Mozar Baptista da Silva

Metaheurísticas Seqüenciais e Paralelas para uma Generalização do Problema do Caixeiro Viajante

Tese apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal Fluminense. Como parte dos requisitos necessários para obtenção do título de Mestre em Ciência da Computação.

Orientadores: Profª Drª Lúcia Maria Assumção Drummond (IC-UFF)
Profº Drº Luiz Satoru Ochi (IC-UFF)

Niterói
2001

Sumário

Lista de Tabelas

Lista de Figuras

Resumo

Propomos neste trabalho uma nova formulação matemática e novas metaheurísticas baseadas em conceitos do GRASP - *Greedy Adaptive Search Procedure*, VNS - *Variable Neighborhood Search* e *Busca Tabu* para a solução de uma generalização do clássico Traveling Salesman Problem (TSP) conhecido na literatura como The Traveling Purchaser Problem (TPP). São apresentadas alternativas para a construção de uma solução viável e para a busca local, reunindo conceitos de GRASP, VNS, heuristicas convencionais e Busca Tabu. Para avaliar o desempenho dos algoritmos propostos, comparamos os resultados dos nossos algoritmos com os melhores algoritmos da literatura, os algoritmos de busca tabu, com listas tabu dinâmicas, propostos por Voss. Propomos alguns algoritmos paralelos para estas metaheurísticas, utilizando três técnicas que são: mestre-escravo, distribuídos e independentes. Também são propostas versões onde utilizamos algoritmos diferentes em cada processador.

Analisamos a performance dos algoritmos propostos comparando todos os algoritmos entre si, considerando o tempo de execução, o custo da solução e o speedup no caso dos algoritmos paralelos.

Abstract

In this work we propose a new mathematical formulation and new metaheuristics based on concepts of GRASP - Greedy Adaptive Search Procedure, VNS - Variable Neighborhood Search and Tabu Search for getting an approximate solution for a generalization of the classical Traveling Salesman Problem (TSP) known in literature as The Traveling Purchaser Problem (TPP). We present different techniques for construction of initial solution and local search, based on concepts of GRASP, VNS, conventional heuristics and Tabu Search. The proposed algorithms were compared with the best algorithm in literature, the Tabu Search procedure that use dynamic lists proposed by Voss. We propose parallel versions of these algorithms, using three models: master-slave, distributed and independent. We also propose parallel versions that use different metaheuristics in each process that compose the parallel program.

We analyze the performance of the proposed algorithms comparing them among themselves and considering the execution times, the costs and the speedups in case of the parallel algorithms.

1 Introdução

A solução de problemas altamente combinatórios tem sido um desafio para os pesquisadores, devido à enorme dificuldade em solucioná-los de forma eficiente. Como a maioria destes problemas pertence à classe NP-Completo ou NP-Difícil, o uso exclusivo de métodos exatos é limitado. Portanto, o caminho mais promissor tem sido o uso de métodos aproximados ou heurísticos.

Mais recentemente, a partir de 1985, começaram a surgir heurísticas genéricas conhecidas como metaheurísticas ou heurísticas inteligentes. As metaheurísticas possuem como principal característica, ferramentas que possibilitam escapar de ótimos locais ainda distantes de um ótimo global em problemas de otimização.

Para resolver problemas de otimização, podemos optar por métodos exatos, baseadas em conceitos rígidos, expressos através de teoremas, ou então podemos utilizar métodos heurísticos ou aproximados, cuja característica marcante é a sua flexibilidade.

As metaheurísticas procuram conjugar as boas características de ambos os métodos, buscando obter algoritmos adaptativos, nem tão rígidos como os métodos exatos, mas também não tão flexíveis como as heurísticas convencionais.

O objetivo deste trabalho é propor novos algoritmos seqüenciais e paralelos, utilizando conceitos de metaheurísticas tais como: Busca Tabu, GRASP - (*Greedy*

Randomized Adaptative Search Procedure) e VNS - (*Variable Neighborhood Search*), para a solução aproximada de problemas de otimização combinatória, com elevada complexidade.

A importância dos algoritmos paralelos, está não somente na busca de redução do tempo computacional exigido na versão seqüencial, mas também na tentativa de melhorar a qualidade das soluções geradas através da troca de informações entre os processos.

Algumas metaheurísticas, tais como: Busca Tabu e Algoritmos Genéticos já possuem várias versões paralelas, embora a maioria utilize modelos mestre-escravo ou independentes. Outras metaheurísticas ainda têm o seu paralelismo pouco explorado, tais como os métodos: GRASP, *Ant systems*, *Scatter Search*, VNS e VNDS.

Esta lacuna observada na literatura afim, conjugada com o fato dos métodos Busca Tabu, GRASP e VNS apresentarem os melhores resultados em diversas aplicações, nos levou a estudar e a propor novos algoritmos seqüenciais e paralelos baseados nestes procedimentos. Para avaliar os algoritmos aqui propostos, aplicamos os algoritmos ao *Traveling Purchaser Problem*, que é uma generalização do caixeiro viajante (PCV), já que ele possui várias aplicações práticas e ainda é pouco explorado na literatura afim.

Para a construção de solução inicial, utilizamos o algoritmo proposto por Voss [48] e propomos 6 algoritmos que são: Solução Inicial Drop, Solução Inicial AddAleatório, Solução Inicial DropAleatório, Solução Inicial AddGeni, Solução Inicial DropGeni e Solução Inicial OrdemCompra. Para a busca local, utilizamos os algoritmos já difundidos na literatura AddProcedure, DropProcedure e Swap [48] além de propormos algumas variações que são: Busca AddDrop, Busca DropAdd, Busca

Híbrida e Busca Vizinhança (vizinhança variável).

Além de propormos os algoritmos de geração de solução inicial e busca local, propomos também algoritmos para as metaheurísticas GRASP, VNS, Busca Tabu e versões híbridas de GRASP com VNS, uma variação do VNDS *Variable Neighborhood Decomposition Search* e o VNDSI, que é um inversão da decomposição da vizinhança.

Propomos alguns algoritmos paralelos para estas metaheurísticas, utilizando três técnicas que são: mestre-escravo, distribuídos e independentes. Também são propostas versões onde utilizamos algoritmos diferentes em cada processador.

Salientamos contudo, que os algoritmos aqui propostos podem ser expandidos para a solução de outros problemas de otimização combinatória, cuja solução é formada por um subconjunto do conjunto de nós do grafo associado.

Os algoritmos propostos foram implementados em linguagem C utilizando a biblioteca MPI, e testes computacionais foram realizados para avaliar empiricamente os seus desempenhos.

Esta dissertação está divida da seguinte forma. No Capítulo 2 é dada uma descrição resumida do problema a ser tratado, o *Traveling Purchaser Problem* (TPP). No Capítulo 3 as metaheuríscas a serem enfocadas neste trabalho são apresentas. No Capítulo 4 é feita uma descrição dos modelos paralelos e dos algoritmos paralelos desenvolvidos. No Capítulo 5 são mostrados os algoritmos seqüenciais de construção e busca local propostos. No Capítulo 6 são descritos os algoritmos seqüenciais propostos seguidos dos testes computacionais efetuados. No Capítulo 7 são apresentados os algoritmos paralelos com os resultados computacionais e, finalmente no Capítulo 8 estão as conclusões e sugestões do problemas futuros.

2 O *Traveling Purchaser Problem* - TPP

O TPP é uma generalização do problema do Caixeiro Viajante (PCV) e foi introduzido por Ramesh em 1981 [40]. Possui diversas aplicações, dentre elas a compra de peças e matéria prima para indústrias manufatureiras, onde o custo total deve ser minimizado. As peças e as matérias primas são oferecidas por diversos fornecedores, localizadas em diferentes áreas e com diferentes preços. O comprador deverá escolher quais os fornecedores serão visitados, e planejar a rota de visitação, a fim de minimizar o custo de transporte e compra do material. Outro exemplo é o de escalonamento de um conjunto de tarefas sobre algumas máquinas. Neste caso, cada máquina possui um custo de preparação diferente e custo de processamento de tarefas diferentes, deve-se determinar quais as máquinas serão utilizadas e em que ordem as tarefas serão processadas nessas máquinas, de modo que, o custo total de processamento seja minimizado [36].

O TPP é computacionalmente intratável e algumas heurísticas são propostas para resolvê-lo aproximadamente, dentre elas temos: *Search Algorithm* [40], o *Generalized-Savings* [18], o *Tour-Reduction* [32], um algoritmo genético paralelo distribuído [31] e o *Commodity-Adding* [35]. Algumas variações destes algoritmos foram propostas por Pearn e Chien [36].

2.1 Descrição Formal do TPP

O *Traveling Purchaser Problema* (TPP) pode ser descrito da seguinte forma:

- Um conjunto de m mercados e uma origem m_0, $M = \{m_0, m_1, m_2, \ldots, m_m\}$.
- Um conjunto de n itens $K = \{k_1, k_2, \ldots, k_n\}$ a serem adquiridos em M.
- A matriz $T = (t_{ij}) : 0 \leq i, j \leq m$, onde t_{ij} representa o custo de mover-se do mercado i para o mercado j.
- A matriz $P = (p_{kj}) : 1 \leq k \leq n, 1 \leq j \leq m$, onde p_{kj} representa o custo do item k no mercado j.

Assumimos que:

- Cada item está disponível em pelo menos um mercado.
- Pode-se passar pela origem ou por qualquer mercado, quantas vezes forem necessárias, sem a necessidade de comprar algum item.
- Nenhum item é oferecido pela origem m_0.
- Um item $k_i \in K$ pode ter custos diferentes em diferentes mercados.

O objetivo do TPP é encontrar uma rota $M' \subseteq M$, partindo de m_0, passando por um subconjunto de $M - \{m_0\}$, retornando a m_0 comprando todos os itens, de forma que os custos da rota e da compra dos itens seja minimizado.

2.2 Formulação Matemática

Desenvolvemos uma formulação matemática para o TPP utilizando variáveis de fluxo de modo a evitar a formação de soluções desconexas com a origem. Com esta formulação, instâncias de pequeno porte poderão ser executados e seus resultados ótimos obtidos.

A formulação matemática para o problema, pressupõe um grafo completo usando distâncias euclidianas, e pode ser descrita como a seguir:

Tendo G(M,E) um grafo direcionado associado com o TPP. Definem-se as sequintes variáveis:

- $x_{ij} = 1$ se o arco (i,j) é visitado; 0 caso contrário;
- $y_{kj} = 1$ se o item k foi comprado no mercado j; 0 caso contrário;
- $T = (t_{ij}) : 0 \leq i, j \leq n$ matriz que representa o custo de mover-se do mercado i para o mercado j;
- $P = (p_{kj}) : 1 \leq k \leq m, 1 \leq j \leq n$, onde p_{kj} representa o custo do item k no mercado j;
- $A = (a_{kj}) : 1 \leq k \leq m, 1 \leq j \leq n$, onde $a_{kj} = 1$, se o item k é oferecido pelo mercado j e $a_{kj} = 0$ caso contrário;
- $F = (f_{ij}) : 0 \leq i, j \leq n$ representa o fluxo (número de items) transportado de i para j, onde $f_{ij} \geq 0$ é inteiro.

$$v(TPP) = minimizar \sum_{i,j \in M} t_{ij} x_{ij} + \sum_{k \in K, j \in M - \{m_0\}} p_{kj} y_{kj} \quad (1)$$

sujeito a:

$$\sum_{j \in M} a_{kj} y_{kj} = 1, \forall k \in K \tag{2}$$

$$\sum_{j \in M} x_{ij} = \sum_{j \in M} x_{ji} = 1, \, para \, i = m_0 \tag{3}$$

$$\sum_{i \in M - \{m_o\}} x_{ij} = \sum_{i \in M - \{m_o\}} x_{ji}, \forall j \in M - \{m_o\} \tag{4}$$

$$\sum_{i \in M} x_{ij} \geq \sum_{k \in K} a^*_{kj} y_{kj}, \forall j \in M - \{m_o\}, \, onde \, a^*_{kj} = a_{kj} / \sum_{p \in K} a_{pj} \tag{5}$$

$$\sum_{j \in M - \{m_o\}} f_{m_0 j} = 1 \tag{6}$$

$$\sum_{p \in M} f_{jp} = \sum_{i \in M - \{m_0\}} f_{ij} + \sum_{k \in K} y_{kj}, \forall j \in M - \{m_0\} \tag{7}$$

$$\sum_{j \in M - \{m_0\}} f_{j m_0} = m + 1 \tag{8}$$

$$f_{ij} > (x_{ij} - 1), \forall i, j \in M \tag{9}$$

$$x_{ij} \geq \frac{f_{ij}}{m+1}, \forall i, j \in M \tag{10}$$

$$x_{ij} \in \{0, 1\}, \forall i, j \in M \tag{11}$$

$$y_{kj} \in \{0,1\}, \forall k \in K, \forall j \in M \tag{12}$$

$$f_{ij} \geq 0 \, inteira, \forall i,j \in M \tag{13}$$

Na função objetivo, equação 2.1, o primeiro termo representa o custo de rota e o segundo o custo dos m itens adquiridos. Portanto devemos minimizar cada termo para conseguir atingir um ótimo global.

As equações 2.2 garantem que cada um dos m itens é adquirido em algum mercado.

Para garantirmos que o comprador parta da origem e retorne a mesma temos as equações 2.3, que para ser mais geral deveria ser ≥ 1 , ao invés de $= 1$, porque o comprador pode passar várias vezes pela origem. Entretanto como estamos trabalhando com distância euclidiana e grafo completo podemos simplificar a equação e considerar que o comprador retorne somente uma vez para origem, o mesmo raciocínio vale para os demais pontos visitados.

As equações 2.4 representam as equações de conservação de fluxo.

As equações 2.5 garantem que se pelo menos um item é comprado no mercado j, então o comprador deve passar ao menos uma vez pelo mercado j.

As próximas restrições são referentes ao fluxo de itens, ou seja, a cada item adquirido, o fluxo é incrementado. Portanto, se temos um fluxo f_0 ao chegarmos no mercado j e neste mercado é comprado um item k, então ao sairmos do mercado j temos um fluxo $f_1 = f_0 + 1$, esta restrição está descrita nas equações 2.7. Na origem compramos um item fictício como mostra a equação 2.6, para garantir que o comprador passará pela origem. Ao retornar a origem teremos adquirido todos os itens, inclusive

o fictício, portanto o fluxo de chegada na origem é $m+1$, representado na equação 2.8

As equações 2.9 garantem que, se o comprador passou pela aresta (i,j) então temos um fluxo positivo nesta aresta. As equações 2.10 garantem a volta, ou seja, se temos fluxo na aresta (i,j) então o comprador deve passar pela mesma.

Cada componente binária a_{kj} pode ser eliminada deste modelo, para tanto quando um item k não for oferecido por um mercado j, o custo deste item neste mercado será P, onde P deverá ser um valor bem alto. Como queremos minizar o custo, este produto não será adquirido neste mercado, o que possibilita as seguintes simplificações

As equações 2.2 após a simplificação ficariam da sequinte forma:

$$\sum_{j \in M} y_{kj} = 1, \forall k \in K \tag{14}$$

A simplificação das equações 2.5, resultariam nas sequintes equações:

$$\sum_{i \in M} x_{ij} \geq \sum_{k \in K} \frac{y_{kj}}{|K|}, \forall j \in M - \{m_o\} \tag{15}$$

3 Metaheurísticas Seqüenciais

3.1 Metaheurísticas

Problemas de otimização combinatória podem ser formulados como problemas de programação linear contínua ou discreta. Tais problemas podem ser resolvidos de forma ótima, utilizando-se, por exemplo, técnicas de busca em árvore ou relaxação lagrangeana, associadas a métodos de enumeração como *Branch and Bound* e *backtracking*. Entretanto essas técnicas necessitam de uma eficiente formulação matemática do problema, que nem sempre é fácil de se obter, e além disso, apresentam complexidade de tempo de processamento que cresce exponencialmente com o tamanho do problema [6, 27], limitando com isso, o uso exclusivo destas técnicas em aplicações reais.

As heurísticas são técnicas de se encontrar um boa solução viável (não necessariamente um ótimo global), utilizando um esforço computacional suportável. Estas técnicas são classificadas basicamente em dois tipos: construtivas e de busca local. A primeira constrói passo a passo uma solução, enquanto a segunda, parte de uma solução e tenta melhorá-la realizando uma busca no espaço de soluções vizinhas. Devido às limitações das heurísticas convencionais, foram desenvolvidas novas técnicas para tratar instâncias maiores e ajudar (guiar) as heurísticas na busca de soluções. Esta técnicas são chamadas de metaheurísticas [6, 27].

Atualmente existe um grande número de metaheurísticas na literatura incluindo: Redes Neurais (RN), *Simulated Annealing* (SA), Algoritmos Genéticos (AG), *Ant Systems* (AS), GRASP, VNS, *Scatter Search* (SS), Busca Tabu (BT), entre outras.

Algumas das metaheurísticas como Redes Neurais e Simulated Annealing, embora muito conhecidas em diversas áreas de conhecimento, não têm alcançado bons resultados em diversas aplicações na área de otimização combinatória. Em parte, estes resultados podem ser justificados pela dificuldade de encontrar o melhor conjunto de parâmetros para estes métodos, como pela existência de boas metaheurísticas na área de otimização combinatória.

De uma forma geral, na área de otimização combinatória, os melhores resultados normalmente têm sido alcançados pela técnica BT na sua forma mais atual incluindo etapas de intensificação e diversificação, além de técnicas eficientes para gerenciar listas Tabu.

Contudo, recentemente duas outras técnicas têm também se destacado com bons resultados, ao lado da BT, estas são GRASP e VNS.

Como a BT já está bastante explorada pela literatura afim, este trabalho propõe enfocar o uso do GRASP e do VNS isoladamente ou em versões híbridas(entre eles e com BT), na solução de uma generalização do PCV e comparar com o algoritmo de BT proposto por Voss 1996 [48, 49].

A seguir veremos uma breve descrição das seguintes metaheurísticas: Busca Tabu [15], GRASP [42] e VNS [21].

3.2 Busca Tabu

A *Busca Tabu* foi proposta por Glover [15, 16, 17]. Ela pode ser descrita como uma heurística de alto nível para resolver problemas de otimização, desenvolvida

para guiar heurísticas de busca local e evitar paradas prematuras em ótimos locais ainda distantes de um ótimo global. A maioria das metaheurísticas são técnicas adaptativas de busca que visam uma exploração inteligente do espaço de soluções. Para conseguir seus objetivos, a Busca Tabu utiliza duas técnicas para direcionar a pesquisa. A primeira é a lista Tabu, que guarda as soluções recentemente visitadas, para evitar ciclos. A segunda é o uso de memórias que auxiliam o processo de busca. Uma delas, conhecida como memória de curto prazo, tem como objetivo proibir, durante algumas iterações, determinados movimentos que se realizados podem fazer o algoritmo retornar a uma solução já visitada recentemente. Na memória de longo prazo, o objetivo é o de buscar novas regiões de busca ainda pouco exploradas [7, 8]. Este processo é conhecido como um *processo de diversificação de busca.*

A BT por outro lado, incentiva uma busca mais intensa em regiões mais promissoras, conhecida como *fase de intensificação.*

A lista Tabu, normalmente, contém os atributos dos movimentos reversos que devem ser proibidos ou desestimulados (movimentos Tabu). Esses movimentos permanecem por um certo tempo na lista Tabu. Esse tempo é chamado de *Prazo Tabu.* Um *Movimento Tabu* pode, no entanto, ser realizado caso ele satisfaça ao critério de aspiração. O critério de aspiração, poderia ser, por exemplo, aceitar um movimento que gere uma solução melhor que a melhor solução encontrada até o momento, mesmo tendo ele o status de Tabu [27].

A eficiência dos algoritmos Tabu dependem dos seguintes parâmetros: definição da vizinhança, a geração da solução inicial, a gerencia da lista Tabu, critério de aspiração e o critério de parada.

Algoritmo Básico 1 : *Busca Tabu*

```
1   s_0 = Solução inicial
2   s = s_0;
3   s* = s_0;
4   listatabu = {}
5   enquanto critério de parada não satisfeito faça
6       encontrar a melhor solução s+ ∈ viz(s) e ∉ listatabu;
7       se custo(s+) < custo(s*) então
8           s* = s+;
        senão
9           se custo(s+) > custo(s) então
10              listatabu = listatabu ∪ {s};
            fim se
        fim se
11      s = s+;
    fim enquanto
```

No algoritmo 1, temos uma descrição do algoritmo básico. Inicialmente gera-se uma solução inicial s_0 através de qualquer método construtivo. Inicializa-se a melhor solução s^* e a solução corrente s com s_0. A lista Tabu é inicializada como vazia.

Na linha 6, a melhor solução vizinha não Tabu ou que satisfaça o critério de aspiração é selecionada. Caso a solução encontrada seja melhor que a melhor encontrada até o momento, atualiza-se esta última na linha 8. Na linha 10, o movimento é inserido na lista Tabu, caso este não melhore a ultima solução encontrada, então o algoritmo está em um ótimo local e tenta sair, permitindo soluções piores e proibindo o movimento reverso.

A gerencia da lista Tabu é muito importante para impedir a ciclagem, e permitir que a busca escape de ótimos locais, ainda distantes de um ótimo local. Deve-se tomar cuidado para que a lista Tabu não proíba a visitação de soluções que ainda não foram visitadas e que a busca não fique bloqueada.

Duas técnicas podem ser incorporadas ao algoritmo básico, a fim de se explorar melhor o espaço de soluções. A primeira é a diversificação que consiste do uso de uma memória para detectar quais as regiões dos espaço de soluções foram pouco exploradas e, portanto devem ser beneficiadas na geração de novas soluções. A segunda técnica, chamada de intensificação, também utiliza uma memória para detectar as regiões que deram bons resultados, a fim de realizar uma busca mais intensa nessas regiões.

3.3 GRASP - *Greedy Randomized Adaptative Search Procedure*

A metaheurística GRASP, foi proposta por Feo e Resende em 1995 [42], e consiste da combinação de um método construtivo e uma busca local. O método construtivo deve a cada iteração gerar uma nova solução na qual será realizada a busca local [42].

Na fase de construção, uma solução viável é formada elemento a elemento, mas de modo diferente dos métodos de construção tradicionais. A cada iteração desta fase, o elemento a ser inserido é determinado por uma função gulosa adaptativa, que estima o benefício da inserção de cada elemento e cria uma lista de candidatos, da qual um elemento será selecionado aleatoriamente. Se o tamanho da lista for um, reduz-se ao uso de uma função gulosa. A lista de candidados é adaptativa por que a cada iteração deve-se refazer ou atualizar a lista. O componente probabilístico do GRASP é caracterizado pela escolha aleatória de um elemento dentre os melhores da lista de candidatos [2, 27, 42].

A solução gerada pela fase de construção não é necessariamente um ótimo local, portanto deve-se realizar uma busca local a fim de melhorar as soluções encontradas na fase de construção [2]

Algoritmo Básico 2 : *GRASP* - Greedy Randomized Adaptative Search Procedure

1 $custo(s^*) = \infty$;
2 **para** i **de** 1 **até** N **faça**
3 Construir uma nova solução s
4 Aplicar busca local em s gerando s^+
5 **se** $custo(s^+) < custo(s^*)$ **então**
6 $s^* = s^+$;
fim se
fim para

No algoritmo 2, temos uma versão básica do algoritmo GRASP, onde na linha 3 gera-se uma solução inicial s através de algum algoritmo construtivo, para então na linha 4 aplicar uma busca local, a fim de se obter o ótimo local s^+. Caso a solução obtida seja a melhor até o momento então atualiza-se, a melhor solução s^* na linha 6.

3.4 VNS - *Variable Neighborhood Search*

Esta metaheurística foi proposta por Mladenović em 1995 [21, 22, 29] e consiste em uma busca local com mudança dinâmica de vizinhança. A vizinhança é expandida ou novos métodos de escolha da vizinhança são utilizados à medida que não se obtêm soluções melhores na vizinhança atual [27]. O objetivo de se trabalhar com uma vizinhança pequena da solução corrente e ir expandindo-a ou modificando-a gradualmente é realizar uma intensificação na solução corrente. Quando essa vizinhança se tornar saturada, aumenta-se a vizinhança, que pode ser expandida de tal forma que os vizinhos sejam completamente diferentes um do outro. Neste ponto, temos uma diversificação.

Algoritmo Básico 3 : *VNS* - Variable Neighborhood Search

```
1   gerar um solução inicial s_0
2   s* = s_0;
3   enquanto critério de parada não satisfeito faça
4      k = 1
5      enquanto k < K_max faça
6         Gere aleatoriamente uma solução s ∈ Viz_k(s*);
7         Aplicar busca local em s gerando s+;
8         se custo(s+) < custo(s*) então
9            s* = s+;
10              k = 1;
          senão
11              k = k + 1
          fim se
       fim enquanto
    fim enquanto
```

No algoritmo 3, temos uma versão básica do algoritmo VNS (Variable Neighborhood Search). Viz_k é a k-ésima forma da vizinhança de s. Na linha 1 gera-se uma solução inicial através de algum algoritmo construtivo, para então na linha 6 realizar uma busca na k-ésima vizinhança obtendo-se s. Na linha 7, aplica-se uma busca local em s, a fim de se obter o ótimo local s^+. Caso a solução obtida seja a melhor até o momento, atualiza-se a melhor solução s^* e retorna-se para a primeira vizinhança, caso contrário, passa-se para a próxima vizinhança na linha 8.

Se na linha 4 for realizada uma busca pela melhor solução vizinha a s^*, teremos uma algoritmo de descida chamado de VND (*Variable Neighborhood Descent*).

Outra versão do VNS é o VNDS (*Variable Neighborhood Decomposition Search*), onde a k-ésima vizinhança é obtida pela decomposição da vizinhança de acordo com k, ou seja, fixam-se N-k variáveis do problema, onde N é o número de variáveis,

passando-se a resolver o problema com as k variáveis restantes. A k-ésima vizinhança é constituída de todas as possíveis variações das N-k variáveis.

4 Metaheurísticas Paralelas

Os algoritmos paralelos têm sido usados para resolver problemas de grande porte, o que se traduz diretamente na tentativa de reduzir os altos custos computacionais envolvidos. A motivação básica para os muitos estudos de algoritmos paralelos considerando-se metaheurísticas é a redução do tempo de processamento necessário para se atingir uma solução aceitável. Além disso, notou-se que em alguns casos os algoritmos paralelos apresentavam soluções melhores do que os algoritmos seqüenciais. Atualmente, a pesquisa de algoritmos paralelos está crescendo, utilizando diferentes estratégias de paralelização como por exemplo:

- modelos independentes ou com comunicação
  - independentes: os processos executam até o fim sem trocar nenhum tipo de informação.
  - com comunicação: os processos trocam informações durante a execução.
- supervisionado ou não supervisionado
  - supervisionado: existe um processo central/mestre que controla toda a aplicação.

  - não supervisionado: aplicações distribuídas, onde não existe o processo central/mestre.

- síncrono ou assíncrono

  - síncrono: os processos executam troca de informação, em pontos específicos do algoritmo.

  - assíncrono: as trocas de informação são realizadas de acordo com as condições de cada processo.

Hoje, já existem algumas classificações sobre o paralelismo de várias metaheurísticas, como por exemplo: Algoritmos Genéticos [24, 31] e Busca Tabu [7, 8], e algumas sugestões para o algoritmo GRASP [1, 2, 27, 28, 30] e para o VNS [30]

4.1 Formas de Paralelização

Verhoeven e Aarts [47], apresentam conceitos que podem ser utilizados na paralelização de qualquer heurística baseada em busca local. Eles apresentam dois tipos básicos de abordagem: percurso único e múltiplos percursos. Em um algoritmo de percurso único, executa-se um único percurso no espaço de soluções, enquanto que no algoritmo de múltiplos percursos, vários percursos são executados simultaneamente.

A classe de percurso único, é dividida em passo único e passo múltiplo. O primeiro consiste em particionar a vizinhança em subconjuntos e distribuí-los entre os processadores, para que cada um deles avalie os movimentos em seu subconjunto. O movimento a ser realizado será o melhor dentre os avaliados pelos processadores. Na segunda estratégia, passos múltiplos, a vizinhança ou domínio do problema é particionado entre os processadores, que executam alguns passos do algoritmo sobre a partição rece-

bida, para em seguida, serem coletadas e combinadas para formar a solução completa do problema [27, 47].

A classe de múltiplos percursos, também pode ser dividida em duas classes que são: percursos interativos e percursos independentes. Esta divisão está relacionada com a comunicação entre os processos. Na estratégia de percursos interativos, durante a execução da busca existe comunicação entre os processos, enquanto na estratégia de percursos independentes os processos só se comunicam após o termino do algoritmo, onde a melhor solução é selecionada dentre as fornecidas por cada processador [47].

4.2 GRASP Paralelo

Uma vez que as iterações do GRASP são independentes, a forma mais direta de paralelização é a do tipo percursos múltiplos independentes. Cada processador executa o mesmo algoritmo GRASP, para a mesma entrada de dados. Supondo que o algoritmo seqüencial execute num_max_it iterações e tenhamos P processadores cada um deles executando um algoritmo GRASP seqüencial, então cada processador executará $\frac{num\_max\_it}{P}$ iterações. A melhor solução é a melhor encontrada entre todos os processadores.

Na literatura, verifica-se que, grande parte dos algoritmos GRASP paralelos, pertencem à classe de percursos múltiplos independentes, como por exemplo Li, Pardalos e Resende [33], que desenvolveram um algoritmo paralelo para a resolução do problema de alocação quadrática, Alvim [1] que apresentou um algoritmo GRASP paralelo para o problema de decomposição de matrizes e Martins, Resende e Ribeiro [28] que apresentaram um algoritmo GRASP paralelo para o problema de Steiner em grafo.

Feo, Resende e Smith [41] apresentaram um algoritmo GRASP paralelo para o problema do conjunto máximo independente, onde a estratégia usada foi percurso único de múltiplos passos.

4.3 Busca Tabu Paralela

Crainic [7, 8], construiu uma taxonomia para a Busca Tabu, onde o paralelismo é analisado de acordo com três dimensões. As primeiras duas dimensões representam os esquemas de paralelização relativos ao controle de trajetória de busca e as formas de comunicação e processamento de informação, enquanto a terceira trata das estratégias usadas para particionar o domínio e especificar os parâmetros para cada busca.

A primeira dimensão se refere à cardinalidade de controle, que pode ser *1-control* ou *p-control*. O controle da pesquisa paralela pode começar com um processador, geralmente chamado mestre ou processador principal, ou ser distribuído entre diferentes processadores. O mestre coleta informação, distribui as tarefas a serem executadas por outros processadores, e determina quando a busca tem que parar. As tarefas que são delegadas podem consistir na exploração paralela da vizinhança ou na construção e avaliação da lista de candidatos. No segundo caso, o controle da busca é compartilhado entre $p > 1$ processadores. Cada processo tem a responsabilidade sobre a sua própria busca, assim como sobre a comunicação entre os processos.

A segunda dimensão da taxonomia é baseada no tipo e flexibilidade do controle: leva-se em consideração a organização da comunicação, sincronização e hierarquia, assim como a forma que a comunicação é processada e compartilhada entre os processos. Esta dimensão é composta de quatro formas. A primeira forma corresponde à *sincronização rígida de processos*. Um modo de operação síncrono geralmente indica

que todos os processos têm que parar e se engajar em alguma forma de comunicação e troca de informação, em pontos específicos do algoritmo (número de iterações, intervalos de tempo, estágios algorítmicos especificados, etc). Esta organização é classificada como rígida, quando pouca troca de informação ocorre entre processos que são dedicados a executar o mesmo nível de tarefas.

A segunda forma é também caracterizada por um modo de operação síncrono, mas com um nível maior de comunicação que permite construir e trocar conhecimento. Ele é chamado de *sincronização baseada em conhecimento*.

A terceira e quarta forma usam o modo de comunicação assíncrono. Neste contexto, cada processo armazena e trata sua própria informação, iniciando a comunicação com alguns ou todos os outros processos de acordo com sua própria lógica interna e *status*. São definidos estes dois níveis em função da quantidade, qualidade e tratamento da informação trocada.

Na terceiro forma, a assíncrona, cada processo executa uma busca Tabu eventualmente diferente sobre todas as partes do domínio. Quando um processo encontra uma solução melhor (localmente ou globalmente, de acordo com a estratégia escolhida), este difunde esta para todos ou alguns de seus vizinhos. Na quarta forma, a *assíncrona baseada em conhecimento*, os conteúdos de comunicação são analisados para inferir informações adicionais relacionadas à trajetória de busca e características globais de boas soluções. Memórias globais e listas Tabu que refletem a dinâmica da exploração paralela assíncrona do domínio podem assim ser construídas baseadas nas soluções e conteúdos de memória enviados pelas buscas individuais. Por isso, a mensagem recebida por um processo geralmente é mais rica e não idêntica, à outra mensagem enviada inicialmente por um outro processo.

A terceira dimensão consiste na classificação de Voss, onde os únicos critérios considerados se referem ao número de soluções iniciais (diferentes) e ao número de estratégias de solução (estabelecimento de parâmetros, gerencia de lista Tabu, etc.) usados por uma implementação em particular. Assim, ele identifica os seguintes 4 casos:

- SPSS - *Single initial point single strategy*: é o caso mais simples.
- SPDS - *Single point different strategies*: se refere ao caso quando cada processador executa uma Busca Tabu diferente, mas inicia com a mesma solução inicial.
- MPSS - *Multiple points single strategy*: referente ao caso onde cada processador inicia de uma solução diferente do domínio, mas usa a mesma Busca Tabu e regras para explorar o domínio.
- MPDS - *Multiple points different strategies*: é o mais geral, contém todas as outras estratégias como casos especiais.

Verificamos que a maior parte dos trabalhos na área, trata de projeto de algoritmos paralelos síncronos (Malek et al (1989) [25] , Taillard (1991,1993 e 1994) [44, 43, 45], Chakrapani e Skorin-Kapov (1992 e 1993) [4] [5], Battiti e Techiolli (1992) [3], Fiechter (1994) [12], Porto (1994) [39]), embora o estudo e projeto de algoritmos assíncronos baseado em conhecimento seja considerado extremamente promissor como estratégia de paralelização [9].

4.4 VNS e VNDS Paralelo

Como esta metaheurística é recente [22], existem poucos trabalhos sobre a sua paralelização. Algumas estratégias de paralelização podem ser formuladas

baseando-se na paralelização das metaheurísticas anteriores (GRASP, Algoritmos Genéticos e Busca Tabu).

A formulação mais imediata seria utilizar múltiplos percursos independentes, onde os processadores executariam algoritmos VNS seqüenciais, com parâmetros distintos.

Outra forma seria a cada busca local, dividir o espaço de busca entre os processadores, onde teríamos uma estratégia de percurso único e passo único.

A formulação mais complexa seria a de percursos múltiplos interativos assíncronos, onde cada processador executaria um algoritmo VNS seqüencial, que realiza troca de conhecimento com os outros processadores de forma assíncrona. Para esta estratégia utiliza-se uma memória centralizada onde serão armazenadas as melhores soluções. O algoritmo poderia ser reinicializado escolhendo-se uma solução contida na memória centralizada [27].

5 Algoritmos de Construção e Busca Local

Em todas as metaheurísticas analisadas, Busca Tabu, GRASP e VNS, são necessários dois algoritmos básicos, um para geração da solução inicial e outro para a busca local. Neste capítulo serão descritos os vários algoritmos propostos para a geração da solução inicial e para a busca local tanto para os algoritmos seqüenciais como paralelos propostos nos próximos capítulos.

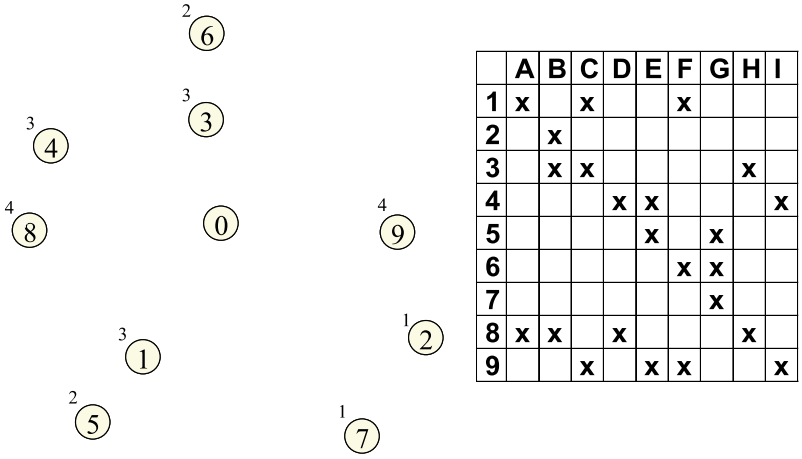
5.1 Algoritmos de Solução Inicial

A solução inicial pode ser gerada de várias formas. Neste trabalho apresentamos métodos de geração de solução inicial. Alguns existentes na literatura foram adaptados para o TPP, e outros propostos neste trabalho.

Para todos os exemplos utilizaremos o grafo da figura 1. Onde temos um grafo completo, e a tabela indica quais produtos são oferecidos pelos mercados. O custo dos produtos, de modo a simplificar, é igual a um para todos os mercados.

5.1.1 Solução Inicial Add

Este algoritmo foi desenvolvido por Golden, Levy e Dahl, em 1981 [18] e consiste em inserir o nó que represente a melhor economia, até que não se possa mais fazer economia, veja na figura 2. Se um produto não é oferecido por um mercado então o custo desse produto neste mercado será igual a M (M deve ser um valor elevado).



	A	B	C	D	E	F	G	H	I
1	x		x			x			
2		x							
3		x	x					x	
4				x	x				x
5					x		x		
6						x	x		
7							x		
8	x	x		x				x	
9			x		x	x			x

Figura 1: Grafo utilizado nos exemplos.

Inicialmente temos a rota partindo da origem e retornando a origem, onde o custo da solução é o custo de comprar todos eles na origem, ou seja, número de produtos x M, a cada passo escolhe-se o nó que oferece a maior economia, caso seja inserido na solução parcial.

Algoritmo Básico 4 :*Solução Inicial Add*

1 $Rota = \{ORIGEM - ORIGEM\}$
2 $Custo = N^o\, de\, produtos * M$
3 **enquanto** pode-se fazer economia adicinando nós **faça**
4 no =nó que oferece maior economia, caso seja inserido na rota;
5 pos =posição em que o nó no deve entrar na rota
6 $Rota = Rota + (no, pos)$
7 $Custo = Custo - Economia(no, pos)$
fim enquanto

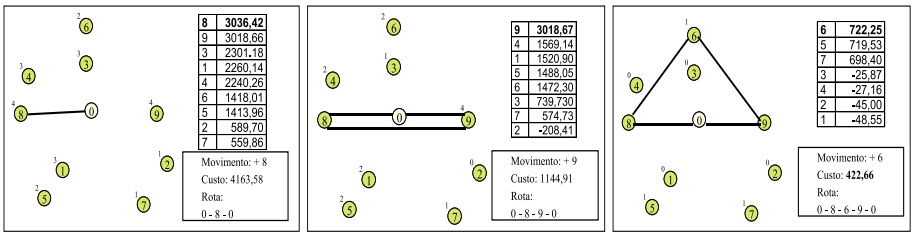


Figura 2: Passos do algoritmo Solução Inicial Add.

5.1.2 Solução Inicial Drop

Neste Algoritmo, parte-se inicialmente de solução com todos os nós e a cada passo retira-se o nó que descartado, oferece a maior economia, veja na figura 3. A rota inicial é determinada utilizando-se o algoritmo GENIUS [13] para o problema do caixeiro viajante associado.

Algoritmo Básico 5 : *Solução Inicial Drop*

1 $Rota$ =Genius aplicado a todos os nós disponíveis
2 $Custo = Custo\ retornado\ pelo\ Genius$
3 **enquanto** pode-se fazer economia retirando nós **faça**
4 no =nó que oferece maior economia, caso seja removido da rota;
5 $Rota = Rota - no$
6 $Custo = Custo - Economia(no)$
fim enquanto

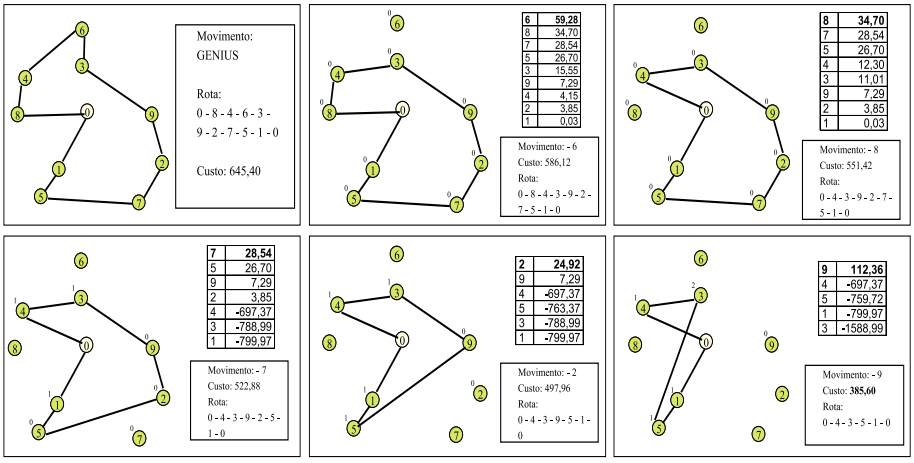


Figura 3: Passos do algoritmo Solução Inicial Drop.

5.1.3 Solução Inicial AddGeni

Este algoritmo se diferencia do 5.1.1, em relação ao cálculo da economia. Neste algoritmo o cálculo da economia de distância, ultiliza o procedimento de inserção do GENIUS [13], enquanto que o primeiro avalia a inserção de um novo nó somente

entre dois vizinhos, veja a figura 4.

Algoritmo Básico 6 :*Solução Inicial AddGeni*

1 $Rota = \{ORIGEM - ORIGEM\}$
2 $Custo = N^o\, de\, produtos * M$
3 **enquanto** pode-se fazer economia adicionando nós **faça**
4 no =nó que oferece maior economia, caso seja inserido na rota;
5 $Rota = Rota$+InserçãoGeni(no)
6 $Custo = Custo - EconomiaGeni(no)$
fim enquanto

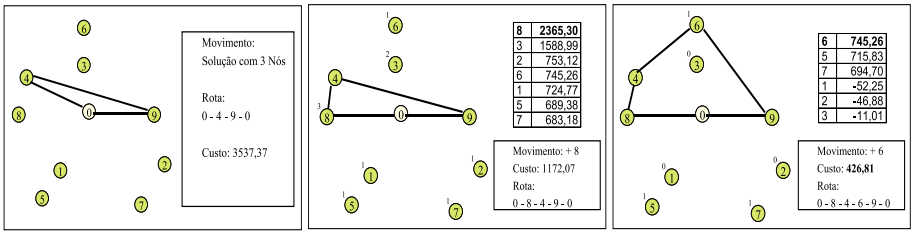


Figura 4: Passos do algoritmo Solução Inicial AddGeni.

5.1.4 Solução Inicial DropGeni

Este algoritmo é similar 5.1.2, entretanto a avaliação da remoção de um nó utilizando o procedimento de remoção do algoritmo GENIUS [13], veja a figura 5.

Algoritmo Básico 7 :*Solução Inicial DropGeni*

1 $Rota$ =Genius aplicado a todos os nós disponíveis
2 $Custo = Custo\, retornado\, pelo\, Genius$
3 **enquanto** pode-se fazer economia retirando nós **faça**
4 no =nó que oferece maior economia, caso seja removido da rota;
5 $Rota = Rota$−ExclusãoGeni(no)
6 $Custo = Custo - EconomiaGeni(no)$
fim enquanto

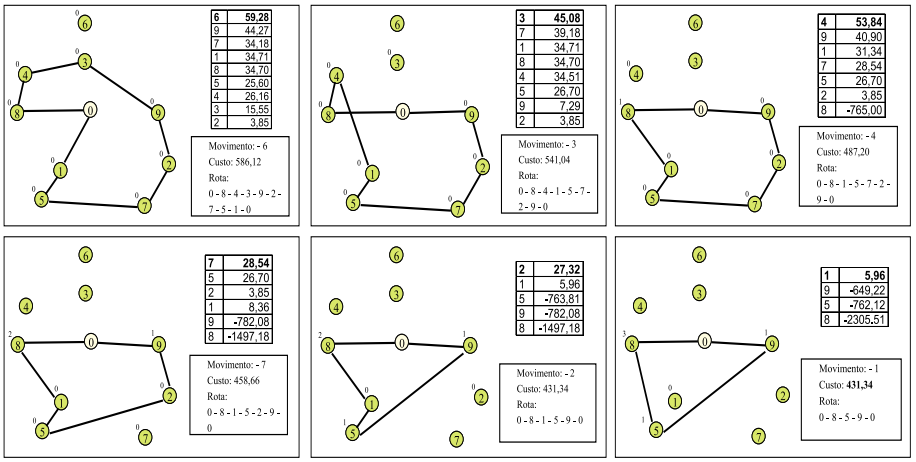


Figura 5: Passos do algoritmo Solução Inicial DropGeni.

5.1.5 Solução Inicial AddAleatório

Este algoritmo é uma varição do 5.1.1, onde a cada passo, cria-se uma lista dos nós mais econômicos e escolhe-se aleatoriamente um nó desta lista para ser inserido na rota atual, veja a figura 6. Observe que este procedimento utiliza conceito de GRASP na construção de uma solução.

Algoritmo Básico 8 :*Solução Inicial AddAleatório*

1 $Rota = \{ORIGEM - ORIGEM\}$
2 $Custo = N^o\, de\, produtos * M$
3 **enquanto** pode-se fazer economia adicinando nós **faça**
4 no = Seleciona aleatoriamente dentre os K nós mais econômicos;
5 pos =posição em que o nó no deve entrar na rota
6 $Rota = Rota + (no, pos)$
7 $Custo = Custo - Economia(no, Pos)$
fim enquanto

5.1.6 Solução Inicial DropAleatório

Este algoritmo, se diferencia do 5.1.2, uma vez que, a cada passo, uma lista de candidatos é gerada. Esta lista é formada pelos elementos mais econômicos de acordo com uma função gulosa que avalia o benefício da remoção de um elemento. Um

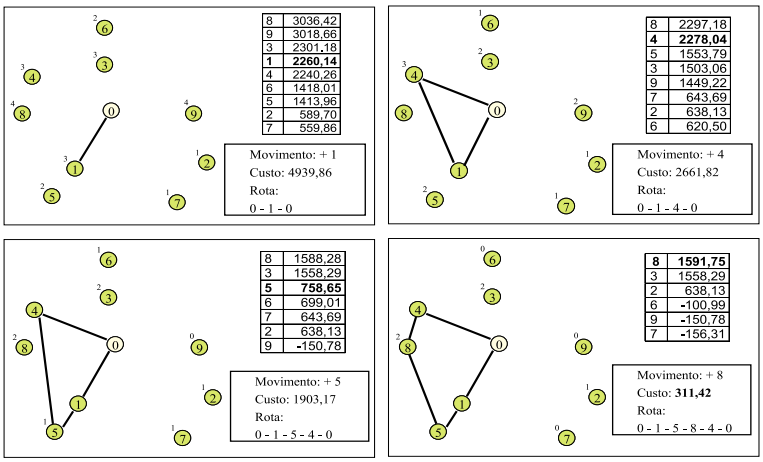


Figura 6: Passos do algoritmo Solução Inicial AddAleatório.

elemento é escolhido aleatoriamente desta lista, para em seguida ser retirado da rota atual, veja a figurafigSiDropAleatorio. Igualmente a 5.1.5

Algoritmo Básico 9 : *Solução Inicial DropAleatório*

1 $Rota$ =Genius aplicado a todos os nós disponíveis
2 $Custo = Custo\, retornado\, pelo\, Genius$
3 **enquanto** pode-se fazer economia retirando nós **faça**
4 no =Seleciona aleatoriamente dentre os K nós mais econômicos;
5 $Rota = Rota - no$
6 $Custo = Custo - Economia(no)$
fim enquanto

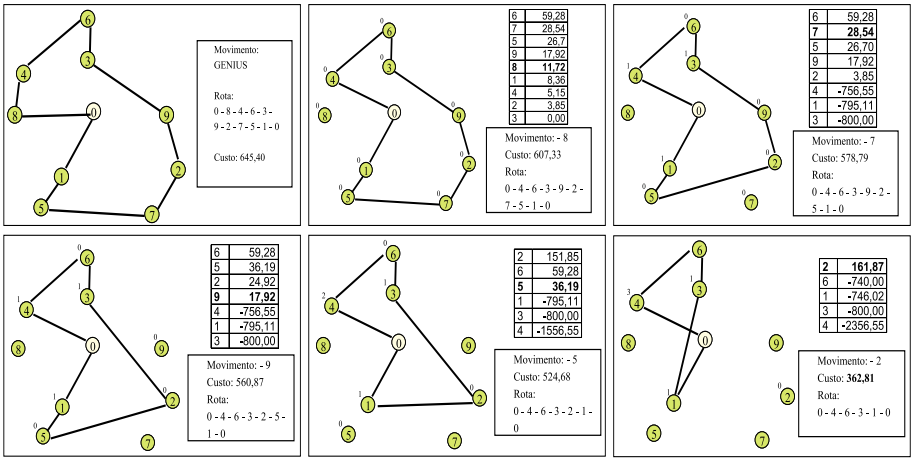


Figura 7: Passos do algoritmo Solução Inicial DropAleatório.

5.1.7 Solução Inicial OrdemCompra

Este algoritmo, visa especificar uma ordem de compra para os produtos. Ele está dividido em três etapas, a primeira consiste em gerar uma ordem de compra aleatória. A partir desta ordem de compras gera-se um grafo direcionado e acíclico, onde só há arcos entre dois níveis adjacentes e sucessivos ($n(i) \longrightarrow n(i+1)$). Não existem arcos de um nível $n(i)$ para níveis anteriores, nem entre dois nós de um mesmo nível. É criado um nó s conectado com todos os nós que oferecem o primeiro produto da lista ordenada. Analogamente um nó terminal c conectado com todos os nós que oferecem o último produto, é criado. A segunda parte consiste em a partir deste grafo direcionado e acíclico, gerar o caminho mínimo aplicando o algoritmo Dijkstra entre s e c. A terceira etapa consiste em usar um procedimento de atalhos para remover os nós que aparecem mais de um vez no caminho mínimo, gerando-se desta forma a rota inicial. veja a figura 8

Algoritmo Básico 10 : *Solução Inicial OrdemCompra*

1 Forma aleatoriamente a ordem de compra dos produtos
2 Construir grupos de níveis que oferecem cada produto
3 Aplicar Dijkstra para encontrar um caminho entre os produtos
4 Remover duplicações de nós na solução
5 Calcular custo da solução

5.2 Algoritmos de Busca Local

Na busca local foram desenvolvidos os seguintes algoritmos.

5.2.1 Busca Local Add

Este algoritmo é o mesmo para a geração da solução inicial Add 5.1.1, diferindo deste, por partir de uma solução já construída por algum algoritmo anterior.

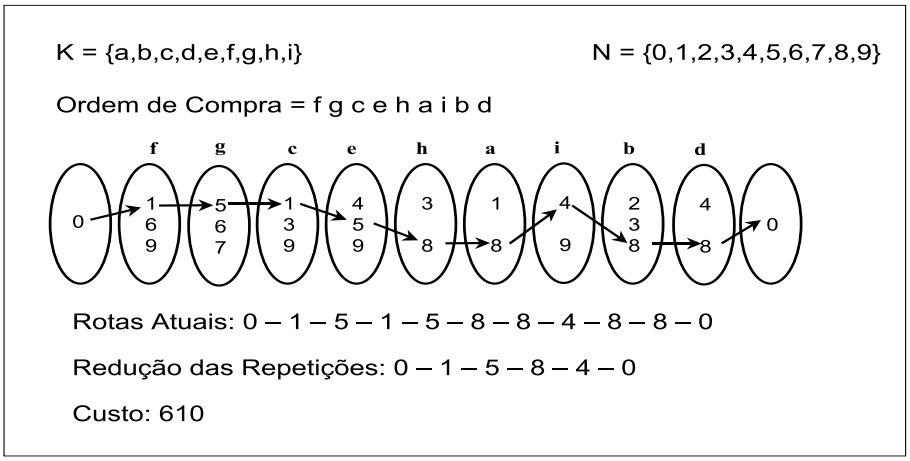


Figura 8: Passos do algoritmo Solução Inicial OrdemCompra.

Algoritmo Básico 11 : *Busca Local Add*

1 **enquanto** pode-se fazer economia adicionando nós **faça**
2 inserir na rota o nó mais econômico
fim enquanto

5.2.2 Busca Local Drop

Este algoritmo é o mesmo para a geração da solução inicial Drop 5.1.2, sendo a única diferença, que este algoritmo já parte de uma solução já construída.

Algoritmo Básico 12 : *Busca Local Drop*

1 **enquanto** pode-se fazer economia retirando nós **faça**
2 retirar da rota nó mais econômico
fim enquanto

5.2.3 Busca Local AddDrop

Este algoritmo consiste em inserir um elemento na rota e em seguida aplicar o algoritmo de busca local drop 5.2.2.

Algoritmo Básico 13 : *Busca Local AddDrop*

1 *para* $i \in$ nós diponíveis **faca**
2 *Adiciona i na rota*
3 *BuscaDrop*
fim para

5.2.4 Busca Local DropAdd

Este algoritmo consiste em retirar um elemento da rota e em seguida aplicar a busca local add 5.2.1.

Algoritmo Básico 14 : *Busca Local DropAdd*

1 *para* $i \in$ nós na rota **faça**
2 *Retira i da rota*
3 *BuscaAdd*
fim para

5.2.5 Busca Local Swap

Este algoritmo consiste em fazer trocas de posicionamento na rota atual, ele realiza permutações entre par de elementos na rota.

Algoritmo Básico 15 : *Busca Local Swap*

1 **para** i *de* 1 até MAX **faça**
2 Seleciona $No1$ e $No2$ aleatoriamente na rota;
3 **se** troca($No1$,$No2$) melhorar solução **então**
4 Executa a troca de $No1$ com $No2$
fim se
fim para

5.2.6 Busca Local Híbrida

Este algoritmo realiza uma Busca Local AddDrop 5.2.3 e Uma DropAdd 5.2.4, para em seguida realizar varia buscas Add5.2.1, Drop5.2.2 e Swap5.2.5.

Algoritmo Básico 16 : *Busca Local Híbrida*

```
1  BuscaAddDrop
2  BuscaDropAdd
3  repeat
4      BuscaDrop
5      BuscaAdd
6      BuscaSwap
   até não melhorou
```

5.2.7 Busca Local Vizinhanca

Este algoritmo realiza uma busca na vizinhança que dista de K elementos, o que neste caso, significa, K elementos a menos. Inicialmente cria-se um conjunto com K elementos escolhidos aleatoriamente na rota. Removem-se estes K elementos da rota, colocando-os como proibidos, ou seja, não podem pretencer a solução. Em seguida executa-se uma busca local (qualquer uma das anteriores) e agora disponibiliza os K elementos novamente, repetindo a busca local.

Algoritmo Básico 17 : *Busca Local Vizinhança*

```
1  para i de 1 até MAX faça
2      S é o conjunto formado por K nós da rota escolhidos aleatóriamente
3      Rota = Rota − S
4      Disponíveis = Disponíveis - S
5      QualquerBuscaLocal
6      Disponíveis = Disponíveis + S
7      QualquerBuscaLocal
   fim para
```

5.2.8 Busca Local VNS

Esta busca se baseada na Metaheurística *Variable Neiborhood Search*, onde temos a vizinhança dinâmica. A k-ésima vizinhança é pesquisada pelo algoritmo de busca 5.2.7. A Busca Local VNS, utiliza uma vizinhança pequena (k=1), aumenta a vizinhança (aumentando K), a cada passo em que a solução não é melhorada e caso melhore a solução, reduz-se a vizinhança para K=1.

Algoritmo Básico 18 : *Busca Local VNS*

```
1  K = 1
2  enquanto K < K_MAX faça
3     BuscaVizinhança;
4     se melhorou então
5        K = 1
      senão
6        K = K + 1
      fim se
   fim enquanto
```

6 Algoritmos Seqüenciais Propostos

Com os métodos de construção de solução inicial e os métodos de busca local, apresentados no capítulo anterior, podemos testá-los em várias metaheurísticas para verificar qual combinação fornece os melhores resultados. Os algoritmos Tabu+CSM e Tabu+REM [48, 49], foram retirados da literatura para servir como parâmetro de comparação, uma vez que é o algoritmo que obteve os melhores resultados para o problema do TPP, até o momento. Todos os demais algoritmos estão sendo propostos nesta tese.

Para os testes, foram criadas 36 instâncias apresentadas na tabela 1, onde o número de nós variou de 50 até 150, o número de itens também variou de 50 a 150 a oferta de itens por mercado variou de 1 a 5, o custo de distância variou de 10 a 300 e o custo dos produtos também variou de 10 a 300.

Cada algoritmo foi executado 5 vezes para cada instância durante 300 ou 600 segundos, em um computador IBM SP/2, dedicado exclusivamente à nossa aplicação

Os testes foram realizados com GRASP, VNS e Busca Tabu. A seguir veremos quais combinações foram feitas e os respectivos resultados obtidos.

Instância	Nº de Mercados	Nº de Itens	Items por Mercado	C. Itens	C. Dist.
1	50	50	1 - 1	100 - 300	100 - 300
2	50	50	2 - 5	100 - 300	100 - 300
3	50	50	2 - 5	10 - 30	100 - 300
4	50	50	2 - 5	100 - 300	10 - 30
5	50	100	1 - 2	100 - 300	100 - 300
6	50	100	2 - 5	100 - 300	100 - 300
7	50	100	2 - 5	10 - 30	100 - 300
8	50	100	2 - 5	100 - 300	10 - 30
9	50	150	1 - 3	100 - 300	100 - 300
10	50	150	2 - 5	100 - 300	100 - 300
11	50	150	2 - 5	10 - 30	100 - 300
12	50	150	2 - 5	100 - 300	10 - 30
13	100	50	1 - 1	100 - 300	100 - 300
14	100	50	2 - 5	100 - 300	100 - 300
15	100	50	2 - 5	10 - 30	100 - 300
16	100	50	2 - 5	100 - 300	10 - 30
17	100	100	1 - 1	100 - 300	100 - 300
18	100	100	2 - 5	100 - 300	100 - 300
19	100	100	2 - 5	10 - 30	100 - 300
20	100	100	2 - 5	100 - 300	10 - 30
21	100	150	1 - 2	100 - 300	100 - 300
22	100	150	2 - 5	100 - 300	100 - 300
23	100	150	2 - 5	10 - 30	100 - 300
24	100	150	2 - 5	100 - 300	10 - 30
25	150	50	1 - 1	100 - 300	100 - 300
26	150	50	2 - 5	100 - 300	100 - 300
27	150	50	2 - 5	10 - 30	100 - 300
28	150	50	2 - 5	100 - 300	10 - 30
29	150	100	1 - 1	100 - 300	100 - 300
30	150	100	2 - 5	100 - 300	100 - 300
31	150	100	2 - 5	10 - 30	100 - 300
32	150	100	2 - 5	100 - 300	10 - 30
33	150	150	1 - 1	100 - 300	100 - 300
34	150	150	2 - 5	100 - 300	100 - 300
35	150	150	2 - 5	10 - 30	100 - 300
36	150	150	2 - 5	100 - 300	10 - 30

Tabela 1: Instâncias usadas para os testes computacionais

6.1 GRASP

Para o GRASP, foram realizadas todas as combinações de construção de solução inicial com busca local, descritas no capítulo 5, resultando um total de 42 algoritmos. Como são muitos os algoritmos, o critério de parada foi inicialmente reduzido para 300 segundos. O resultado desta primeira bateria de teste, está na Tabela 2.

Na Tabela 2, foi considerada a melhor solução obtida por todos os métodos, aqui denominada de (MS). Assim os valores mostrados nesta tabela, mostram a distância média da solução final de cada algoritmo em relação a MS. O ideal seria comparar com a solução ótima, mas na falta desta foi utilizada a MS.

É importante salientar que a MS é obtida comparando-se todos os algoritmos: GRASP, VNS, GRASP+VNS, Busca Tabu e todas as paralelizações.

S.I. \ B.L.	Swap	Add	Drop	AddDrop	DropAdd	Híbrido
Add	2.01	2.01	1.69	1.18	1.27	1.07
Drop	2.80	2.56	2.78	2.27	1.99	1.77
AddAleatório	1.23	1.25	0.43	0.29	0.31	0.20
DropAleatório	3.32	1.82	3.87	1.34	1.14	0.96
AddGeni	1.36	1.38	1.02	0.51	0.42	0.50
DropGeni	2.92	2.45	2.97	1.87	2.03	1.94
OrdemCompra	14.52	21.18	9.87	7.56	3.67	1.98

Tabela 2: Resultados médios obtidos pelos algoritmos GRASP Seqüenciais. .Todos os valores representam a porcentagem do erro médio em relação a MS

Podemos observar que as buscas locais Swap, Add e Drop não produziram bons resultados e que as soluções iniciais OrdemCompra, Drop e Add também não produziram bons resultados.

Os algoritmos de busca local: Add, Drop e Swap, são simples e acabam sendo executados pelas outras buscas, o que explica o seu baixo desempenho, comparado com as outras buscas aqui propostas.

Os algoritmos de solução inicial Add e Drop, não obtiveram bons resultados porque, geram sempre a mesma solução inicial, não sendo adequados, para usar com o GRASP, uma vez que a cada iteração precisa-se gerar uma nova solução inicial diferente das soluções iniciais anteriores.

O algorimo de solução inicial OrdemCompra, não obteve bons resultados possivelmente porque ele gera soluções iniciais muito distantes dos ótimos locais, fazendo com que a fase de busca local fique mais lenta, exigindo um tempo maior para obter melhores resultados.

Após a primeira bateria de testes, verificou-se que algumas combinações não obtiveram bons resultados na média e portanto foram descartadas. Uma nova bateria de testes, agora com o critério de parada estendido para 600 segundos, foi

realizado com os algoritmos remanescentes.

Algoritmo	Erro Médio (%)	Tempo (s)	Total de Acertos (%)
AddAleatório Híbrido	0.09	340.72	52.78
AddGeni Híbrido	0.16	343.75	47.22
AddAleatório DropAdd	0.14	298.00	41.67
AddGeni DropAdd	0.25	324.86	44.44
AddAleatório AddDrop	0.15	334.72	30.56

Tabela 3: Os Desempenho médio dos 5 melhores algoritmos GRASP Seqüenciais.

A Tabela 3 mostra os resultados médios dos 5 melhores algoritmos, onde o Erro Médio, é a porcentagem de erro médio entre a solução obtida e a MS, Total Acertos é o percentual de MS atingidos e Tempo é o tempo médio exigido pelo algoritmo em segundos, para chegar a melhor solução.

Assim os valores da primeira linha, terceira e quarta coluna da Tabela 3, significam que o algoritmo GRASP com construção AddAleatório e busca local Híbrido, atingiu a MS em 52,78% dos casos, exigindo em tempo médio de 340,72 segundos para obter a Melhor Solução, que em 52,78% dos caso foi a MS.

6.2 VNS

Além do algoritmo VNS básico, também foram implementas duas variações, a primeira que é o VNDS [29] onde na vizinhança k fixam-se N-k variáveis, e a segunda chamada de VNIDS (*Variable Neighborhood Inverse Decomposition Search*), proposta nesta tese, é uma variação do VNDS onde na vizinhança k fixam-se k variáveis. Esta proposta parece interessante porque, deve-se realizar todas as possíveis k fixações de variáveis, que é bem menor do que as N-k variações do VNDS, um vez que k normalmente não ultrapassa o valor 7.

Para o VNS e variantes, foram realizadas todas as combinações possíveis entre todos os algoritmos de solução inicial e os seguintes algoritmos de busca local:

AddDrop, DropAdd e Híbrido, resultando num total de 63 algoritmos. O resultado desta primeira bateria de testes é mostrada nas Tabelas 4, 5 e 6, onde o critério de parada foi de 300 segundos.

S.I. \ B.L.	AddDrop (%)	DropAdd (%)	Híbrido (%)
Add	0.46	0.32	0.33
Drop	0.35	0.44	0.39
AddAleatório	0.34	0.37	0.36
DropAleatório	0.40	0.51	0.51
AddGeni	0.28	0.23	0.22
DropGeni	1.56	1.65	1.34
OrdemCompra	1.80	0.73	0.81

Tabela 4: Resultados médios obtidos pelos algoritmos VNS Seqüenciais.

A mesma bateria de testes efetuada nos algoritmos da Tabela 4, foi usada para as variantes do VNS (Tabelas 5 e 6).

S.I. \B.L.	AddDrop (%)	DropAdd (%)	Híbrido (%)
Add	1.17	1.18	1.19
Drop	1.77	1.96	1.81
AddAleatório	1.23	1.16	1.11
DropAleatório	2.20	1.48	1.86
AddGeni	0.98	0.99	1.02
DropGeni	2.15	2.27	2.53
OrdemCompra	9.16	2.30	2.67

Tabela 5: Resultados médios obtidos pelos algoritmos VNDS Seqüenciais.

S.I. \B.L.	AddDrop (%)	DropAdd (%)	Híbrido (%)
Add	1.17	1.10	1.17
Drop	2.23	1.91	2.25
AddAleatório	1.43	0.99	1.06
DropAleatório	2.05	1.79	1.74
AddGeni	1.23	0.97	1.13
DropGeni	2.33	2.20	2.11
OrdemCompra	9.52	2.65	2.65

Tabela 6: Resultados médios obtidos pelos algoritmos VNDSI Seqüenciais.

Podemos observar pelos resultados médios das Tabelas 5 e 6, que as variantes do VNS, não obtiveram bons resultados, porque, ao fixarmos as K ou N-

K variáveis, faz-se um busca exaustiva nesta vizinhança, este processo é lento e a cada vizinhança se torna mais lento, não permitindo uma exploração adequada das vizinhanças distantes, no tempo computacional aqui adotado.

Para o VNS e suas variações, os algoritmos de Solução inicial Add e Drop, conseguiram bons resultados uma vez que são utilizados uma única vez. O Algoritmo OrdemCompra, continuou obtendo resultados distantes dos demais, uma vez que, gera soluções iniciais muito distantes dos ótimos locais.

Após a primeira bateria de testes, vericou-se que algumas combinações não obtiveram bons resultados e portanto foram descartadas das análises seguintes. Para a próxima etapa ficaram somente os algoritmos que utilizam o VNS básico, sendo as suas variantes descartadas. A Tabela 7 mostra a segunda bateria de testes, agora com critério de parada de 600 segundos.

S.I. \B.L.	AddDrop (%)	DropAdd (%)	Híbrido (%)
Add	0.20	0.20	0.20
Drop	0.23	0.28	0.11
AddAleatório	0.31	0.18	0.18
DropAleatório	0.22	0.22	0.16
AddGeni	0.11	0.20	0.21
DropGeni	0.24	0.33	0.19
OrdemCompra	0.87	0.54	0.42

Tabela 7: Resultados médios obtidos pelos melhores algoritmos VNS Seqüenciais.

Na Tabela 8 são apresentados os resultados dos 5 melhores algoritmos VNS, onde o erro, é a porcentagem de erro médio entre a solução obtida e a MS. A quarta coluna, indica a porcentagem de MS atigindas e Tempo é o tempo médio que o algoritmo leva para chegar a melhor solução.

Mais uma vez lembramos que os resultados dos algoritmos VNS, levaram em conta a MS, que é obtida comparando-se todos os algoritmos: GRASP, VNS,

Algoritmo	Erro Médio (%)	Tempo (s)	Total de Acertos (%)
DropGeni Híbrido	0.19	318.56	47.22
Drop Híbrido	0.11	324.97	44.44
AddGeni Híbrido	0.11	346.28	36.11
DropAleatório Híbrido	0.16	247.58	41.67
AddAleaório Híbrido	0.18	320.94	36.11

Tabela 8: Resultado médio dos 5 melhores algoritmos VNS seqüenciais.

GRASP+VNS, Busca Tabu e todas as paralelizações.

6.3 GRASP+VNS

Estes algoritmos, reúnem conceitos do GRASP com o VNS, onde temos basicamente um algoritmo GRASP, onde a busca local GRASP é um VNS, ou temos um VNS que é inicializado a partir de varios pontos diferentes.

Para estes algoritmos, foram realizadas todas as combinações possíveis entre os algoritmos de construção de solução inicial: AddAleatório, DropAleatório, AddGeni e DropGeni e os seguintes algoritmos de busca local: AddDrop, DropAdd e Híbrido, resultando num total de 12 algoritmos. Entretanto, usamos todos algoritmos VNS desenvolvidos, o que aumentou o total de algoritmos para 36.

As Tabelas 9, 10 e 11 mostram os resultados médios da primeira bateria de testes, onde o critério de parada foi de 300 segundos.

S.I. \B.L.	AddDrop (%)	DropAdd (%)	Híbrido (%)
AddAleatório	0.25	0.25	0.31
DropAleatório	0.43	0.50	0.51
AddGeni	0.37	0.42	0.23
DropGeni	1.20	1.00	1.28

Tabela 9: Resultados médios obtidos pelos algoritmos GRASP+VNS Seqüenciais.

A partir das Tabelas 10 e 11, podemos observar que as variantes do VNS, continuaram obtendo resultados distantes do VNS. O que era previsível, diante das simulações anteriores.

S.I. \B.L.	AddDrop (%)	DropAdd (%)	Híbrido (%)
AddAleatório	1.14	1.05	1.05
DropAleatório	3.22	3.36	2.93
AddGeni	1.20	1.34	1.15
DropGeni	2.72	2.54	2.67

Tabela 10: Resultados médios obtidos pelos algoritmos GRASP+VNDS Seqüenciais.

S.I. \B.L.	AddDrop (%)	DropAdd (%)	Híbrido (%)
AddAleatório	1.11	1.06	1.19
DropAleatório	2.61	2.60	2.57
AddGeni	1.13	1.30	0.98
DropGeni	2.61	3.15	2.37

Tabela 11: Resultados médios obtidos pelos algoritmos GRASP+VNDSI Seqüenciais.

Após a primeira bateria de teste, verificou-se que algumas combinações obtiveram bons resultados e portanto devem ser melhor analisadas. Para a próxima etapa ficam somente os algoritmos que utilizam o VNS básico, sendo as suas variantes descartadas. A Tabela 12, mostra os resultados dos algoritmos GRASP+VNS, com critério de parada de 600 segundos.

S.I. \B.L.	AddDrop (%)	DropAdd (%)	Híbrido (%)
Add	0.32	0.19	0.09
Drop	0.27	0.42	0.24
AddAleatório	0.21	0.08	0.09
DropAleatório	0.33	0.28	0.19
AddGeni	0.17	0.17	0.04
DropGeni	0.48	0.40	0.32
OrdemCompra	1.10	0.52	0.34

Tabela 12: Resultados médios obtidos pelos melhores algoritmos GRASP+VNS Seqüenciais.

Na Tabela 13 são apresentados os 5 melhores algoritmos GRASP+VNS, onde temos: o erro percentual em relação a MS, o percentual de obtenção da MS e o tempo médio para atingir a melhor solução.

Mais uma vez é importante salientar que os resultados dos algoritmos GRASP+VNS, levaram em conta a MS, que é obtida comparando-se todos os algorit-

Algoritmos	Erro Médio (%)	Tempo (s)	Total de Acertos (%)
AddGeni Híbrido	0.04	327.31	63.89
AddAleaório Híbrido	0.09	279.50	55.56
AddAleaório DropAdd	0.08	249.92	52.78
AddGeni DropAdd	0.17	351.33	50.00
Add Híbrido	0.09	274.81	33.33

Tabela 13: Resultado médio dos 5 melhores algoritmos GRASP+VNS Seqüenciais.

mos: GRASP, VNS, GRASP+VNS, Busca Tabu e todas as paralelizações.

6.4 Busca Tabu

Com o objetivo de avaliar a qualidade das soluções obtidas pelos algoritmos aqui propostos e devido à inexistência ao menos de nosso conhecimento, de bibliotecas de testes para o TPP, implementamos algumas versões do algoritmo de Busca Tabu para o TPP que até o momento tinham produzido, segundo seus autores, os melhores resultados para este problema.

Para a Busca Tabu foram implementados três algoritmos para a gerencia da lista Tabu. O primeiro é o algoritmo básico, onde usa-se uma lista de movimentos e proíbe-se o movimento contrário. A segunda é uma adaptação do algoritmo *Cancelation Sequence Method* ou CSM proposto por Voss [49] e o terceiro também é uma adaptação de um algoritmo proposto por Voss [49], o *Reverse Elimination Method* (REM). Estes algoritmos foram escolhidos por terem obtido os melhores resultados na literatura até o momento.

Os algorimos de busca local foram baseados nos algoritmos de Voss [49]. Para a construção da solução inicial, usamos os algoritmos Add e Drop [48], para a busca local foram utilizados os dois algoritmos propostos por Voss que são AddTabu e DropTabu [48], e uma proposta que é uma variação do algoritmo de busca local Híbrido, sendo chamado de Tabu Híbrido. Esta variação foi necessária por que a busca

deve considerar, o critério de aspiração e os movimentos que são Tabu.

Além da comparação entre os varios tipos de gerencias de listas, também verificou-se qual o ganho de se utilizar a técnica de intensificação e diversificação na Busca Tabu.

Portanto para os testes, temos, três variações para a gerencia da lista Tabu, dois algoritmos para a geração da solução inicial e três algoritmos para a busca local. Além dos testes com intensificação e diversificação. Portanto temos um total de 72 algoritmos. Nos teste o critério de aspiração utilizado foi se o movimento Tabu gerar uma solução melhor que a melhor solução, então o movimento deixa de ser Tabu.

O resultado da primeira bateria de testes onde não usamos intensificação e nem diversificação é mostrado na tabela 14, usando o mesmo conjunto de testes usados nos outros algoritmos: GRASP, VNS e GRASP+VNS.

S.I. \B.L.		Add			Drop	
Gerencia	AddTabu	DropTabu	TabuHíbrido	AddTabu	DropTabu	TabuHíbrido
Simples	1.09	1.46	1.04	2.38	1.76	1.26
CSM	1.00	1.14	1.04	2.03	1.51	1.50
REM	0.99	1.04	1.04	2.02	0.77	1.57

Tabela 14: Resultados médios obtidos pelos algoritmos de Busca Tabu Seqüenciais. Todos os valores representam a porcentagem do erro médio em relação a MS.

Após estes resultados verificamos que a busca local DropTabu com a solução inicial Add e a busca local AddTabu com a solução inicial Drop, não obtiveram bons resultados nos testes realizados.

Com os resultados médios da Tabela 14, selecionamos os melhores algoritmos e avaliamos o uso da intensificação e diversificação, o qual pode ser visto na Tabela 15.

A partir da Tabela 15, podemos verificar que a intensificação e a diversificação conseguem melhorar a solução final, observamos ainda que na média a diversi-

S.I.	B.L.	Gerencia	normal	Intensif.	Diversif.	Inten.+ Diver.
Drop	DropTabu	REM	1.17	1.32	0.32	0.41
Add	DropTabu	CSM	1.13	0.95	0.37	0.27
Add	AddTabu	CSM	1.00	1.03	0.47	0.44
Add	TabuHíbrido	Simples	1.03	1.03	0.54	0.62
Drop	TabuHíbrido	REM	1.83	1.43	0.80	0.80

Tabela 15: Avaliação das fases de Intensificação e Diversificação dos algoritmos Tabu Seqüenciais: Resultados médios obtidos. Todos os valores representam a porcentagem do erro médio em relação a MS.

ficação oferece um ganho mais significativo, do que a intensificação e que utilizando-se as duas técnicas conseguimos melhorar mais a qualidade da solução.

S.I.	B.L.	Gerencia	Estratégia	E. Médio (%)	Tempo (s)	T. Acertos (%)
Drop	DropTabu	REM	Int. + Div.	0.41	234.69	50.00
Add	AddTabu	CSM	Int. + Div.	0.44	234.19	47.22
Add	DropTabu	CSM	Int. + Div.	0.27	302.72	41.67
Add	DropTabu	CSM	Div.	0.37	239.47	33.33
Drop	DropTabu	REM	Div.	0.33	294.86	33.33

Tabela 16: Os 5 Melhores Algoritmos de Busca Tabu Seqüenciais.

Na Tabela 16 são apresentados os 5 melhores algoritmos de Busca Tabu, onde temos: o erro percentual médio em relação a MS, o percentual de obtenção da MS e o tempo médio para atingir a melhor solução.

6.5 Resumo dos Resultados dos Algoritmos Seqüenciais

Na Tabela 17, são mostrados os melhores algoritmos seqüencias, onde são comparados os algoritmos GRASP, os VNS e suas variantes e os algoritmos de Busca Tabu aqui propostos.

A Tabela 17 mostra o resultado dos 5 melhores algoritmos, onde o Erro Médio, é a porcentagem de erro médio entre a solução obtida e MS, o Total de Acertos, indica quantas vezes esse algoritmo chegou a MS e Tempo é o tempo médio para chegar a melhor solução.

A partir da Tabela 17, podemos concluir que os algoritmos GRASP+VNS, obtiveram os melhores resultados médios, o que indica, ser um cam-

Algorithm	S.I.	B.L.	E. Médio (%)	Tempo (s)	T. Acertos (%)
GRASP	AddAleatório	Híbrido	0.09	340.72	52.78
GRASP	AddGeni	Híbrido	0.16	343.75	47.22
VNS	Drop	Híbrido	0.11	324.97	44.44
VNS	DropGeni	Híbrido	0.19	318.56	47.22
GRASP+VNS	AddGeni	Híbrido	0.04	327.31	63.89
GRASP+VNS	AddAleatório	Híbrido	0.09	279.50	55.56
Tabu+CSM	Add	DropTabu	0.27	302.72	41.67
Tabu+REM	Drop	DropTabu	0.41	234.69	50.00

Tabela 17: Melhores Algoritmos Seqüenciais.

inho promissor, o desenvolvimento de heurísticas híbridas. Esses resultados parciais nos mostram também que tanto os algoritmos GRASP, como os VNS mais principalmente as versões híbridas que conjungam conceitos de GRASP e VNS, conseguem se equiparar as versões mais sofisticadas de Busca Tabu existentes na literatura utilizando gerencia dinâmica da lista tabu e incluindo fases de intensificação e diversificação como proposto por Voss [49].

7 Algoritmos Paralelos Propostos

Algumas propostas de paralelização para as metaheurísticas GRASP, Tabu Search e VSN, são apresentadas a seguir.

Os testes foram executados em um computador IBM SP/2 com seis processadores idênticos, dedicados exclusivamente à nossa aplicação. O critério de parada foi o número máximo de iterações. Com os testes computacionais, podemos avaliar o *speedup* dos algoritmos paralelos, bom como se há melhora na qualidade da solução.

7.1 Paralelização GRASP

No Algoritmo GRASP, as iterações são independentes umas das outras, o que facilita o uso da estratégia múltiplos percursos independentes, onde as iterações são divididas entre os processadores. A divisão do número de iterações entre os processos pode ser analisada como um problema de balanceamento de carga. Para balancear o número de iterações entre os processos temos duas estratégias que são: Balanceamento Estático e Balanceamento Dinâmico. Portanto, serão realizadas comparações entre essas estratégias.

7.1.1 Balaceamento Estático

No balanceamento estático, o número de iterações que cada processo irá executar é determinado antes da execução. Para os algoritmos propostos, cada

processo executa $\frac{numero\,de\,iteracoes\,sequencial}{numero\,de\,processos}$ iterações, ou seja, o número de iterações do algoritmos seqüencial correspondente, foi dividido igualmente pelos processos. Cada processo então executa independentemente dos demais processos.

Adotamos duas técnicas no balanceamento estático, na primeira estratégia o mesmo algoritmo é executado em todos os processos. Na segunda estratégia algoritmos diferentes são executados nos diferentes processos. A diferença dos algoritmos está nos diferentes critérios adotados para a geração de solução inicial e busca local.

Portanto no balanceamento estático foram desenvolvidos dois algoritmos: um de *percurso único passo único* e outro de *múltiplos percursos independentes*.

Algoritmo 19 :*percurso único passo único*

1 **para** i **de** 1 **até** $\frac{numero\,de\,iteracoes\,sequencial}{numero\,de\,processos}$ **faça**
2 executa algotimo GRASP
fim para

Algoritmo 20 :*múltiplos percursos independentes*

1 Configura algoritmo GRASP de acordo com a identificação do processo.
2 **para** i **de** 1 **até** $\frac{numero\,de\,iteracoes\,sequencial}{numero\,de\,processos}$ **faça**
3 executa algoritmo GRASP.
fim para

No algoritmo 20, a escolha do algoritmo GRASP a ser executado é feita na linha 1. Essa escolha se baseia na identificação do processo e no desempenho dos algoritmos seqüenciais.

7.1.2 Balanceamento Dinâmico

Neste tipo de balanceamento, o número de iterações, que cada processo vai executar é determinado durante a execução. Este tipo de balanceamento visa evitar

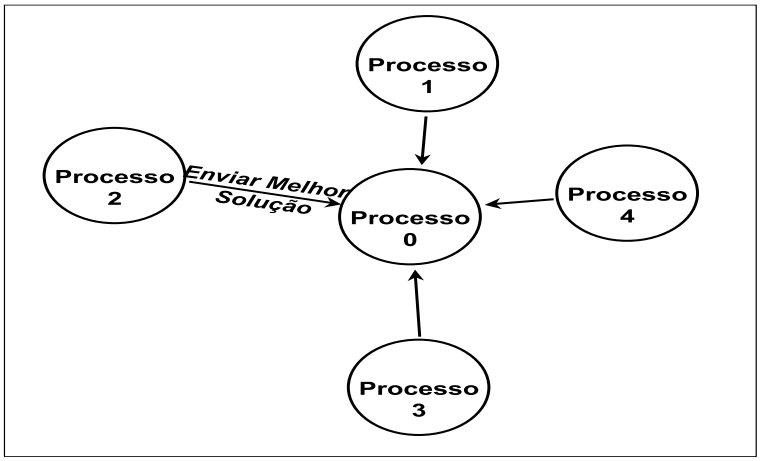


Figura 9: Diagrama do balanceamento estático.

que um processo lento, atrase a execução da aplicação toda, ou seja, tenta aproveitar melhor os processos, dando mais iterações para os processos mais rápidos e poucas iterações para os processos lentos.

Foram implementadas duas técnicas de balanceamento dinâmico, que são: Mestre-escravo e distribuído. Todas as técnicas implementadas são assíncronas, por serem menos exploradas na literatura.

A primeira técnica mestre-escravo, consiste em designar um processo como organizador das iterações, chamado de mestre, este não executa o algoritmo GRASP, dedicando-se a controlar o número de iterações. Os demais processos são chamados de escravos, veja a figura 10.

No início da execução todos os processos escravos começam com um número fixo de iterações, que deve ser pequeno. Quando um processo completa as iterações iniciais, ele informa ao mestre que acabou e espera a resposta do mestre, informando o término da aplicação ou atribuindo-lhe mais iterações. Para fins de comparação o número de iterações do mestre é inicialmente igual ao número de iterações do algoritmo seqüencial. Quando o mestre verifica que algum escravo está ocioso, ele atribui mais algumas iterações para esse escravo, e incrementa o número de iterações

alocadas para os escravos. Quando o número de iterações for máximo o algoritmo pára, o critério de parada usado foi o número máximo de iterações .

Algoritmo 21 : *Mestre-Escravo (Escravo)*

```
1  enquanto critério de parada não satisfeito faça
2      envia pedido de iteração para o mestre.
3      recebe iterações do mestre.
4      para i de 1 até iterações faça
5          executa Algoritmo GRASP.
       fim para
   fim enquanto
```

Algoritmo 22 : *Mestre-Escravo (Mestre)*

```
1  enquanto existe iterações faça
2      recebe pedido de iterações do escravo.
3      envia iterações para o escravo.
   fim enquanto
```

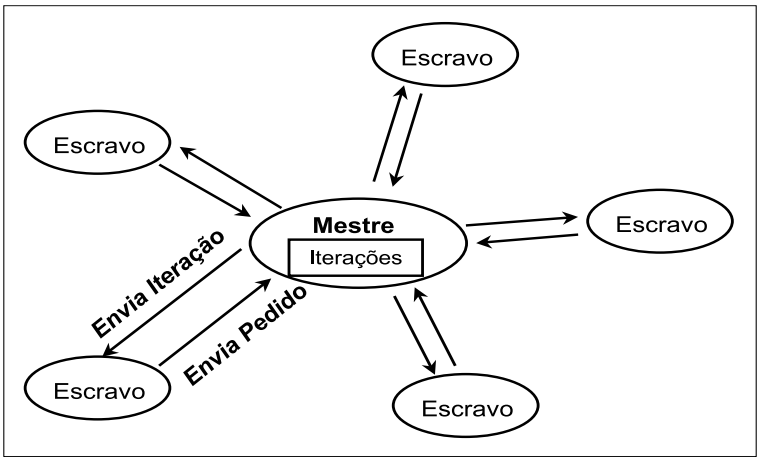


Figura 10: Diagrama do balanceamento dinâmico (mestre-escravo).

Na segunda técnica, distribuído, para cada processo P é criado um outro processo irmão chamado Q, que controlará o número de iterações do processo P. O

processo Q, passará a maior parte do tempo ocioso, portanto ele concorrerá muito pouco, com o processo P, pelo processador local.

Inicialmente as iterações são divididas igualmente entre os processos Q. O processo P, executa o algoritmo para um certo número pequeno de iterações, ao fim destas, ele solicita mais iterações ao irmão Q, este então verifica se tem mais iterações para o P. Se tiver, ele atribui mais iterações ao irmão e volta a ficar esperando. Caso não tenha mais iterações, o processo Q realiza um pedido para qualquer um dos outros processos $Q's$ disponíveis, escolhido aleatoriamente. Este processo então verifica se tem iteração para doar, se tiver ele envia para o processo Q que pediu. Se não tiver, ele envia uma mensagem informando que não tem iterações para doar. O processo Q original ao receber a confirmação e o número de iterações passa para o processo irmão P, o número de iterações. Se o processo Q original, receber uma resposta negativa, ele marca este processo Q como indisponível e escolhe outro processo Q disponível e realiza os mesmos procedimentos. Caso não existam mais processos Q disponíveis, ele inicia o processo de terminação.

Existem na literatura muitos algoritmos para a detecção de terminação de algoritmos distribuídos, tais como os algoritmos propostos por Huang[23] e Dijkstra e Scholten[10]. O algoritmo utilizado é baseado nestes algoritmos, onde um processo Q inicia a terminação quando detecta que todos os outros processos Q não tem mais iterações para doar, então este processo envia uma mensagem de terminção para todos os outros processos Q's e espera a resposta(mensagem de terminação) dos mesmos, para poder terminar. Entretanto, antes de terminar, cada processo Q precisa terminar o irmão P. Para terminar o irmão P, o processo Q precisa saber que iniciou a terminação dos processos Q's e que o processo P não enviará mais mensagens de pedido, isso é

garantido quando o processo Q recebe uma mensagem de pedido do processo P e não existe mais nenhum Q disponível.

Algoritmo Paralelo 23 : *Dinâmico Distribuido (Q)*

```
1   fim = falso
2   fimP = falso
3   enquanto não fim ou não fimP faça
4      recebe MSG
5      caso MSG = PEDIDO
6         se Iterações Restantes = 0 então
7            se origem da MSG = Irmão P então
8               se exite algum Q disponível então
9                  envia MSG = PEDIDO para qualquer Q disponível
                senão
10                 envia MSG = TERMINO para o irmão P
11                 fimP = verdadeiro
                fim se
            senão
12             Envia MSG = NÃO ATENDIMENTO para origem
            fim Se
         senão
13          envia MSG = ATENDIMENTO, com o número de iterações, para
            o solicitante (origem)
14          marca origem como indisponível
         fim se
15    caso MSG = ATENDIMENTO
16       repassa MSG = ATENDIMENTO para o irmão P
17    caso MSG = NÃO ATENDIMENTO
18       marca origem como indisponível
19       se todos os Q's estão indisponívies então
20          envia MSG = TERMINO para todos os Q's
         senão
21          envia MSG = PEDIDO para qualquer Q disponível
         fim se
22    caso MSG = TERMINO
23       após receber a mensagem de término de todos os Q's fim = ver-
         dadeiro
   fim enquanto
```

Algoritmo Paralelo 24 : *Dinâmico Distribuido (P)*

```
1   fim = falso
2   enquanto não fim faça
3      caso MSG = ATENDIMENTO
4         para i de 1 até iterações faça
5            executa algoritmo GRASP
          fim para
6         envia MSG = PEDIDO ao Q
7      caso MSG = TERMINO
8         termina a execução deste processo P
9         fim = verdadeiro
    fim enquanto
```

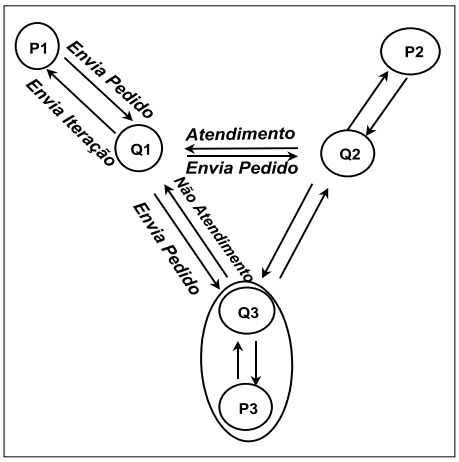


Figura 11: Diagrama do balanceamento dinâmico distribuído.

Como os processos irmãos concorrem pouco pela CPU, e têm trocas de mensagem entre eles, o ideal, é realizar a troca de informações por memória compartilhada, a fim de minimizar o tempo, para isso eles devem ficar no mesmo processador. Para criarmos o processo Q no mesmo processador que o processo P, usou-se o comando *fork*. Assim temos a garantia de minimizarmos o tempo com as trocas de mensagem entre os processo irmãos.

Os processos executados por cada processador e a comunicação entre eles pode ser visualizada na figura 11, onde cada par de processos P_i e Q_i compartilha o

mesmo processador.

7.1.3 Resultados Computacionais

Para o teste de todos os algoritmos paralelos aqui propostos, usamos duas instâncias, e executamos os algoritmos seqüenciais correspondentes por 500 iterações e os algoritmos paralelos também usamos como critério de parada 500 iterações. Com este critério de parada podemos avaliar o *speedup* dos algoritmos paralelos bem como se há melhora na qualidade da solução em relação a versão seqüencial correspondente.

A primeira instância S1, corresponde à instância 19, contém 100 nós e 100 itens e custo de itens varia de 10 a 30, o custo de distância entre nós varia de 100 a 300 e a quantidade de itens por nó varia de 2 a 5.

A segunda instância S2, corresponde à instância 23, contém 100 nós e 150 itens e custo de itens varia de 10 a 30, o custo de distância entre nós varia de 100 a 300 e a quantidade de itens por nó varia de 2 a 5.

Estas duas instâncias foram escolhidas por serem as que obtiveram os piores resultados na execução seqüencial.

Nos primeiros testes utilizamos os seguintes algoritmos:

1. Seq: Algoritmo GRASP seqüencial, usando para a solução inicial o procedimento AddAleatório e para a busca local o procedimento Híbrido.

2. ParInd: Paralelização do algoritmo seqüencial, utilizando a estratégia de Múltiplos percursos independentes.

3. ParDist: Paralelização do algoritmo seqüencial, utilizando a estratégia de Múltiplos percursos interativos, ou seja, assíncrono e com troca de informação para realizar o balanceamento de iterações.

4. ParMs: Paralelização mestre-escravo do algoritmo seqüencial, utilizando a estratégia de Múltiplos percursos interativos, com balanceamento de iterações.

Algorithm	Custo S1	Custo S2	Tempo S1	Tempo S2	Speedup
Seq	5700.98	7112.37	1316	1720	
ParInde	5693.50	7111.47	257	335	5.13
ParDist	5682.78	7112.13	278	363	4.73
ParMS	5693.32	7118,75	312	409	4.20

Tabela 18: Resultados dos algoritmos GRASP AddAleatório paralelos.

Mais uma bateria de testes foi realizado com os seguintes algoritmos:

1. Seq: Algoritmo GRASP seqüencial, usando para a solução inicial o procedimento AddGeni e para a busca local o procedimento Híbrido.

2. ParInd: Paralelização do algoritmo seqüencial, utilizando a estratégia de Múltiplos percursos independentes.

3. ParDist: Paralelização do algoritmo seqüencial, utilizando a estratégia de Múltiplos percursos interativos, com balanceamento de iterações.

4. ParMs: Paralelização mestre-escravo do algoritmo seqüencial, utilizando a estratégia de Múltiplos percursos interativos, com balanceamento de iterações.

Algorithm	Custo S1	Custo S2	Tempo S1	Tempo S2	Speedup
Seq	5718.37	7116.42	3216	4300	—
ParInde	5719.39	7119.78	620	825	5.20
ParDist	5718.76	7119.72	653	894	4.86
ParMS	5712.44	7119,98	741	992	4.33

Tabela 19: Resultados dos algoritmos GRASP AddGeni paralelos.

A partir dos resultados obtidos concluímos, que a paralelização reduziu bastante o tempo computacional sem prejudicar a qualidade da solução. A diferença de tempo entre os algoritmos paralelos se deve ao fato dos testes, serem feitos em uma

rede homogênea com dedicação exclusiva e de que as buscas locais não foram suficientes para gerar cargas diferenciadas nos processadores, a fim de justificar um balanceamento dinâmico de iterações. Portanto, os algoritmos que faziam o balanceamento dinâmico, gastaram tempo com um balanceamento que era afinal desnecessário. Em especial o ParMS, que utiliza um mestre, perde um processador que fica dedicado a controlar as iterações, entretanto, se o número de processadores for aumentado esta perda não será percebida.

7.2 Paralelização do VNS

As técnicas de paralelização empregadas no VNS, são análogas às empregadas na paralelização do algoritmo GRASP. Assim, na primeira técnica que emprega balanceamento estático, cada processo executa de forma independente.

Algorithm 25 :*percurso único passo único*

1 **para** i **de** 1 **até** $\frac{numero\,de\,iteracoes\,sequencial}{numero\,de\,processos}$ **faça**
2 executa algoritmo VNS
end for

Algoritmo 26 :*múltiplos percursos indenpendentes*

1 Configura algoritmo VNS de acordo com a identificação do processo.
2 **para** i **de** 1 **até** $\frac{numero\,de\,iteracoes\,sequencial}{numero\,de\,processos}$ **faça**
3 executa algotimo VNS.
fim para

Na segunda técnica, mestre-escravo, que emprega balanceamento dinâmico, o processo mestre além de distribuir as iterações, mantém uma lista das melhores soluções. Esta lista é atualizada a cada novo pedido de iteração. Os processos escravos,

partem de uma solução, inicialmente gerada por algum método de construção, para então realizar a busca VNS. Quando um escravo pede mais iteração para o mestre, ele envia uma mensagem de pedido contendo também a melhor solução obtida, para que o mestre atualize a lista de melhores soluções. Ao receber mais iterações, o escravo recebe também, uma nova solução inicial, escolhida aleatoriamente da lista de melhores.

Algoritmo 27 :*Mestre-Escravo (Escravo)*

```
1  enquanto critério de parada não satisfeito faça
2      envia pedido de iteração e a melhor solução para o mestre.
3      recebe iterações e a nova solução inicial do mestre.
4      para i de 1 até iterações faça
5          executa Algoritmo VNS.
       fim para
   fim enquanto
```

Algoritmo 28 :*Mestre-Escravo (Mestre)*

```
1  enquanto existe iterações faça
2      recebe pedido de iterações e melhor solução do escravo.
3      atualiza a lista de melhores soluções.
4      envia iterações e um solução escolhida aleatoriamente da lista de mel-
       hores soluções, para o escravo.
   fim enquanto
```

A terceira paralelização, é uma variação da segunda, sem o mestre. Para cada processo P, cria-se um processo irmão Q, que controla as iterações e gerencia a lista de melhores soluções.

Algoritmo Paralelo 29 : *Dinâmico Distribuido (Q)*

```
1   fim = falso
2   fimP = falso
3   enquanto não fim ou não fimP faça
4      recebe MSG
5      caso MSG = PEDIDO
6         atualiza a lista de melhores soluções
7         se Iterações Restantes = 0 então
8            se origem da MSG = Irmão P então
9               se exite algum Q disponível então
10                 envia MSG = PEDIDO,juntamente com lista de melhores
                   soluções, para qualquer Q disponível
                senão
11                 envia MSG = TERMINO para o irmão P
12                 fimP = verdadeiro
                fim se
             senão
13              Envia MSG = NÃO ATENDIMENTO para origem
             fim Se
          senão
14           envia MSG = ATENDIMENTO, com o número de iterações e uma
             solução escolhida aleatóriamente da lista de melhores soluções,
             para o solicitante (origem).
15           marca origem como indisponível.
          fim se
16     caso MSG = ATENDIMENTO
17        repassa MSG = ATENDIMENTO para o irmão P
18     caso MSG = NÃO ATENDIMENTO
19        marca origem como indisponível
20        se todos os Q's estão indisponívies então
21           envia MSG = TERMINO para todos os Q's
          senão
22           envia MSG = PEDIDO, juntamente com lista de melhores
             soluções, para qualquer Q disponível
          fim se
23     caso MSG = TERMINO
24        após receber a mensagem de término de todos os Q's fim = ver-
          dadeiro
   fim enquanto
```

Algoritmo Paralelo 30 : *Dinâmico Distribuido (P)*

```
1  fim = falso
2  enquanto não fim faça
3     caso MSG = ATENDIMENTO
4        para i de 1 até iterações faça
5           executa algoritmo VNS
         fim para
6        envia MSG = PEDIDO com a melhor solução ao Q
7     caso MSG = TERMINO
8        termina a execução deste processo P
9        fim = verdadeiro
   fim enquanto
```

Utilizou-se o comando *fork* para a criação do processo Q, para que a comunicação entre P e Q seja feita através de memória compartilhada.

7.2.1 Resultados Computacionais

Paralelizamos 2 variações dos algoritmos GRASP+VNS, uma vez que estes algoritmos obtiveram os melhores resultados médios, nos testes seqüencias.

Nos primeiros testes utilizamos os seguintes algoritmos:

1. Seq: Algoritmo GRASP+VNS seqüencial, usando para a solução inicial o procedimento DropGeni e para a busca local o procedimento Híbrido.

2. ParInd: Paralelização do algoritmo seqüencial, utilizando a estratégia de Múltiplos percursos independentes.

3. ParMS: Paralelização mestre-escravo do algoritmo seqüencial, utilizando a estratégia de Múltiplos percursos interativos, com balanceamento de iterações.

4. ParMSM: Paralelização mestre-escravo do algoritmo seqüencial, utilizando a estratégia de Múltiplos percursos interativos, com balanceamento de iterações e

troca de melhor solução.

Algorithm	Custo S1	Custo S2	Tempo S1	Tempo S2	Speedup
Seq	5666.65	7108.73	1719	2866	—
ParInde	5688.17	7127.58	351	541	5.09
ParMSM	5675.13	7120.23	411	647	4.31
ParMS	5693.49	7136.48	455	687	3.98

Tabela 20: Algoritmos GRASP+VNSDropGeni paralelos.

Para o segundo teste os algoritmos utilizados foram:

1. Seq: Algoritmo GRASP+VNS seqüencial, usando para a solução inicial o procedimento Drop e para a busca local o procedimento Híbrido.

2. ParInd: Paralelização do algoritmo seqüencial, utilizando a estratégia de Múltiplos percursos independentes.

3. ParMS: Paralelização mestre-escravo do algoritmo seqüencial, utilizando a estratégia de Múltiplos percursos interativos, com balanceamento de iterações.

4. ParMSM: Paralelização mestre-escravo do algoritmo seqüencial, utilizando a estratégia de Múltiplos percursos interativos, com balanceamento de iterações e troca de informações.

Algorithm	Custo S1	Custo S2	Tempo S1	Tempo S2	Speedup
Sequen	5678.41	7112.99	1685	2857	—
ParInde	5696.37	7129.44	309	501	5.57
ParMSM	5681.84	7110.27	376	608	4.59
ParMS	5655.63	7123.89	372	610	4.61

Tabela 21: Algoritmos GRASP+VNSDrop.

A partir dos resultados parciais concluimos, que a paralelização dos algoritmos GRASP+VNS, também reduziu bastante o tempo computacional sem prejudicar a qualidade da solução. A partir dos resultados obtidos podemos concluir que a

paralelização das metaheurísticas, conseguiu reduzir bastante o tempo computacional sem prejudicar a qualidade da solução independente do algoritmo paralelizado.

A diferença de tempo entre os algoritmos paralelos se deve novamente, ao fato dos testes, serem feitos em uma rede homogênea com dedicação exclusiva e que as buscas locais não foram suficientes para gerar cargas diferenciadas nos processadores, a fim de justificar um balanceamento dinâmico de iterações, portanto, os algoritmos que faziam o balanceamento dinâmico, gastaram tempo com um balanceamento que era desnecessario.

7.3 Resumo dos Resultados dos Algoritmos Paralelos

Para o teste dos algoritmos paralelos aqui propostos, usamos as mesmas instâncias dos algoritmos GRASP paralelos, e executamos os algoritmos por 600 segundos.

Os Algoritmos paralelos testados foram:

1. ParGRASPAddAleatório: paralelo independente com algoritmo GRASP, SI Add-Aleatório e BL Híbrido.

2. ParGRASPAddGeni: paralelo independente com algoritmo GRASP, SI AddGeni e BL Híbrido.

3. ParGRASP+VNSDropGeni: paralelo independente com algoritmo GRASP+VNS, SI DropGeni e BL Híbrido.

4. ParGRASP+VNSDrop: paralelo independente com algoritmo GRASP+VNS, SI Drop e BL Híbrido.

5. ParGRASP: paralelo independente com algoritmos GRASP diferentes em processos diferentes.

6. ParVNS : paralelo independente com algoritmos GRASP+VNS diferentes em processos diferentes.

7. ParGRASPVNS : paralelo independente com algoritmos diferentes em processos diferentes, temos algoritmos GRASP em alguns processos e algoritmos VNS em outros processos.

Algorithm	Custo S1	Custo S2	Tempo S1	Tempo S2
ParGRASPAddAleatório	5694.37	7121.76	286.25	344.60
ParGRASPAddGeni	5736.82	7144.90	377.40	393.00
ParGRASP+VNSDropGeni	5651.16	7118.48	329.50	407.00
ParGRASP+VNSDrop	5713.21	7110.95	269.25	454.60
ParGRASP	5707.93	7115.06	355.50	320.40
ParGRASPVNS	5655.59	7114.87	392.75	283.50
ParVNS	5650.48	7107.41	292.25	372.00

Tabela 22: Algoritmos Paralelos independentes com critério de parada de 600 segundos

Pela Tabela 22, podemos perceber que os algorimos ParGRASP+VNSDropGeni e ParGRASP+VNSDrop, obtiveram melhores resultados que os algoritmos ParGRASPAddGeni e ParGRASPAddAleatório, este resultado era esperado uma vez que os algoritmos seqüencias correspondentes também apresentaram esse comportamento.

O algoritmo ParGRASP, obtém um resultado melhor que o ParGRASPAddGeni, porque o algoritmo ParGRASP, engloba o algoritmo ParGRASPAddGeni, em sua execução, ou seja, executa este algoritmo em um dos processadores. O mesmo ocorre com os algoritmos ParVNS, que engloba os algoritmos ParGRASP+VNSDropGeni e ParGRASP+VNSDrop.

Um resultado não esperado foi o algoritmo ParGRASP+VNS, que deveria ter os melhores resultados, uma vez, que ele engloba todos os demais algoritmos.

Nos algoritmos paralelos a execução com 6 processadores, temos a seguinte situações: o ParGRASP+VNS tem 3 processadores executando algoritmos GRASP e 3 executando algoritmos GRASP+VNS, enquanto que o algoritmo ParGRASP tem 6 processadores executando algoritmos GRASP e o algoritmos ParVNS tem 6 processadores executando algoritmos GRASP+VNS.

O resultado adverso do algoritmo ParGRASP+VNS pode ser contornado de duas formas a primeira é aumentando-se o tempo de execução para compensar os poucos algoritmos executando GRASP+VNS, uma vez que o algoritmo ParVNS obteve os melhores resultados e a segunda forma é aumentar o número de processadores para aumentar o número de processadores executando GRASP+VNS.

8 Conclusões

O objetivo deste trabalho, foi o de propor algoritmos seqüenciais e paralelos para a solução de generalizações do Problema do Caixeiro Viajante (PCV), onde apenas parte dos nós é utilizada numa solução viável. Dentre as aplicações possíveis, está um problema conhecido na literatura como The *Traveling Purchaser Problem* (TPP), que utilizamos como ilustração para avaliar os algoritmos aqui propostos.

O TPP, por ser uma generalização do PCV, está classificado como um problema NP-Difícil e com isso, a sua solução exata, fica limitada a instâncias de pequeno porte, mesmo levando-se em conta o grande desenvolvimento que tem se verificado em métodos exatos de otimização combinatória.

Este fato, tem levado pesquisadores a propor métodos aproximados ou heurísticos para a maioria das aplicações em otimização combinatória de elevada complexidade. Contudo sabemos que algoritmos heurísticos clássicos de construção e busca local, possuem alguns limitantes como por exemplo, a dificuldade em superar ótimos locais ainda distantes de um ótimo global.

Em parte, esta limitação foi superada com a proposta das chamadas heurísticas genéricas ou heurísticas inteligentes, que mais tarde recebeu o nome de metaheurísticas. As metaheurísticas além do seu carácter genérico, possuem mecanismos que procuram evitar as chamadas paradas prematuras que normalmente ocorrem

em heurísticas clássicas.

Atualmente na literatura, são várias as metaheurísticas utilizadas na área de otimização, desde *Redes Neurais Artificiais* (RNs) e *Simulated Annealing* que são as mais antigas e que juntamente com *Algoritmos Genéticos* (AGs), são as mais conhecidas em outras áreas. Na area de otimização, atualmente existem metaheurísticas bem conhecidas tais como: *Busca Tabu (Tabu Search)* (TS); *Scatter Search* (SS) que na verdade é uma versão menos probabilística dos AGs; GRASP e VNS. Dentre todas estas técnicas, as que têm se sobressaído na área de otimização, são: TS, GRASP e VNS.

O método TS já é mais explorado pela literatura afim, existem dezenas de trabalhos propondo algoritmos seqüenciais e um número considerável de propostas de algoritmos TS paralelos. Contudo na área de algoritmos paralelos, ainda pouco se tem visto sobre GRASP e VNS, devido possivelmente ao fato destes terem sido descobertos mais recentemente.

Estes fatos ligados ao bom desempenho dos métodos TS, GRASP e VNS na solução de diferentes problemas de otimização combinatória, nos incentivou a propor este trabalho.

Para isso, selecionamos uma generalização do clássico PCV, conhecido na literatura como *Traveling Purchaser Problem* (TPP), que apesar das inúmeras aplicações práticas, ainda é pouco explorado pela literatura.

Além disso, a escolha do TPP para avaliar as nossas propostas, foi influenciado pela existência de algoritmos do tipo TS com as estruturas de listas Tabu dinâmicas propostas por Voss [48, 49] para a solução deste problema.

Inicialmente partimos em busca de algoritmos de construção e de busca

local que pudessem ser incluídos em métodos do tipo GRASP, VNS e TS. Para isso utilizamos desde técnicas já bastante conhecidas como ADD, DROP e SWAP adaptando-as para o TPP, como variantes destas, formas híbridas, até o uso de metaheurísticas do tipo VNS e suas variantes para efetuar busca local num algoritmo GRASP ou TS.

Com esse conjunto de alternativas para construção e refinamento de uma solução viável, partimos para a construção de diferentes versões do algoritmo GRASP, VNS e TS, onde no TS sempre utilizamos a estrutura de listas dinâmicas propostas por Voss [49].

Dos resultados computacionais obtidos, podemos concluir que os algoritmos do tipo GRASP e do tipo VNS conseguiram na média resultados semelhantes a melhor versão existente atualmente de TS na literatura. Além disso, resultados mais promissores foram obtidos por versões híbridas conjugando conceitos de GRASP e de VNS num único algoritmo. Estes resultados parciais nos mostram o enorme potencial destas técnicas na solução de problemas de elevada complexidade computacional como é o caso do TPP.

Baseado nos resultados dos algoritmos seqüenciais aqui propostos, selecionamos aqueles de melhor desempenho, para a construção de versões paralelas do GRASP e VNS. Propomos o uso de diferentes estratégias , incluindo modelos: independentes ou com comunicação; supervisionado ou não supervisionado; síncrono ou assíncrono; com e sem balanceamento de cargas.

Diferentes algoritmos paralelos para as metaheurísticas GRASP, VNS e versões híbridas, foram implementados e submetidas a testes computacionais. Devido à limitação de tempo para concluir o curso de Mestrado, reconhecemos que a bateria de testes foi pequena, mas nos resultados obtidos pudemos ter as seguintes indicações:

como era de se esperar, algoritmos paralelos tem como função básica a redução dos tempos computacionais exigidos pelo seqüencial. Isso ocorreu com a manutenção da qualidade das soluções obtidas pelas melhores versões do seqüencial. Alguns resultados obtidos nos mostram a necessidade de efetuarmos testes adicionais, para avaliarmos melhor por exemplo, o problema de balanceamento de carga. Os resultados mostram que o balanceamento de carga estático apresenta os melhores resultados quando comparado com os algoritmos de balanceamento de carga dinâmico considerando-se o tempo de execução. Isto pode ser explicado pelo fato de que nossos testes foram executados em um computador IBM SP/2 com seis processadores idênticos, dedicados exclusivamente à nossa aplicação. Assim, a aplicação não pode tirar vantagem do balanceamento de carga dinâmico, o qual poderia melhorar o tempo de execução da aplicação se os processos executassem em velocidades diferentes. No modelo completamente distribuído, como inicialmente as iterações são divididas igualmente entre os processos, este se comporta como o algoritmo que usa balanceamento de carga estático. Sendo que a diferença está no fato de que ele perde tempo para terminação, aumentando um pouco o tempo de execução. Assim, testes computacionais deverão prosseguir em sistemas heterogêneos, sem dedicação exclusiva, ou com simulação de carga externa, para avaliar melhor a estratégia de balanceamento de carga dinâmico.

De uma forma geral, acreditamos que os resultados obtidos pelos algoritmos GRASP, VNS e GRASP+VNS justificam a nossa proposta, já que conseguimos resultados médios semelhantes ou em alguns casos até superiores que uma das melhores versões de TS existentes na literatura.

8.1 Propostas Futuras

Como propostas futuras, podemos incluir: Um estudo mais profundo

com uma variedade maior de testes computacionais para as versões paralelas aqui propostas. Uma avaliação da formulação matemática aqui proposta, onde utilizamos variaveis de fluxo para definir restrições que evitam a formação de sub rotas desconexas da origem. Esta avaliação poderá ser feita utilizando por exemplo, softwares do tipo XPRESS ou similares. Adaptação dos algoritmos aqui propostos para outros problemas de otimização combinatória. Esta tarefa acreditamos não ser complexa, devido à forma que trabalham os algoritmos GRASP e VNS que exigem basicamente algoritmos de construção e busca local.

Bibliografia

[1] Alvim, A.C., Estratégias de paralelização da Metaheurìstica GRASP,Dissertação de Mestrado,*Departamento de Informática PUC-RJ* (1998).

[2] Alvim, A.C. e Ribeiro C.C., Balanceamento de Carga na Paralelização da Metaheurística GRASP,*Departamento de Informática PUC-RJ* (1999).

[3] Battiti, R., Tecchiolli, G. Parallel biased search for combinatorial optimization: genetic algorithms and tabu.*Microprocessors and microsystems 16*, (1992), pp. 351–367.

[4] Chakrapani, J., Skorin-Kapov, J. A connectionist approach to the quadratic assignment problem. *Computer and Operations Reasearch 19* (1992), pp. 287–295.

[5] Chakrapani, J., Skorin-Kapov, J. Massively parallel tabu search for the quadratic assignment problem. *Annals of Operations Reasearch 41*, 1993, pp. 327–341.

[6] Colin,*Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications (1993).

[7] Crainic, T. G., Toulouse, M., Gendreau, M., Toward a taxonomy of parallel tabu search heuristics.*INFORMS Journal on Computing* (1996).

[8] Crainic, T. G., Toulouse, M., Gendreau, M., Toward a taxonomy of parallel tabu search Algorithms.*Research Report CRT-933, Centre de Recherche sur les Transports*, Université de Montréal (1993).

[9] Crainic, T. G., Toulouse, M., Gendreau, M., Parallel asynchronous tabu search for muticommodity location-allocation with balancing requirements. *Annals of Operations Research 63* (1996), pp. 277–299.

[10] Dijkstra, E. W., Scholten, C. S., Termination Detection for Diffusing Computations.*Information Processing Letters* (1980), 11, pp. ,1–4

[11] Eshelman, L. The chc adaptive search algorithm: How to have a safe search when engaging in nontraditional genetic recombination. Foundation of genetic algorithms and classifier systems. Gregory J. Hawlins Ed. Morgan-kauffman, 1991.

[12] Fiechter, C. N. A parallel tabu search algorithm for large travelling salesman problems. *Discrete Applied Mathematics 51* (1994), pp. 243–267.

[13] Gendreau, M.,Hertz, A. e Laporte, G., New Insertion and Postoptimization Procedures for The Traveling Slesman Problem,*Operations Research 40/6* (1986), pp. 1086-1094.

[14] George-Schleuter, M. Asparagos: An asynchronous parallel genetic optimization strategy.*Proc. of the Third International Conference on Genetic Algorithms*, 1989, pp. 422–427.

[15] Glover, F., Future Paths for Integer Programing and Links to Artificial Intelligence Problem,*Management Science 40/10* (1986), pp. 1276–1290.

[16] Glover, F., Tabu Search Part I,*ORSA Journal on Computing 1* (1989), pp. 190–206.

[17] Glover, F., Tabu Search Part II,*ORSA Journal on Computing 2* (1990), pp. 4–32.

[18] Golden, B. L., Levy, L. e Dahl, R.,Two generalizations of the Traveling Purchaser Problem,*OMEGA* (1981),9(4),439–445.

[19] Golden, B. L., Levy, L. e Gheysens, F. G.,The fleet size and mix vehicle routing problem.,*Comp. and Oper. Res* (1984),49–66.

[20] Hansen, P. The steepest ascent mildest descent heuristic for combinatorial programming. *Congress on Numerical Methods on Combinatorial Otimizations*, Capri, Italy, 1986.

[21] Hansen, P. e Mladenovič, N., Variable Neighborhood Search for the p-median,*Location Science* (1997), 5,207–226.

[22] Hansen, P.,Mladenovič, N. e Perez-Brito, D., Variable Neighborhood Decomposition Search,*GERAD* (1998).

[23] Huang, S. T., Termination Detection by Using Distributed Snapshots. *Information Processing Letters*(1989), 32, pp. 113–119

[24] Jog, P., Suh, J.Y. e Van Gucht, D., Parallel Genetic Algorithms Applied to the Traveling Salesman Problem, *SIAM Journal of Optimization 1*, (1991), pp. 515–529

[25] Malek, M. Guruswamy, M. Pandya, M, Owens, H. Serial and parallel simulated annealing and tabu search algorithms for the travelling salesman problem. *Annals of Operations Reasearch 21* (1989), pp. 59–84.

[26] Manderick, B., Spiessens, P. A massively parallel genetic algorithm: Implementation and first results.*Proc. of the Fourth International Conference on Genetic Algorithms*, 1991, pp. 279–286.

[27] Martins, L. S., Paralelização de Metaheurìsticas em ambientes de Memória Distribuída. *Tese de Doutorado*(1999), PUC, Rio de Janeiro, Departamento de informática.

[28] Martins, S. L., Resende, M.G.C. e Ribeiro, Celso C., A parallel GRASP for the Steiner problem in graphs using a hybrid local search. *MIC'99 - Third Metaheuristic International Conference*, 1999.

[29] Mladenović, N., A veriable neighborhood algorithm - a new metaheuristic for combinatorial optimization. *Abstract of papers present at Optimization Days*,1995, Montreal, pp 12.

[30] Silva, M. B., Drummond, L. M. A., Ochi, L.S., Variable Neighborhood Search for The Traveling Purchaser Problem,*Proc. of the 27th Int. Conf. on Computer e Industrial Engineering Beijing*, China, 2000.

[31] Ochi, L. S., Drummond, L. M. A. e Figueiredo, R. M. V., Um Novo Algoritmo Genético Paralelo Distribuí para o Travelling Purchaser Problem.

[32] Ong, H. L.,Approximate Algorithms for the Traveling Purchaser Problem,*Operations Research Letters* (1982), 1(5), pp. 201–205.

[33] Li, Y.,Pardalos, P. M. e Resende, M.G.C., A Greedy Randomized Adaptative Search Procedures for the quadratic assigment problem, em Quadratic assignment

and related problems (P.Pardalos e H. Wolkowicz, editores),*DIMACS Series on Discrete Mathematics and Theoretical Computer Science 16* (1994),pp. 237–261.

[34] Pearl, J. Heuristics: intelligent search strategies for computer problem solving. Addison Wesley, R. Hass, 1984.

[35] Pearn, W. L., On The Traveling Purchaser Problem,*Department of Industrial Engeneering and Management, National Chiao Tung University* (1991).

[36] Pearn, W. L., Chien, R. C., Improved Solutions for the Traveling Purchaser Problem,*Computers Operation Research* (1998), 25(11), pp. 879–885.

[37] Pettey, C. B., Lenze, M. R., and Grefenstatte, J. J. A parallel genetic algorithm. *Proc. of the Third International Conference on Genetic Algorithms*, 1989, pp. 398–405.

[38] Porto, S.C.S. e Ribeiro, C.C., A Tabu Search Approach to Task Scheduling on Heterogeneous Processors under Precedence Constrains *International Journal of High Speed Computing 7*, 1995, pp. 45–71.

[39] Porto, S.C.S. e Ribeiro, C.C., Parallel Tabu Search Message-Passing Synchronous Strategies for Task Scheduling under Precedence Constrains *Journal of Heuristics 1*, 1996.

[40] Ramesh, T.,Traveling Purchaser Problem,*OPSEARCH* (1981), 18(2), pp. 78–91.

[41] Resende, M.G.C. e Feo, T.A., A Greedy Randomized Adaptative Search Procedures for maximum independent set,*Operations Research 42* (1994),pp. 860–878.

[42] Resende, M.G.C. e Feo, T.A., Greedy Randomized Adaptative Search Procedures,*Journal of Global Optimization 6* (1995), pp. 109–133.

[43] Taillard, E. Parallel iterative search methods for vehicle routing routing problems. *NETWORKS 23* (1993), pp. 661–673.

[44] Taillard, E. Robust taboo search for the quadratic assigment problem. *Parallel Computing 17* (1991), pp. 443–455.

[45] Taillard, E. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing 6* (1994), pp. 108–117.

[46] Tanese, R. Distributed genetic algorithms.*Proc. of the Third International Conference on Genetic Algorithms*, 1989, pp. 434–440.

[47] Verhoeven, M.G.A., Aarts, E.H.L., Parallel Local Search.*Journal of Heuristics 1*, 1995, pp. 43–65.

[48] S. Voss (1996), ADD and DROP-procedures for the traveling purchaser problem. Methods of operations research, 53, pp:317-318.

[49] S. Voss (1996), Dynamic tabu search strategies for the traveling purchaser problem. Annals of Operations Research, 63, pp:253-275.

[50] Whitley, D. The genetic algorithm and selective pressure: Why rank-based allcation of reproductive trials is best. *Proc. of the Third International Conference on Genetic Algorithms*, 1989, pp. 116–121.