# Abordagem *Variable Neighborhood Search* para o Problema de Seqüenciamento com Máquinas Paralelas e Tempos de Preparação Dependentes da Seqüência

**Mateus Rocha de Paula**[1], **Martín Gómez Ravetti**[1], **Panos M. Pardalos**[2]

[1] Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

[2]Dept. of Industrial and Systems Engineering (ISE), University of Florida (ISE-UF)

{mahdi,martin}@dcc.ufmg.br , pardalos@ufl.edu

***Abstract.*** *Variable Neighborhood Search (VNS) is a modern metaheuristic based on systematic changes of the neighborhood of the solutions to solve optimization problems. The aim of this paper is to propose and analyze a VNS algorithm to solve scheduling problems with parallel machines and sequence dependent setup times, which is of great importance on the industrial context. Through extensive experiments, for instances with 60 jobs or more, the VNS algorithm has shown a better performance when compared with three algorithms based on the GRASP (Greedy Randomized Adaptive Search Procedure) metaheuristic.*
***Key-words: Scheduling, VNS, Parallel Machines***

***Resumo.*** *A Variable Neighborhood Search (VNS) é uma metaheurística moderna que se baseia em mudanças sistemáticas da vizinhança das soluções para resolver problemas de otimização combinatória. O objetivo deste trabalho é propor e analisar um algoritmo VNS para resolver problemas de seqüenciamento considerando máquinas paralelas e tempos de preparação de máquinas dependentes da seqüência. Através da análise de vários experimentos, para instâncias com 60 ou mais tarefas, o algoritmo VNS apresentou um melhor desempenho quando comparado com três algoritmos baseados na metaheurística GRASP (Greedy Randomized Adaptive Search Procedure).*
***Palavras-chave: Problemas de Seqüenciamento, VNS, Máquinas Paralelas Área de classificação principal: Metaheurísticas***

## 1. Introdução

Neste trabalho consideramos o problema de seqüenciamento com máquinas paralelas e tempos de preparação dependentes da seqüência de tarefas e da máquina, minimizando a soma do tempo de conclusão (*makespan*) e dos atrasos ponderados. Para casos com máquinas paralelas, é possível provar que o problema, considerando duas máquinas idênticas e minimizando o *makespan*, é *NP-Difícil* (Garey and Johnson; 1997). Dessa forma, o caso mais complexo de minimização do *makespan* com máquinas paralelas idênticas ou não-relacionadas e tempos de preparação dependentes da seqüência é também *NP-Difícil*.

Diversos trabalhos na literatura abordam os problemas de seqüenciamento com máquinas paralelas mas somente alguns os tratam com tempos de preparação dependentes da máquina e da seqüência, máquinas não relacionadas e datas de entrega. Elmagraby and Park (1974), Barnes and Brennan (1977) e Vasilescu and Amar (1983) tratam o caso com máquinas idênticas e sem tempos de preparação de máquinas, utilizando um esquema de *Branch-and-Bound*. Dogramaci (1984) o soluciona utilizando programação dinâmica. Arkin and Roundy (1991) se envolvem com o caso onde os pesos considerados são proporcionais aos tempos de processamento com uma heurística denominada *Earliest Gamma Date*. Chen and Powell (1999) fazem uso de geração de colunas juntamente com um algoritmo *Branch-and-Bound* para seqüenciar tarefas com uma mesma data de entrega em máquinas idênticas ou não-relacionadas.

O VNS é uma metaheuristica moderna que se baseia em mudanças sistemáticas de vizinhança das soluções para resolver problemas de otimização. Um estudo bastante completo sobre ele pode ser encontrado em (Hansen and Mladenovic; 1999), (Hansen and Mladenovic; 2003) e (Hansen et al.; 2002). Essa metaheuristica foi aplicada com sucesso para resolver diversos problemas como o p-mediano (García-López et al.; 2002), o problema de roteamento com várias estações (Polacek et al.; 2004) e muitos outros problemas clássicos (Hansen and Mladenovic; 1999), (Hansen and Mladenovic; 2003) e (Hansen and Mladenovic; 2002).

Definiçoes, modelos e abordagens clássicas podem ser encontrados em Pinedo (1995), Lee and Pinedo (2002), Błażewicz et al. (1996) e Brucker (2004).

Na implementação proposta, uma boa solução inicial é obtida através de um algoritmo baseado no proposto por Nawaz et al. (1983) (NEH), conhecido por ser uma das melhores heurísticas polinomiais para o problema de *flowshop* permutacional.

Na Seção 2 deste artigo apresentamos o problema de seqüenciamento com máquinas paralelas. Na Seção 3 detalhamos o algoritmo VNS. Na Seção 4 apresentamos os algoritmos baseados na metaheuristica GRASP utilizados para a análise de desempenho do VNS; na Seção 5 apresentamos os resultados computacionais e algumas conclusões parciais. Finalmente, na Seção 6 concluimos o trabalho e são apresentadas propostas de pesquisas futuras.

## 2. Problemas de seqüenciamento com máquinas paralelas

Seja um conjunto de máquinas $\mathcal{I} = \{1, 2, .., m\}$ e um conjunto de tarefas $\mathcal{J} = \{1, 2, .., n\}$ com tempos de processamento positivos $p_{ji}$ e penalidades $w_j$ positivas na entrega de uma tarefa, para cada tarefa $j \in \mathcal{J}$ e cada máquina $i \in \mathcal{I}$. O problema consiste em seqüenciar todas as tarefas, de forma a minimizar uma determinada função objetivo, que depende dos objetivos almejados.

Para processar a tarefa $j$ depois da tarefa $j'$ é necessário um tempo de preparação $s_{j'ji}$ que depende da seqüência de tarefas $j' \prec j$ e da máquina $i \in \mathcal{I}$ onde estão sendo processadas.

A data de término da tarefa $j$ pode ser recursivamente definida como $C_j = C_{j'} + p_{ji} + s_{j'ji}$, onde $C_{j'}$ é zero se a tarefa $j$ é a primeira da seqüência ou a data de término da tarefa $j'$ caso contrário. O atraso $t_j$ de cada tarefa é calculado como $t_j = max((C_j - d_j), 0)$, onde $d_j$ é a data

planejada de entrega da tarefa $j$.

A função objetivo explorada neste artigo é a minimização da soma do *makespan* e os atrasos ponderados, $C_{max} + \sum_{j=1}^{n} w_j.t_j$. Este problema é também conhecido como $R|s_{jj'i}, \tilde{d}_j|C_{max}, \sum w_j T_j$ [1].

## 3. O algoritmo VNS

A arquitetura do VNS e notação aqui utilizadas seguem os modelos propostos por Hansen and Mladenovic (2003).

Definindo uma seqüência de tarefas $I_i$ para a máquina $i$ como a permutação dos elementos do subconjunto $\mathcal{I}_i \subseteq 2^{\mathcal{J}}$, onde $2^{\mathcal{J}}$ é o conjunto de todos os subconjuntos de $\mathcal{J}$, e:

$$\mathcal{I}_i \cap \mathcal{I}_{i'} = \emptyset, \qquad \forall i' \in \mathcal{I}, i \neq i'$$

$$\bigcup_{i \in \mathcal{I}} \mathcal{I}_i = \mathcal{J}$$

Seja $\mathcal{F}$ o conjunto de todas as soluções viáveis. Considerando ainda uma solução $\mathcal{S} = \{I_1, I_2, .., I_m\} \in \mathcal{F}$. Uma *Estrutura de Vizinhança* associa a cada $\mathcal{S} \in \mathcal{F}$ uma vizinhança $\mathcal{N}_k(\mathcal{S}) \subseteq \mathcal{F}$ da solução $\mathcal{S}$.

Neste artigo, definimos três vizinhanças:

1. Trocas de tarefas na mesma máquina. Uma máquina é escolhida e são consideradas todas as possíveis trocas entre as tarefas nela seqüenciadas.
2. Trocas de tarefas entre duas máquinas diferentes. Duas máquinas são escolhidas e são consideradas todas as possíveis trocas de tarefas entre elas.
3. Transferências de tarefas de uma máquina para outra. Duas máquinas são escolhidas e são consideradas todas as possíveis transferências da primeira para a segunda. desta máquina para qualquer outra.

É importante notar que as vizinhanças em si ($\mathcal{N}_1(\mathcal{S})$, $\mathcal{N}_2(\mathcal{S})$ e $\mathcal{N}_3(\mathcal{S})$, respectivamente) são determinadas tanto por sua respectiva estrutura quanto pela solução sobre a qual estão sendo aplicadas.

O tamanho da vizinhança $\mathcal{N}_1(\mathcal{S})$ é da ordem de $O(m.n^2)$, o da vizinhança $\mathcal{N}_2(\mathcal{S})$ é $O(m^2.n^2)$ e o da vizinhança $\mathcal{N}_3(\mathcal{S})$ é $O(m^2.n^2)$.

### 3.1. A solução inicial

Qualquer método capaz de gerar uma solução viável poderia ser utilizado nesta primeira parte do algoritmo VNS, mas Johnson et al. (1989) e Matsuo et al. (1989) mostraram que uma boa solução inicial pode reduzir consideravelmente o tempo de computação.

Assim sendo, neste trabalho é proposta a utilização de um algoritmo baseado no NEH (Nawaz et al.; 1983), que tem se mostrado uma das abordagens polinomiais mais interessantes para o problema de seqüenciamento *flowshop*.

Uma vez que o alvo desta implementação são instâncias realistas, isto é, com um grande número de tarefas, o custo de computação acrescentado por este algoritmo é pouco relevante se comparado com o custo imposto pelas buscas locais. Assim sendo, é vantajoso utilizar o NEH, vista a boa qualidade da solução inicial proposta por ele. A implementação aqui proposta tem ordem de complexidade $O(n^3.m)$.

[1] A notação utilizada em problemas de seqüenciamento pode variar. Em nosso caso, seguimos Brucker (2004)

**Algoritmo 1** Esquema básico do VNS

```
1: Encontra uma solução inicial S*
2: l ← 1
3: for Iterações ← 1 até um número máximo de iterações do
4:    S ← S*
5:    Shake procedure: Encontra uma solução aleatória S' ∈ N_l(S)
6:    Faz uma busca local em N_l(S') para encontrar uma solução S"
7:    if S" < S* then
8:       S* ← S"
9:       l ← 1
10:   else
11:      l ← l+1
12:   end if
13: end for
```

**Algoritmo 2** Construção da solução inicial

```
1: Ordena as tarefas em ordem de datas de entrega
2: for Cada tarefa i do
3:    MKS ← INT-MAX;
4:    for Cada máquina m do
5:       for Cada posição p em cada máquina m do
6:          MKS'= INSERE tarefa i em m na posição p
7:          if MKS' < MKS then
8:             MKS ← MKS'
9:          else
10:            Retira i de m na posição p.
11:         end if
12:      end for
13:   end for
14: end for
```

### 3.2. Soluções aleatórias

Nosso algoritmo associa um procedimento para a geração de uma solução aleatória e uma busca local a cada estrutura de vizinhança. Os procedimentos de geração de solução aleatória são aplicados à melhor solução encontrada até o momento no início de cada iteração. Como são três as estruturas de vizinhança definidas neste artigo, três procedimentos de geração de soluções aleatórias são definidos:

1. Para $\mathcal{N}_1(S)$:
   - É escolhida uma máquina $i$ aleatoriamente.
   - São escolhidas duas tarefas $j_1$ e $j_2$ na máquina $i$ aleatoriamente.
   - Troca-se a tarefa $j_1$ pela tarefa $j_2$ e vice-versa.
2. Para $\mathcal{N}_2(S)$:
   - São escolhidas 2 máquinas $i_1, i_2$ aleatoriamente.
   - São escolhidas uma tarefa $j_1$ em $i_1$ uma tarefa $j_2$ em $i_2$ aleatoriamente.
   - Troca-se a tarefa $j_1$ pela tarefa $j_2$ e vice-versa.
3. Para $\mathcal{N}_3(S)$:
   - São escolhidas aleatoriamente uma tarefa $j_1$ e uma máquina $i_2$, onde $j_1$ não pertença a $i_2$.
   - É escolhida aleatoriamente uma posição válida *pos* em $i_2$.
   - Transfere-se a tarefa $j_1$ para $i_2$ na posição *pos*.

### 3.3. Buscas locais

Da mesma forma que no caso anterior, para cada vizinhança definimos uma busca local específica, de forma a melhor aproveitar sua estrutura. Os algoritmos 3, 4 e 5, apresentam o pseudo-código das buscas locais.

**Algoritmo 3** [Busca Local 1.] Trocas de tarefas na mesma máquina. Para cada máquina disponível são analisadas todas as possíveis trocas entre as tarefas nela seqüenciadas. Mesmo quando a máquina escolhida não é a de maior *makespan*, é possível melhorar a função objetivo através da diminuição dos atrasos. O algoritmo tem ordem de complexidade $O(m.n^2)$.

```
1: for cada i do
2:   for cada j1 em i do
3:     for cada j2 em i, j1 ≠ j2 do
4:       if Solução considerando j1 e j2 trocados < Solução atual then
5:         Troca j1 e j2
6:       end if
7:     end for
8:   end for
9: end for
```

Pela análise das buscas locais, podemos concluir que a Busca Local 1 (BL1) restringe bastante o espaço de busca, sendo desta forma a mais rápida das três.

Desta forma, o nosso algoritmo VNS, utiliza as buscas locais em forma sequencial, BL1, BL2 e BL3. Se após uma iteração o valor da função objetivo não é melhorado, a próxima vizinhança, é utilizada ($l$ é incrementado). Se, durante a busca local, uma solução melhor é encontrada, a primeira vizinhança (BL1) volta a ser utilizada na próxima iteração ($l$=1).

Para instâncias com datas de entrega mais folgadas, BL1 é eficaz somente quando a máquina escolhida é a que ocasiona o *makespan*. Para instâncias com datas de entrega mais apertadas, a mesma também pode melhorar a função objetivo reduzindo os atrasos e, conseqüentemente

**Algoritmo 4** [Busca Local 2.] Trocas de tarefas entre duas máquinas diferentes. Para cada possível par de máquinas disponíveis são analisadas todas as possíveis trocas de tarefas entre elas. São pesquisadas mais soluções que pela Busca Local 1. O algoritmo tem ordem de complexidade $O(m^2.n^2)$.

```
1: for cada i_1 ∈ I do
2:    for cada j_1 ∈ i_1 do
3:       for cada i_2 ∈ I, i_1 ≠ i_2 do
4:          for cada j_2 ∈ i_2 do
5:             if A solução, considerando j_1 e j_2 trocados < Solução atual then
6:                Troca j_1 e j_2
7:             end if
8:          end for
9:       end for
10:   end for
11: end for
```

**Algoritmo 5** [Busca Local 3.] Inserção de tarefas. São analisadas todas as possíveis transferências de tarefas da máquina de maior para a de menor *makespan*. O algoritmo tem ordem de complexidade $O(n^2)$.

```
1: Encontra a máquina com o maior makespan i_1.
2: Encontra a máquina com o menor makespan i_2, i_1 ≠ i_2.
3: for cada j ∈ i_1 do
4:    for cada posição válida pos em i_2 do
5:       if Solução considerando j transferido de i_1 para i_2 na posição pos < Solução atual then
6:          Transfere j de i_1 para i_2 na posição pos.
7:       end if
8:    end for
9: end for
```

o tempo computacional. O espaço de busca utilizado pelas BL2 é bem mais amplo que o das outras duas buscas. Já a BL3 propõe alterações mais radicais na solução tratada, podendo causar um impacto maior no valor da função objetivo.

Instâncias mal comportadas, isto é, que possuem simultâneamente intervalos com datas de entrega apertadas e folgadas, são também tratadas de forma conveniente utilizando todas as buscas locais disponíveis. Maiores informações sobre técnicas de busca locais em vizinhanças podem ser encontradas em Ahuja et al. (2002).

## 4. O algoritmo GRASP

A metaheuristica GRASP (*Greedy Randomized Adaptive Search Procedure*) utilizada neste trabalho tem uma estrutura similar à do GRASP básico Feo and Resende (1995). Ela tem se mostrado muito eficiente para diversos tipos de problemas combinatórios (Feo and Resende; 1995) e (Feo et al.; 1996). Festa and Resende (2002) apresentam uma bibliografia extensa sobre a metaheuristica.

Basicamente, esta metaheuristica é composta de duas fases que são repetidas a cada iteração: a construção de uma solução viável e uma busca local. Também utilizamos a técnica de *Path Relinking* para intensificar a busca local. Resende and Ribeiro (2003) apresentam uma análise detalhada de possíveis aplicações do *Path Relinking* no algoritmo GRASP.

Esta metaheruística foi proposta para este problema por Gomez Ravetti et al. (2006) e utilizada como limite superior para uma abordagem do tipo Branch & Bound por Rocha et al. (2004).

**Algoritmo 6** Esquema do GRASP

1: Encontra uma solução inicial S* ordenando as tarefas
2: **for** Iterações ← 1 até um número máximo de iterações **do**
3: *Shake procedure*: Cria uma solução aleatória S aplicando uma função de probabilidade em S*
4: Faz uma buscas local em S para encontrar uma solução S'
5: Realiza o *Path-Relinking* para melhorar S', obtendo S"
6: **if** S" < S* **then**
7: S* ← S"
8: **end if**
9: **end for**

O esquema acima é utilizado para criar as três implementações GRASP citadas nos experimentos. Cada versão utiliza uma das buscas locais descritas na Seção 3.3.

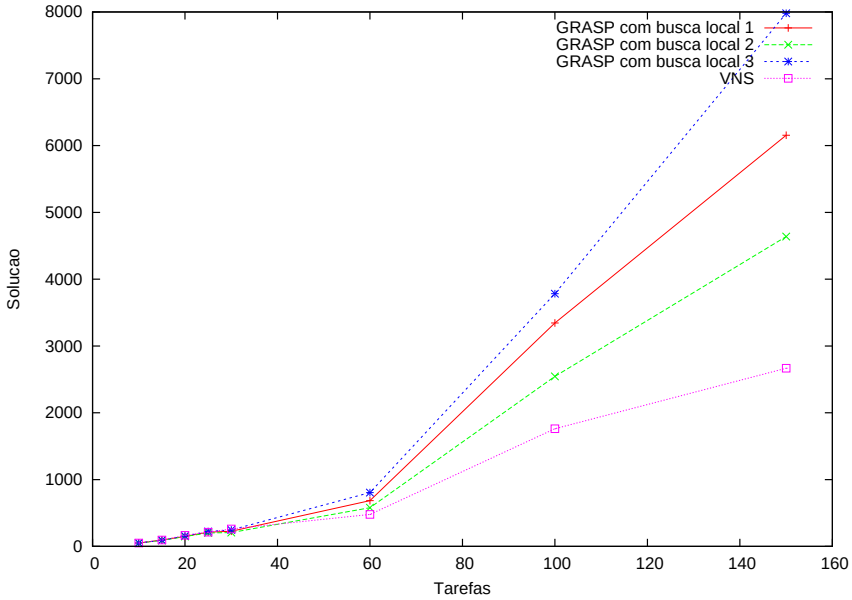
## 5. Resultados Computacionais

Nesta seção são apresentados os resultados computacionais de diversos experimentos considerando problemas de seqüenciamento de diversas categorias de data de entrega. Todos os algoritmos foram implementados em C, utilizando a versão 4.0.3 do GCC. Os experimentos foram executados num Pentium 4 com clock de 3.0 GHz e 1 MB de memória RAM, em ambiente Debian GNU/Linux.

As instâncias utilizadas foram geradas aleatoriamente, detalhes sobre a geração podem ser encontrados em (Rocha et al.; 2006).

Neste experimento utilizam-se, 8 conjuntos, com 10, 15, 20, 25, 30, 60, 100 and 150 tarefas são utilizados. Cada conjunto é composto por 30 instâncias com datas de entrega progressivamente mais apertadas. Cada instância é resolvida utilizando dez sementes diferentes, e a solução e tempo

**Figura 1. Resultados obtidos com instâncias com datas de entrega apertadas. O VNS é a curva marcada com quadrados.**




de computação médios são considerados. O número de iterações executadas por cada algoritmo é constante e igual a 10.000.

As figuras 1, 2 e 3 comparam os resultados computacionais das três versões do GRASP e do VNS, considerando três tipos de instâncias: com datas de entrega apertadas, folgadas e com os dois tipos. Em todas as três categorias, o algoritmo VNS provê soluções médias melhores ao considerar instâncias com 60 ou mais tarefas.

**Figura 2. Resultados obtidos com instâncias com datas de entrega folgadas. O VNS é a curva marcada com quadrados.**

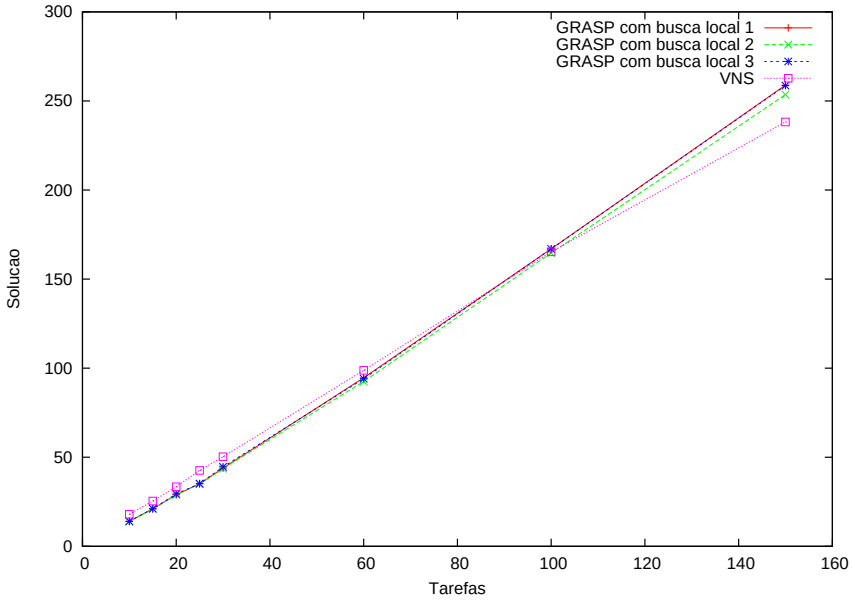
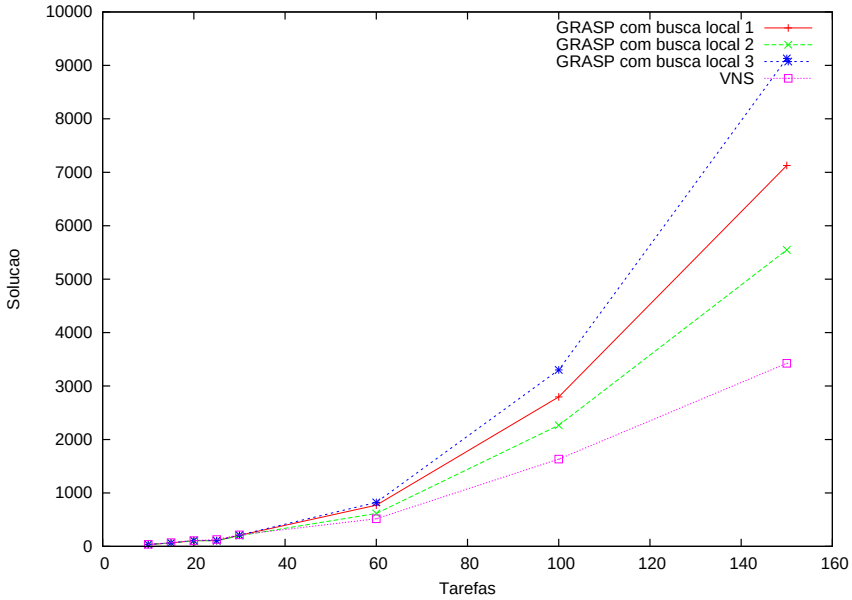
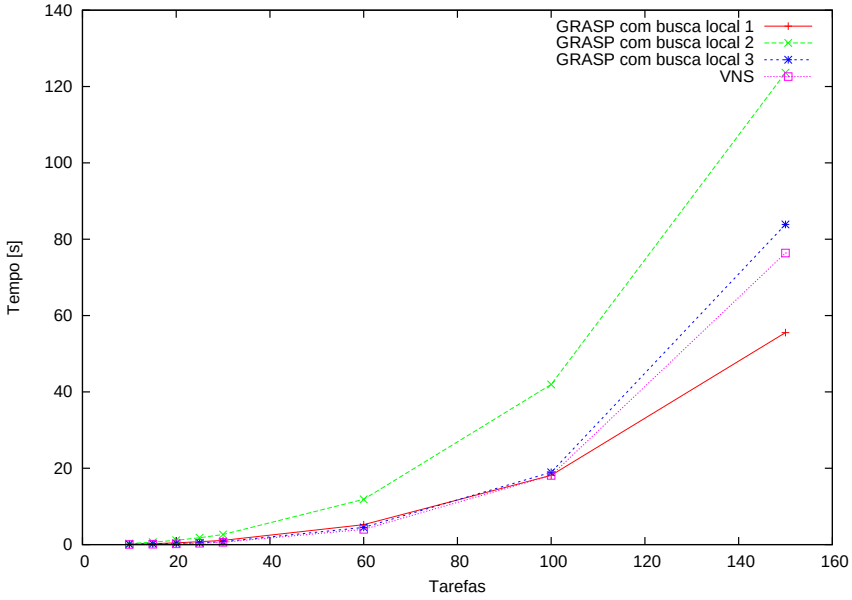


**Figura 3. Resultados obtidos com instâncias com datas de entrega apertadas ou folgadas. O VNS é a curva marcada com quadrados.**



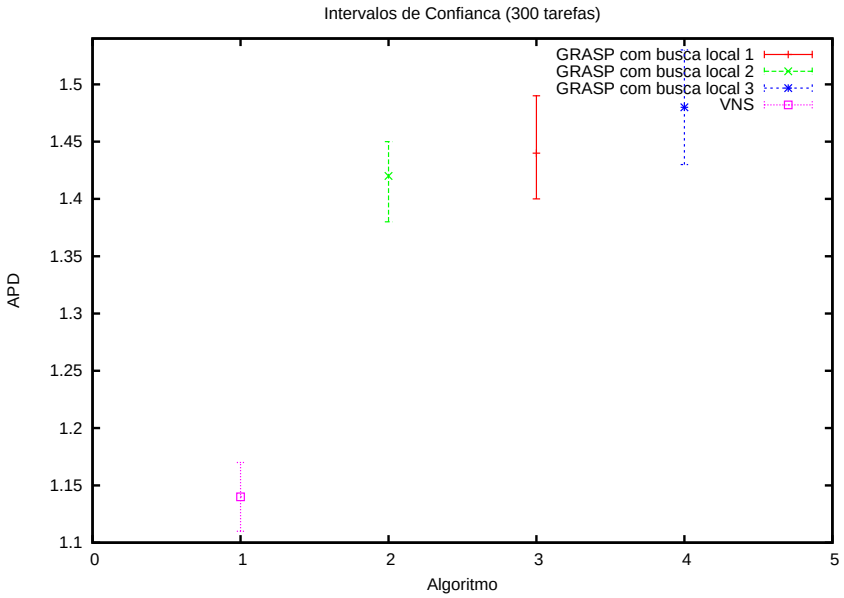

Para instâncias com datas de entrega apertadas (Figura 1) o algoritmo VNS apresenta soluções de melhor qualidade e, analisando as curvas do GRASP, é possível perceber que a BL2 é a mais adequada. Para instâncias com datas de entrega folgadas (Figura 2), o algoritmo VNS apresenta soluções melhores quando considerando mais de 100 tarefas, e o desempenho geral dos algoritmos é similar. Para o caso com intervalos de datas de entrega apertadas e folgadas (Figura 3), o VNS apresenta um desempenho melhor para 60 tarefas ou mais.

**Figura 4. Tempo de computação [em segundos] para instâncias com datas de entrega apertadas ou folgadas. O VNS é a curva marcada com quadrados.**



Considerando o tempo de computação, a combinação das buscas locais permite que o VNS obtenha bons resultados rapidamente. A Figura 4 mostra que o VNS obtém resultados mais rápido que duas das versões do GRASP, principalmente quando se tratando de instâncias de grande porte.

**Figura 5. Intervalos de Confiança do APD para instâncias com 150 tarefas. GRASP1, GRASP2 e GRASP3 representam os algoritmos GRASP utilizando as buscas locais 1, 2 e 3 respectivamente (see subsection 3.3).**




A Figura 5 apresenta os intervalos de confiança de APD para o grupo de instâncias de 300 tarefas. Desta forma, considerando instâncias realistas, com um numero de tarefas entre 150 e 300, podemos concluir que, em média o algoritmo VNS encontra melhores soluções em 95% dos casos. Existe ainda uma forte tendência de que, com o aumento do número de tarefas, maior a probabilidade de que o VNS obtenha soluções de qualidade melhor do que as propostas pelo GRASP.

## 6. Conclusões

Neste artigo tratamos o problema de seqüenciamento com máquinas paralelas e tempos de preparação dependentes da seqüência com um grande número de tarefas, minimizando o *makespan* somado aos atrasos ponderados.

A principal contribuição do trabalho é o desenvolvimento de um algoritmo baseado no VNS para resolver instâncias de grande porte. Para testar o seu desempenho, foi realizada uma comparação com a metaheuristica GRASP de eficiência demonstrada.

Analisando os resultados computacionais, é possível observar que o VNS obtém soluções médias muito boas quando considerando 60 ou mais tarefas, especialmente quando as datas de entrega são apertadas. A combinação das três buscas locais é bastante eficiente com uma implementação simples e sem necessidade de ajuste de muitos parâmetros.

Como trabalhos futuros podemos citar um estudo mais detalhado sobre técnicas de busca locais e o desenvolvimento de um algoritmo VNS considerando uma arquitetura paralela.

## Referências

Ahuja, R., Ergun, O. and Orlin, James B.and Punnen, A. (2002). A survey of very large-scale neighborhood search techniques, *Discrete Applied Mathematics* **123**: 75–102.

Arkin, E. and Roundy, R. (1991). Weighted-tardiness scheduling on paralell machines with proportional weights, *Operations Research* **39**: 64–81.

Barnes, J. and Brennan, J. (1977). A improved algorithm for scheduling jobs on identical machines, *IEE Transactions* **9**: 25–31.

Błażewicz, J., Ecker, K., Pesch, E., G., S. and Węglarz, J. (1996). *Scheduling Computer and Manufacturing Processes*, Springer - Verlag, Berlin.

Brucker, P. (2004). *Scheduling Algorithms*, Springer-Verlag, Berlin.

Chen, Z. and Powell, L. (1999). A column generation based decomposition algorithm for a parallel machine just-in-time scheduling problem, *European Journal of Operational Research* **116**: 220–232.

Dogramaci, A. (1984). Production scheduling of independent jobs on parallel identical machines, *International Journal of Production Research* **16**: 535–548.

Elmagraby, S. and Park, S. (1974). Scheduling jobs on a number of identical machines, *IEE Transactions* **6**: 1–13.

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

Feo, T., Sarathy, K. and McGahan, J. (1996). A grasp for single machine scheduling with sequence dependent setup costs and linear delay penalties, *Computers & Operations Research* (23): 881–895.

Festa, P. and Resende, M. (2002). GRASP: An annotated bibliography, *in* C. Ribeiro and P. Hansen (eds), *Essays and surveys in metaheuristics*, Kluwer Academic Publishers, pp. 325–367.

García-López, F., Melián-Batista, B., Moreno-Pérez, J. and Moreno-Vega, J. M. (2002). The parallel variable neighborhood search for the p-median problem, *Journal of Heuristics* **8**: 275–388.

Garey, M. and Johnson, D. (1997). *Computers and Intractability: A Guide to the Theory of NP-Completness*, W. H. Freeman.

Gomez Ravetti, M., L., R. P., R, M. G. and Pardalos, P. (2006). A scheduling problem with unrelated parallel machines and sequence dependent setups. International Journal of Operational Research. To appear.

Hansen, P. and Mladenovic, N. (1999). Variable neighborhood search: Principles and applications, *European journal of operational reasearch* **130**: 449–467.

Hansen, P. and Mladenovic, N. (2002). Variable neighborhood search, *in* P. M. Pardalos and M. G. C. Resende (eds), *Handbook of Applied Optimization*, Oxford University Press, New York.

Hansen, P. and Mladenovic, N. (2003). A tutorial on variable neighborhood search, *Le cahiers du GERARD* **G-2003**: 46.

Hansen, P., Mladenovic, N. and Brimberg, J. (2002). Convergence of variable neighborhood search, *Les Cahiers du GERAD* **G-2002-21**.

Johnson, D., Aragon, C., McGeoch, L. and Schevon, C. (1989). Optimization by simulated annealing: An experimental evaluation; part 1, graph partitioning, *Operations Research* **37**: 865–892.

Lee, C.-Y. and Pinedo, M. (2002). Optimization and heuristics of scheduling, *in* P. M. Pardalos and M. G. C. Resende (eds), *Handbook of Applied Optimization*, Oxford University Press, New York.

Matsuo, H., Sug, C. and Sullivan, R. (1989). A controlled search simulated annealing method for the single machine weighted tardiness problem, *Annals of Operations Research* **21**: 85–108.

Nawaz, M., Enscore, E. and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, *Omega* (11): 91–95.

Pinedo, M. (1995). *Scheduling Theory, Algorithm and System*, Prentice Hall.

Polacek, M., F. Hartl, R. and Doerner, K. (2004). A variable neighborhood search for the multi depot vehicle routing problem with time windows, *Journal of Heuristics* **10**: 613–627.

Resende, M. and Ribeiro, C. (2003). GRASP and path-relinking: Recent advances and applications, *in* T. Ibaraki and Y. Yoshitomi (eds), *Proceedings of the Fifth Metaheuristics International Conference (MIC2003)*, pp. T6–1 – T6–6.

Rocha, P. L., Gomez Ravetti, M., R, M. G. and Pardalos, P. (2006). Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine dependent setup times. Submitted to Computers & Operations Researc. (Second Review).

Rocha, P. L., Ravetti, M. G. and Mateus, G. R. (2004). The metaheuristic grasp as an upper bound for a branch and bound algorithm in a scheduling problem with non-related parallel machines and sequence-dependent setup times, *EU/ME Workshop*, Nottingham, Inglaterra.

Vasilescu, E. and Amar, A. (1983). An empirical evaluation of the entrapment procedure for scheduling jobs on identical machines, *IEE Transactions* **15**: 261–263.