     fim-se;
28.     retorna res;
fim-procedimento;
```

Figura 4: procedimento de inserção de um vocábulo em uma solução completa

O primeiro passo do procedimento acima é realizar a transcrição do trecho contido no vocábulo para a solução S dada. Isso é feito da seguinte forma: retira-se de S as cidades que estão contidas em V, obtendo-se uma rota incompleta S'. Então, verifica-se qual a melhor

posição para a introdução do trecho contido no vocábulo dentro de S'. Faz-se a operação de inserção nesse ponto e retorna-se a nova rota completa obtida. Essa operação é feita de forma que a ordem das cidades seja mantida, com exceção das que estão presentes no vocábulo. Esse procedimento é ilustrado na figura abaixo.

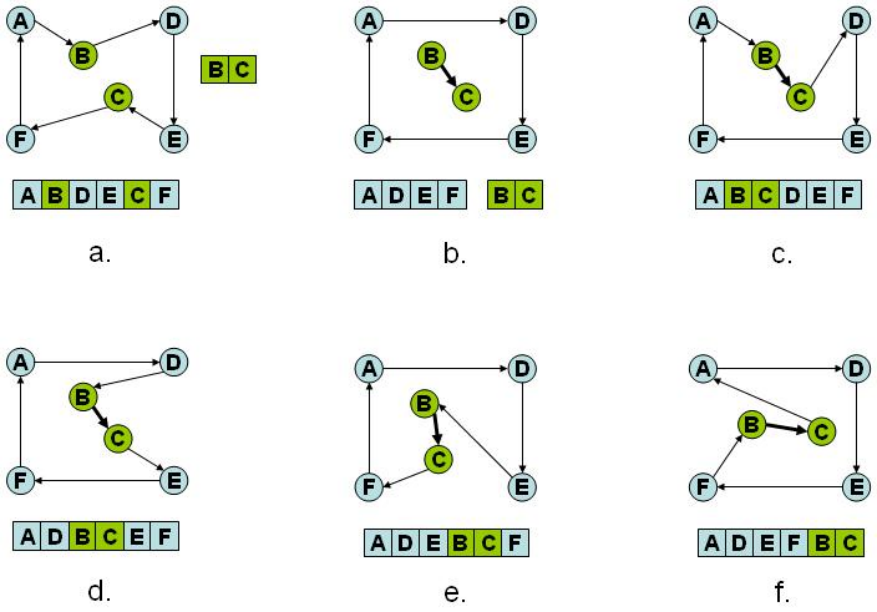


Figura 5: inserção do vocábulo {b, c} na solução completa {a, b, d, e, c, f}

Na parte **a**, temos uma rota completa e o vocabulário a ser inserido (vetor de 2 posições). Em **b**, verifica-se a estrutura obtida após a retirada das cidades contidas no vocábulo. De **c** a **f**, tem-se as tentativas de introdução do vocábulo na rota incompleta obtida. Após verificar todas as posições possíveis, o procedimento retorna a solução que possui melhor *fitness*.

Após realizar a transcrição, o procedimento *inserir_vocabulo_em_solucao* verifica se a operação causou uma melhora na solução dada. Se houve melhora, incrementa-se o contador do vocábulo; caso contrário, seu contador é decrementado. Feito isso, é necessário verificar se o contador do vocábulo chegou a um valor muito baixo ou muito alto: no segundo caso, realiza-se a combinação desse vocábulo com algum outro do Pool. Para o primeiro caso, é necessário saber se o vocábulo é composto (originado por dois outros vocábulos) ou elementar (representando apenas um aresta). Se for um vocábulo composto, ele é retirado do pool e quebrado em dois. As partes resultantes da quebra são devolvidas ao pool, com seus contadores reinicializados no valor máximo (é necessário que haja um valor máximo, pois um vocábulo pode se mostrar interessante em um momento da busca e, logo depois, se tornar incapaz de realizar melhorias). Para o caso de o vocábulo ser elementar (árvore de quebra não guarda nenhuma informação), gera-se um número aleatório e, dependendo de uma probabilidade PROB, pode-se realizar a alteração ou o descarte do vocábulo.

A seguir, tem-se uma figura que ilustra o processo de alteração do vocábulo. Supõe-se que o vocábulo já efetuou o processo de inserção e o trecho contido no vocábulo também está presente na solução completa.

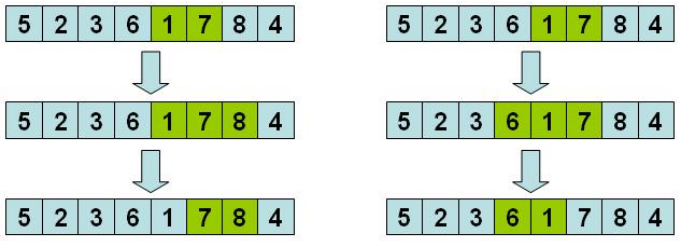


Figura 6: alteração do vocábulo {1, 7} utilizando como referência a solução completa {5, 2, 3, 6, 1, 7, 8, 4}

A alteração do vocábulo se dá de acordo com o vetor-solução em que ele foi inserido. Basicamente, o vocábulo descarta uma de suas partes e copia uma parte da solução para si. Na figura acima, temos dois exemplos de alteração: em **a**, o trecho {1,7} passa a ser {7, 8}; em **b**, o mesmo trecho se altera para {6, 1}. É escolhida sempre a alteração que resultar na menor relação *peso da aresta inserida – peso da aresta retirada*.

O processo de combinação ocorre quando um vocábulo X atinge um alto valor em seu contador, indicando que ele tem realizado melhora no conjunto de soluções vigente. Com isso, procura-se um vocábulo Y que seja compatível e que tenha o maior contador possível. Dois vocábulos são considerados compatíveis quando não possuem nenhuma cidade em comum ou quando o início de um coincide com o final do outro; dessa forma, não existe a possibilidade de haver vocábulos com cidades duplicadas. Abaixo, segue uma ilustração do processo de combinação, que mostra a construção da árvore de quebra de um vocábulo.

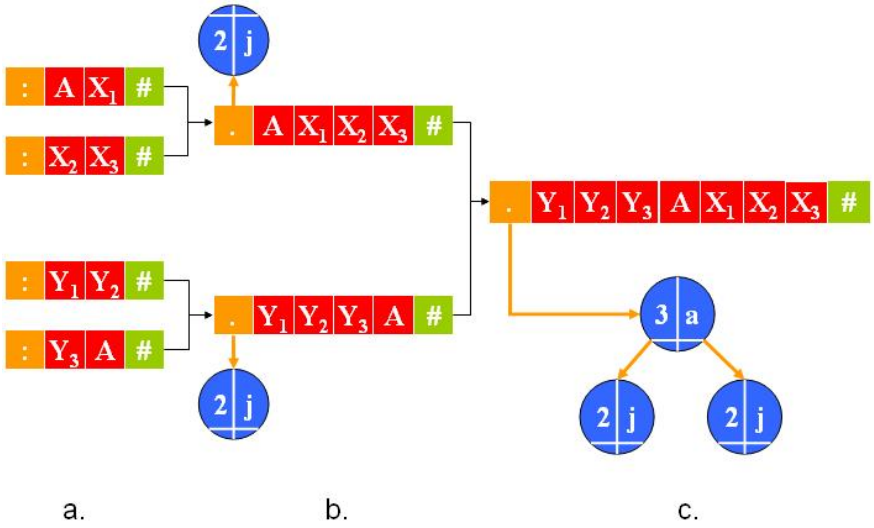


Figura 7: combinação de vocábulos

Em **a**, tem-se quatro vocábulos distintos. De **a** para **b**, ocorre a combinação dos dois vocábulos de cima e dos dois vocábulos de baixo. A árvore de quebra retém a informação necessária para realizar a separação desses vocábulos: o número **2** indica que o segundo vocábulo se inicia a partir da posição número 2 (terceira posição) do vetor-solução; a letra **j** (de Justaposição) indica que os vocábulos originais não possuíam nenhuma cidade em comum. De **b** para **c**, ocorre a combinação dos dois vocábulos recém-gerados. Como eles possuem uma cidade da extremidade em comum, esse fato está indicado pela letra **a** (de Aglutinação) da árvore de quebra do vocábulo resultante. O numeral do nó-raiz, mais uma vez, indica a posição a partir da qual o vetor armazena o segundo vocábulo da combinação. Com a árvore de quebra à disposição, é possível realizar o processo inverso: esse processo é chamado de desnaturação, no procedimento *inserir_vocabulo_em_solucao*.

### 3.2. IMPLEMENTAÇÃO 2

O Path Relinking (PR) é uma técnica de otimização que opera sobre um conjunto de soluções (método evolucionário). PR emprega procedimentos de combinação para gerar novas soluções a partir de outras pré-existentes. O Vocabulary Building opera de maneira similar, diferindo no fato de que pode manipular tanto soluções completas como soluções parciais: a idéia é que se possa utilizar trechos de soluções, combinando-os entre si e com vetores completos.

Dois dos principais aspectos que caracterizam uma implementação de PR ou VB são:

1. O método de combinação utilizado; e
2. O método pelo qual os resultados das combinações são explorados

Como as técnicas de PR e VB são flexíveis, é possível utilizar qualquer regra de combinação que se considere adequada. Mais de uma regra pode ser utilizada; contudo, a estratégia padrão consiste em utilizar somente uma. Quando mais de uma regra é utilizada, elas são geralmente aplicadas de forma isolada uma da outra.

Em [7], Glover cita um dos primeiros trabalhos envolvendo estratégias de solução baseadas em combinação de regras de decisões ([11]). Comenta também que a abordagem utilizada nesse último trabalho foi motivada pela suposição de que a atratividade de diversas escolhas é capturada de forma diferente por regras distintas. Supõe também que essa informação pode ser melhor explorada por meio de mecanismos de combinação de regras do que pela utilização de regras isoladas.

A segunda implementação do Vocabulary Building apresentada nesse trabalho utiliza a noção de contração de vértices e faz uso da idéia proposta por Glover. Apresenta-se um mecanismo de combinação que se utiliza de um conjunto de soluções para construir vocábulos e, posteriormente, realizar um processo de montagem para se obter soluções completas (nesse caso, ciclos Hamiltonianos). Como regras básicas, utiliza-se heurísticas construtivas.

O mecanismo proposto funciona da seguinte forma:

1. Cada heurística construtiva produz um determinado número de soluções;
2. Considerando-se todas as soluções geradas, identificam-se as arestas que estão presentes em todas elas;
3. Com esse conjunto de arestas em mãos, identificam-se e concatenam-se trechos que são formados por arestas adjacentes (processo que chamamos de normalização);
4. De posse desses trechos, cria-se um grafo auxiliar no qual cada trecho é representado por um único vértice (contração – ver figura abaixo); A matriz de custos [aij] desse grafo auxiliar é calculada de acordo com [26] e [12], sendo [gij] a matriz de custos do grafo original:
    a. aij = gij; se i e j não representam trechos (i e j são vértices comuns, que não fazem parte de nenhuma aresta identificada pelo mecanismo descrito acima);
    b. aij = gfim(i),j; se i representa um trecho e j representa um vértice comum. fim(i) equivale ao último vértice do trecho representado por i;
    c. aij = gi,ini(j); se i representa um vértice comum. ini(j) equivale ao primeiro vértice do trecho j; e
    d. aij = gfim(i),ini(j); se ambos i e j representam trechos
5. Com a matriz de custos definida, cria-se um conjunto de soluções para esse grafo auxiliar (neste trabalho, isso é feito através de métodos construtivos seguidos de busca local).
6. Faz-se a "tradução" das soluções produzidas com o grafo auxiliar para soluções viáveis com o grafo original. Isso é feito através da expansão dos nós que representam os trechos identificados anteriormente. A esse conjunto de soluções viáveis damos o nome de População Auxiliar.
7. Executa-se busca local nas soluções da População Auxiliar.
8. Insere-se a melhor solução do conjunto atual na População Auxiliar.
9. Exclui-se a pior solução da População Auxiliar e substitui-se o conjunto de

soluções atual com a primeira.

Abaixo, segue uma ilustração da identificação e contração de trechos presentes em várias soluções.

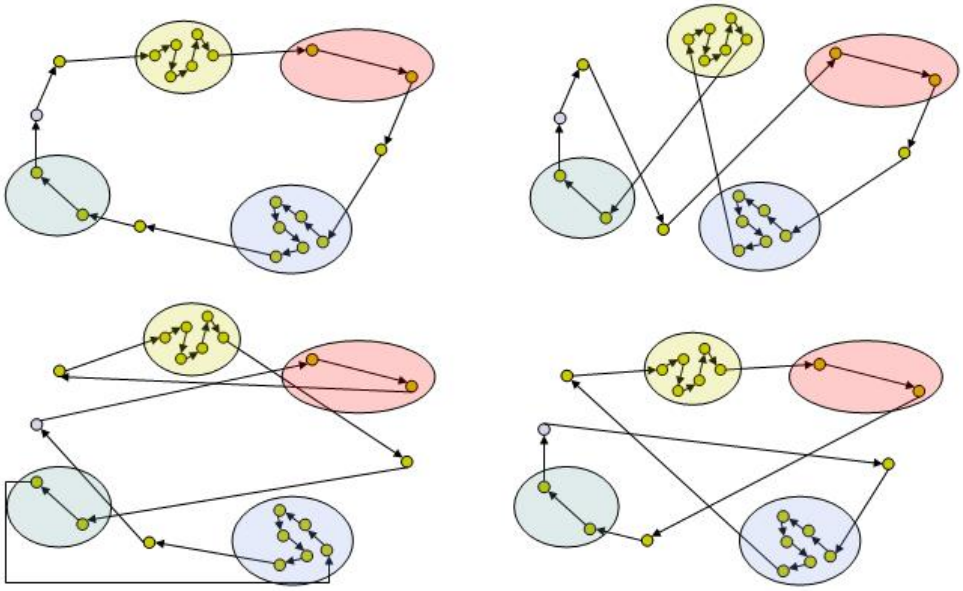

Figura 8: identificação e contração de trechos comuns a soluções-elite

Esses trechos são, após identificados, condensados. A partir desses novos nós, cria-se um grafo auxiliar e tenta-se resolver o problema sobre essa nova estrutura.

A principal idéia subjacente a esse procedimento é a suposição de que uma variável que está com um determinado valor em várias soluções de boa qualidade tende a possuir esse mesmo valor em uma solução ótima. Sendo assim, procura-se por arestas que estejam presentes em diversas soluções de alta qualidade e faz-se com que elas estejam automaticamente presentes nas próximas soluções. Além disso, cada heurística atua de uma maneira diferente, considerando-se a importância de uma determinada aresta para a construção de uma solução ótima. É interessante notar que essa suposição está relacionada com aquela feita por Glover (citada anteriormente).

Neste trabalho, as heurísticas utilizadas para gerar as soluções foram:

- Heurística do Vizinho Mais Próximo;
- Heurística da Inserção Arbitrária; e
- Uma variação que utiliza conjuntamente as heurísticas KSP e GKS, mostradas em [12].

Essas três heurísticas foram utilizadas devido a fazerem parte de três classes diferentes de heurísticas construtivas ([1]; [18]).

## 4. ALGORITMO MEMÉTICO COM VOCABULARY BUILDING

Como já foi dito anteriormente, o Vocabulary Building é uma instância do Path Relinking. Sendo esse último um método evolucionário, tem-se que o Vocabulary Building deve atuar sobre uma população de soluções.

Na literatura, o Algoritmo Memético tem se mostrado um dos mais eficazes métodos evolucionários encontrados para a resolução de problemas de otimização combinatória. É uma metaheurística cuja flexibilidade permite a inserção de novos elementos no processo de busca, tendo sido escolhida para a utilização dos métodos aqui propostos.

O algoritmo memético deste trabalho consiste, basicamente, de um algoritmo genético acrescido de busca local e dos procedimentos de Vocabulary Building discutido até então. O processo da implementação 1 é executado antes da geração de uma nova população;

o procedimento da implementação 2 é chamado após a geração da população em questão.

```
procedimento algoritmo_memetico_PV_CV(G);
Entrada: Matriz de distâncias G;
Saída: Melhor solução encontrada;
1.      pop_atual ← gerar_população_inicial();
2.      otimizar_populacao(pop_atual); //Busca local
3.      avaliar_populacao(pop_atual);
4.      P ← gerar_pool_inicial();
5.      enquanto (condições de paradas não satisfeitos) faça
6.              nova_pop ← ∅;
7.              elitismo(pop_atual, nova_pop);
8.              para cada solução S do conjunto elite faça
9.                      aplicar_pool_em_solucao(P, S);
10.                     otimizar_solucao(S); //Busca local
11.             fim –para;
12.             Copiar as soluções recém-modificadas em nova_pop;
13.             enquanto (nova_pop não estiver completa) faça
14.                     x, y ← selecionar_pais(pop_atual);
15.                     x', y' ← recombinacao(x, y);
16.                     mutacao(x');
17.                     mutacao(y');
18.                     nova_pop ← nova_pop ∪ {x'} ∪ {y'};
19.             fim-enquanto;
20.             otimizar_populacao(nova_pop);
21.             avaliar_populacao(nova_pop);
22.             pop_atual ← nova_pop;
23.             contracao_de_vertices(pop_atual);
24.     fim-enquanto;
25.     retorna pop_atual.melhor;
fim-procedimento;
```

Figura 9: algoritmo memético com Vocabulary Building

## 5. EXPERIMENTOS COMPUTACIONAIS

Foram feitos testes nas 27 instâncias assimétricas da TSPLIB, com 20 execuções e 1000 iterações para cada uma delas. A TSPLIB tem sido utilizada como parâmetro para comparar os testes realizados com o PCVA. Cada uma das instâncias representa um problema real cuja solução ótima é conhecida.

Os algoritmos apresentados foram implementados usando-se a linguagem C++ e os testes realizados em uma máquina Pentium IV 1.4 GHz e 128Mb de memória RAM.

Os testes realizados incluem:

- Algoritmo memético puro;
- Algoritmo memético com o procedimento da implementação 1;
- Algoritmo memético com o procedimento da implementação 2; e
- Algoritmo memético com ambos os procedimentos.

É interessante citar que houve uma forte cooperação entre ambos os métodos propostos. A seguir, tem-se a tabela relativa aos dois métodos atuando em conjunto, dentro do algoritmo memético.

Tabela 1: Algoritmo memético com Vocabulary Building (20 execuções)

| Instância TSPLIB | Gap Inicial | Ótimo (Nº de vezes) | Gap (%) na média | Tempo Médio de CPU (seg) |
|---|---|---|---|---|

| PCVA | (%) | | | |
|---|---|---|---|---|
| **br17** | 0,000 | **20** | 0,000 | 0,000 |
| **ftv33** | 1,555 | **20** | 0,000 | 0,500 |
| **ftv35** | 1,107 | 16 | 0,027 | 120,300 |
| **ftv38** | 1,275 | 14 | 0,039 | 173,650 |
| **p43** | 0,155 | 19 | 0,001 | 118,650 |
| **ftv44** | 3,506 | 14 | 0,186 | 225,050 |
| **ftv47** | 0,532 | **20** | 0,000 | 10,500 |
| **ry48p** | 1,680 | 10 | 0,282 | 160,850 |
| **ft53** | 4,713 | **20** | 0,000 | 36,350 |
| **ftv55** | 2,758 | **20** | 0,000 | 40,500 |
| **ftv64** | 2,887 | **20** | 0,000 | 104,200 |
| **ft70** | 1,464 | 0 | 0,616 | 742,750 |
| **ftv70** | 2,933 | 19 | 0,010 | 149,800 |
| **ftv90** | 2,153 | 17 | 0,057 | 180,200 |
| **ftv100** | 2,019 | 16 | 0,067 | 266,850 |
| **ftv110** | 2,656 | 18 | 0,020 | 389,950 |
| **ftv120** | 4,995 | 13 | 0,083 | 765,100 |
| **kro124p** | 3,684 | 2 | 0,136 | 647,450 |
| **ftv130** | 3,078 | 12 | 0,056 | 787,400 |
| **ftv140** | 2,231 | 11 | 0,062 | 1074,600 |
| **ftv150** | 2,068 | 7 | 0,067 | 1440,750 |
| **ftv160** | 0,596 | 18 | 0,015 | 380,500 |
| **ftv170** | 0,935 | 16 | 0,056 | 1325,000 |
| **rbg323** | 0,000 | **20** | 0,000 | 3,000 |
| **rbg358** | 0,000 | **20** | 0,000 | 2,000 |
| **rbg403** | 0,000 | **20** | 0,000 | 5,000 |
| **rbg443** | 0,000 | **20** | 0,000 | 7,000 |
| **Média** | 1,814 | 15,630 | 0,066 | 339,181 |

## 6. CONCLUSÃO

Esse trabalho tratou sobre a técnica de otimização conhecida como Vocabulary Building, uma técnica recorrente, cujas idéias aparecem várias vezes nas publicações do renomado pesquisador Fred Glover. Foi visto que esse é um conceito praticamente inexplorado em aplicações práticas, o que nos motivou a implementar duas formas distintas da técnica e a testar a sua utilização em uma metaheurística populacional.

Foram realizados testes computacionais (algo feito por somente uma publicação, até então) sobre o Problema do Caixeiro Viajante e, de acordo com os resultados, pode-se afirmar que o algoritmo desenvolvido atuou com uma boa eficácia e eficiência na resolução das instâncias testadas.

## 7. REFERÊNCIAS BIBLIOGRÁFICAS

[1] CIRASELLA, J.; JOHNSON, D.S.; MCGEOCH, L.A.; ZHANG, W. The Asymmetric Traveling Salesman Problem: Algorithms, Instance Generators and Tests. In: WORKSHOP ON ALGORITHM ENGINEERING AND EXPERIMENTS (ALENEX), 3, 2001, Washington. **Proceedings..**. Washington: Springer, 2001, p. 32-59.

[2] GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability**: a Guide to the Theory Of NP-Completness. New York: W. H. Freeman, 1979.

[3] GLOVER, F; KOCHENBERGER, G. A. **Handbook of Metaheuristic**, 1. ed. Norwell: Kluwer Academic Publishers, 2002.

[4] GLOVER, F. New Ejection Chain and Alternating Path Methods for Traveling Salesman Problems. In: SHARDA, R. (Ed.); BALCI, O. (Ed.); ZENIOS, S. (Ed.). **Computer Science and Operations Research**: New Developments in Their Interfaces. Cambridge: Elsevier, p. 449-509, 1992.

[5] GLOVER, F.; LAGUNA, M. Tabu Search. In: REEVES, C (Ed.). **Modern Heuristic Techniques for Combinatorial Problems**. Oxford: Blackwell Scientific Publishing, p. 71-140, 1993.

[6] GLOVER, F. A Template for Scatter Search and Path Relinking. In: HAO, J.K. (Ed.); LUTTON, E. (Ed.); RONALD, E. (Ed.); SHOENAUER, M. (Ed.); SNYERS, D. (Ed.). **Lecture Notes in Computer Science**, n. 1363, p. 13-54, 1997.

[7] GLOVER, F.; LAGUNA, M. **Tabu search**. Massachusetts: Kluwer Academic Publishers, 1997.

[8] GLOVER, F. Scatter Search and Path Relinking. In: CORNE, D. (Ed.); DORIGO, M. (Ed.); GLOVER, F. **New Ideas in Optimization**. London: McGraw-Hill, p. 297-316, 1999.

[9] GLOVER, F; LAGUNA, M.; MARTI, R. Fundamentals of Scatter Search and Path Relinking. **Control and Cybernetics**, vol. 29, n. 3, p. 653-684, 2000.

[10] GLOVER, F. Tutorial on Surrogate Constraint Approaches for Optimization in Graphs. **Journal of Heuristics**, vol. 9, n.3, p. 175-228, 2003.

[11] GLOVER, F. Parametric Combinations of Local Job Shop Rules. In: GLOVER, F. **Research Memorandum**, n. 117, GSIA, Carnegie Melon University, Pittsburg, 1963.

[12] GLOVER, F.; GUTIN, G.; YEO, A.; ZVEROVICH, A. Construction Heuristics for the Asymmetric TSP. **European Journal of Operational Research**, vol. 129, n. 3, p. 555-568, 2001.

[13] GUTIN, G. (Ed.); PUNNEN, A. P. (Ed.). **The Traveling Salesman Problem and Its Variations**. 1. ed. Boston: Springer, 2002.

[14] HANSEN, P.; RIBEIRO, C., **Essays and Surveys in Metaheuristics**. Boston**:** Springer, 2001.

[15] HOFFMAN, A. J.; WOLFE, P. History. In: LAWLER, E. L (Ed.); LENSTRA, J. K. (Ed.); RINNOOY KAN, A. H. G. (Ed.); SHMOYS, D. B. (Ed.). **The Traveling Salesman Problem**: A Guided Tour of Combinatorial Optimization. New York: John Wiley & Sons, 1985. p. 1-16.

[16] IBARAKI, T.; NONOBE, K.; YAGIURA, M. **Metaheuristics**: Progress as Real Problem Solvers. 1. ed. Boston: Springer, 2005.

[17] JOHNSON, D. S.; MCGEOCH, L.A. The Traveling Salesman Problem: A Case Study in Local Optimization, **Local Search in Combinatorial Optimization**, E. H. L. Aarts and J.K. Lenstra (ed), John Wiley and Sons Ltd, p. 215-310, 1997.

[18] JOHNSON, D. S.; GUTIN, G.; MCGEOCH, L. A.; YEO, A.; ZHANG, W.; ZVEROVICH, A. Experimental Analysis of Heuristics for the ATSP. In: GUTIN, G. (Ed.); PUNNEN, A. P. (Ed.). **The Traveling Salesman Problem and Its Variations**. 1 ed. Dordrecht: Kluwer Academic Publishers, 2002. p. 445-487.

[19] KELLY, J.P.; XU, J. Tabu Search and Vocabulary Building for Routing Problems. **Technical Rreport**, Graduate School of Business Administration, University of Colorado at Boulder, 1995.

[20] LAPORTE, G.; ASEF-VAZIRI, A.; SRISKANDARAJAH, C. Some Applications of the Generalized Travelling Salesman Problem. **Journal of Operations Research Society,** 47, p. 1461-1467, 1996.

[21] MELAMED, I. I.; SERGEEV, S. I.; SIGAL, I. Kh. The Traveling Salesman Problem. **Surveys**. New York: Plenum Publishing Corporation, 1990. p. 1147-1173.

[22] PAPADIMITRIOU, C. H.; STEIGLITZ, K. **Combinatorial Optimization**: Algorithms and Complexity. 1. ed. New York: Prentice Hall, 1982.

[23] REEVES, C. R. (Ed.). **Modern Heuristic Techniques for Combinatorial Problems**. Oxford: Blackwell Scientific Publication, 1993.

[24] ROCHAT, Y.; TAILLARD, E. Probabilistic Diversification and Intensification in Local Search for Vehicle Routing. **Journal of Heuristics**, vol. 1, n.1, p. 147-167, 1995.

[25] SCHOLL, A.; KLEIN, R.; DOMSCHKE, W. Pattern Based Vocabulary Building for Effectively Sequencing Mixed-Model Assembly Lines. **Journal of Heuristics**, vol. 4, n. 4, p. 359-381, 1998.

[26] YEO, A. Large Exponential Neighbourhoods for the Traveling Salesman Problem. **Preprint,** n. 47, Department Maths and CS, Odense University, 1997.