# Um GRASP Simples e Robusto para o *Job Shop Scheduling Problem*

**Dayan de Castro Bissoli**
Universidade Federal do Espírito Santo (UFES)
Av. Fernando Ferrari, 514 – Goiabeiras – 29075910 – Vitória/ES – Brasil
dcbissoli@inf.ufes.br

**André Renato Sales Amaral**
Universidade Federal do Espírito Santo (UFES)
Av. Fernando Ferrari, 514 – Goiabeiras – 29075910 – Vitória/ES – Brasil
amaral@inf.ufes.br

## RESUMO

No problema conhecido como *job shop scheduling problem* (JSP), temos um conjunto de *jobs* e um conjunto de máquinas. Um *job* é caracterizado por uma ordem fixa de operações. Cada uma dessas operações deve ser processada em uma máquina específica, e cada uma dessas máquinas pode processar no máximo uma tarefa de cada vez, respeitando a restrição de que, antes de iniciar uma nova tarefa, deve-se finalizar a atual. O escalonamento (*scheduling*) é uma atribuição das operações a intervalos de tempo nas máquinas. O objetivo do JSP é encontrar um escalonamento que minimiza o tempo máximo de conclusão (*makespan*) dos *jobs*. Este artigo descreve um algoritmo GRASP para o JSP. Experimentos computacionais com um conjunto padrão de instâncias do problema indicaram que a nossa implementação do GRASP é robusta e competitiva para encontrar soluções aproximadas para o JSP.

**PALAVRAS CHAVE. *Job Shop*, GRASP, *Scheduling*, Área de classificação principal (MH – Metaheuristicas).**

## ABSTRACT

In the job shop scheduling problem (JSP), we have a set of jobs and a set of machines. A job is characterized by a fixed order of operations. Each of these operations must be processed on a specific machine, and each of these machines can process at most one job at a time, respecting the restriction that, before starting a new task, must end the current. The scheduling is a assignment of time slots operations on machines. The purpose of JSP is to find a schedule that minimizes the maximum completion time (makespan) of jobs. This article describes a GRASP algorithm for JSP. Computational experiments with a standard set of instances of the problem indicate that our implementation of GRASP is robust and competitive to find approximate solutions to the JSP.

**KEYWORDS. Job Shop, GRASP, Scheduling, Main area (MH – Metaheuristics).**

## 1. Introdução

O problema *job shop* clássico (JSP) pode ser descrito da seguinte forma: Nos são dados $n$ *jobs*, cada um composto de várias operações. Esses *jobs* devem ser processados em $m$ máquinas. Cada operação utiliza uma das $m$ máquinas por uma duração fixa de tempo. Cada máquina pode processar no máximo uma operação por vez; e uma vez que uma operação inicie o processamento em uma determinada máquina, esta operação deve ser concluída sem interrupção. As operações que compõe um determinado *job* devem ser processadas em uma determinada sequência, isto é, respeitando uma ordem de precedência. O problema consiste em encontrar um escalonamento das operações nas máquinas, tendo em vista as restrições de precedência, de modo a minimizar o $Cmax$ (*makespan*), isto é, o tempo de finalização da última operação completada no escalonamento.

O JSP pode ser formulado matematicamente como segue. Dado um conjunto $M$ de máquinas (considere o tamanho de $M$ como $|M|$) e o conjunto $J$ de *jobs* (considere o tamanho de $J$ como $|J|$), seja $\sigma_1^j \prec \sigma_2^j \prec \sigma_{|M|}^j$ o conjunto ordenado das $|M|$ operações do *job* $j$, onde $\sigma_k^j \prec \sigma_{(k+1)}^j$ indica que a operação $\sigma_{(k+1)}^j$ pode iniciar somente depois de completar a operação $\sigma_k^j$. Seja $O$ o conjunto de operações. Cada operação $\sigma_k^j$ é definida por dois parâmetros: $M_k^j$ é a máquina onde $\sigma_k^j$ é processada e $p_k^j = p(\sigma_k^j)$ é o tempo de processamento da operação $\sigma_k^j$. Definindo $t(\sigma_k^j)$ como o instante de início da $k$-ésima operação $\sigma_k^j \in O$, uma formulação para o JSP é apresentada a seguir (Aiex et al., 2003):

$$\text{Minimizar } C_{max} \tag{1}$$

$$\text{Sujeito a: } C_{max} \geq t(\sigma_k^j) + p(\sigma_k^j), \quad \forall \sigma_k^j \in O, \tag{2}$$

$$t(\sigma_k^j) \geq t(\sigma_l^j) + p(\sigma_l^j), \quad \forall \sigma_l^j \prec \sigma_k^j, \tag{3}$$

$$t(\sigma_k^j) \geq t(\sigma_l^i) + p(\sigma_l^i) \vee \tag{4}$$

$$t(\sigma_l^i) \geq t(\sigma_k^j) + p(\sigma_k^j), \quad \forall i,j \in J : M_{\sigma_l^i} = M_{\sigma_k^j}, \tag{5}$$

$$t(\sigma_k^j) \geq 0, \quad \forall \sigma_k^j, \in O. \tag{6}$$

onde $C_{max}$ é o *makespan* a ser minimizado.

Uma solução viável para o JSP pode ser gerada a partir da permutação de $J$ em cada uma das máquinas em $M$, (observando as restrições de precedência, a restrição de que a máquina pode processar apenas uma operação por vez, e exigindo que uma vez iniciado, o processamento de uma operação deve ser ininterrupto até a sua conclusão). Cada conjunto de permutações tem um escalonamento correspondente. Assim, o objetivo do JSP é encontrar um conjunto de permutações com o menor valor de *makespan* (Binato et al., 2001).

A Figura 1 ilustra o gráfico de Gannt de uma solução aproximativa para uma instância do JSP conhecida como ft06 (Fisher e Thompson, 1963). No gráfico, o conjunto de máquinas é disposto no eixo vertical e a escala do tempo é indicada no eixo horizontal, estabelecendo uma barra horizontal para cada tempo de processamento de cada operação em cada máquina.

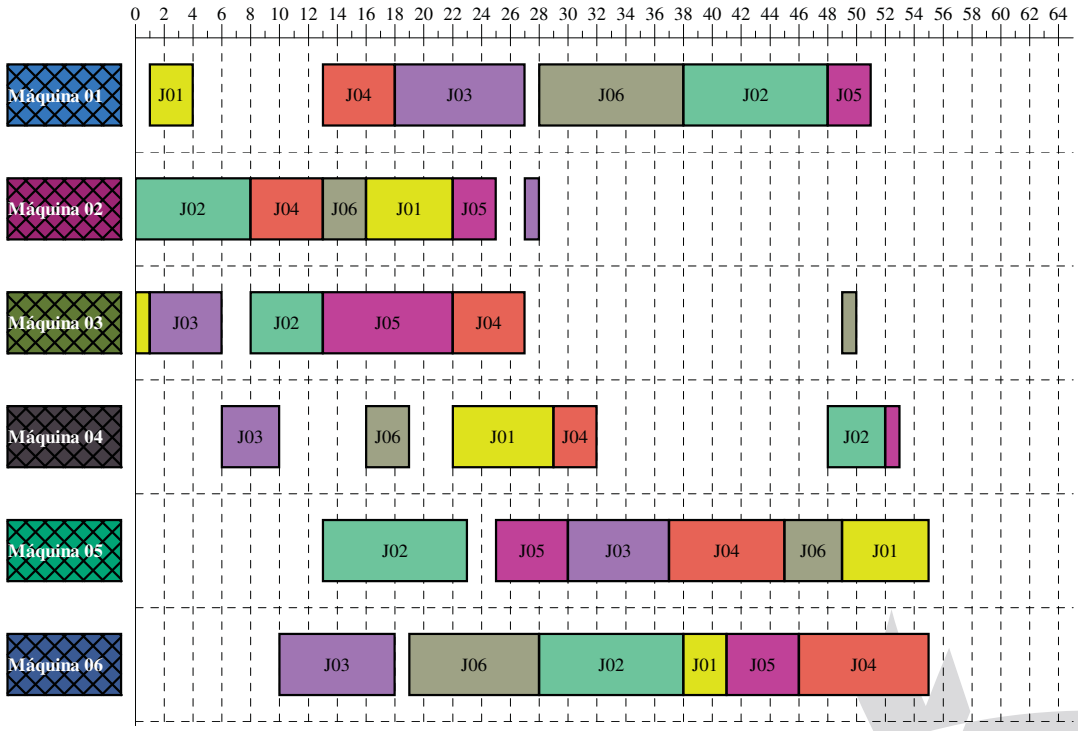


**Figura 1. Exemplo de escalonamento no ambiente *Job Shop* - Solução da instância ft06.**

O JSP é *NP-hard* (Laguna e Martí, 1999), mesmo para instâncias com três máquinas e tempos unitários de processamento, assim como para instâncias com três *jobs*.

Dessa forma, no presente trabalho implementou-se um algoritmo baseado na metaheurística GRASP. Os experimentos computacionais com um conjunto padrão de instâncias para o problema indicaram que a nossa implementação do GRASP é competitiva para encontrar soluções aproximativas para o JSP.

Este artigo está organizado da seguinte forma. Na próxima seção, apresentamos uma breve revisão da metaheurística GRASP, seguida de sua aplicação para o JSP. Os experimentos computacionais são relatados na seção 4 e na sequência, as conclusões.

## 2. Breve revisão do GRASP

De acordo com (Feo e Resende, 1995) e Festa e Resende (2009), o GRASP é um processo iterativo, onde a cada iteração, são realizadas duas fases: construção e busca local. A fase de construção gera uma solução viável, cuja vizinhança é explorada pela busca local. A melhor solução encontrada durante a execução do algoritmo GRASP é retornada como resultado.

Na fase de construção, considera-se que uma solução para um problema consista de vários elementos, que são adicionados um por vez. Dessa forma, a cada iteração da fase de construção, no máximo $|J|$ *jobs* podem ser escalonados, e este conjunto de *jobs* candidatos é indicado por $O_c$. Os elementos ainda não adicionados à solução são ordenados em uma lista de candidatos de acordo com uma função gulosa que mede a melhora que seria obtida no valor da solução pela adição de cada elemento. Considere que a função adaptativa

gulosa dada por $h(\tau)$ denote o *makespan* resultante a partir da adição do *job* $\tau$ aos *jobs* já escalonados. A escolha do próximo *job* a ser escalonado é dado por:

$$\underline{\tau} = min\,(h(\tau)|\tau \in O_c).$$

Definindo,

$$\overline{\tau} = max\,(h(\tau)|\tau \in O_c),$$

$\underline{h} = h\,(\underline{\tau})$, e $\overline{h} = h\,(\overline{\tau})$, a Lista Restrita de Candidatos (LCR) é definida como:

$$LCR = \left\{\tau \in O_c|\underline{h} \leq h(\tau) \leq \underline{h} + \alpha(\overline{h} - \underline{h})\right\},$$

onde o parâmetro $\alpha$ restringe-se a $0 \leq \alpha \leq 1$.

Para $\alpha = 0$ são geradas soluções totalmente gulosas, e $\alpha = 1$, são geradas soluções totalmente aleatórias. O componente adaptativo da heurística decorre do fato de que os benefícios associados a cada elemento são atualizados a cada iteração da fase de construção para refletir as mudanças trazidas pela seleção dos elementos anteriores. O componente probabilístico do GRASP é caracterizado pela escolha aleatória de um dos melhores candidatos na LRC, mas geralmente não o melhor. Essa forma de escolha permite que diferentes soluções sejam obtidas ao final da fase de construção do GRASP, embora não necessariamente comprometa o componente adaptativo guloso.

Uma solução gerada pelo GRASP na fase construtiva não possui garantia de ser localmente ótima em relação às simples definições de vizinhança. Por este motivo, aplica-se uma busca local para tentar melhorar cada solução gerada na fase construtiva. Um algoritmo de busca local funciona de forma iterativa, substituindo sucessivamente a solução atual por uma solução melhor, em sua vizinhança. Define-se como seu critério de parada o fato de não encontrar nenhuma solução melhor na vizinhança, em relação a alguma função de custo.

O Algoritmo 1 ilustra o pseudocódigo de implementação genérica do GRASP. A entrada para o GRASP inclui parâmetros para definir o tamanho da lista de candidatos (*ListSize*), o número máximo de iterações (*MaxIter*), e a semente para o gerador de números aleatórios (*RandomSeed*). As iterações do GRASP são realizadas nas linhas 3-9. Cada iteração do GRASP consiste na fase de construção (linha 4), fase de busca local (linha 5) e, se necessário, atualização da melhor solução encontrada (linha 7). Dessa forma, após chegar ao limite de número de iterações definido, o algoritmo retorna a melhor solução encontrada (linha 10).

A aplicação do GRASP ao JSP, é descrita na próxima seção.

## 3. Um algoritmo GRASP para o JSP

Nesta seção é apresentado um algoritmo GRASP para o JSP. Inicialmente é descrito o esquema básico de construção e posteriormente apresenta-se o algoritmo de busca local.

### 3.1. Fase de construção

A fase de construção do GRASP gera uma solução viável, conforme ilustrado na Figura 2. Na citada figura, apresentam-se os vértices do grafo, representando cada tempo de processamento e dois tipos de arcos: Conjuntivo e Disjuntivo. O conjunto de arcos conjuntivos são relativos à sequência de operações de um *job*, ou seja, tais arcos representam

**Algoritmo 1** Pseudocódigo genérico do GRASP

```
1:  GRASP ( ListSize, MaxIter, RandomSeed )
2:  MIN ← ∞;
3:  Para 0 até maxIter faça
4:     x ← ConstruçãoGrasp();
5:     x ← BuscaLocal(x);
6:     Se x é viável e f(x) < MIN então
7:        x* ← x ;
8:        MIN ← f(x) ;
9:     Fim-se;
10: Fim-para;
11: Retorne x*;
```

as restrições de precedência entre as operações de um mesmo *job*. O conjunto de arcos disjuntivos são correspondentes às limitações dos recursos, não possuindo direção, representando o par de operações de diferentes *jobs* a serem executadas na mesma máquina, sendo que a escolha de uma direção desses arcos estabelecerá a ordem de execução dos *jobs* na mesma máquina.

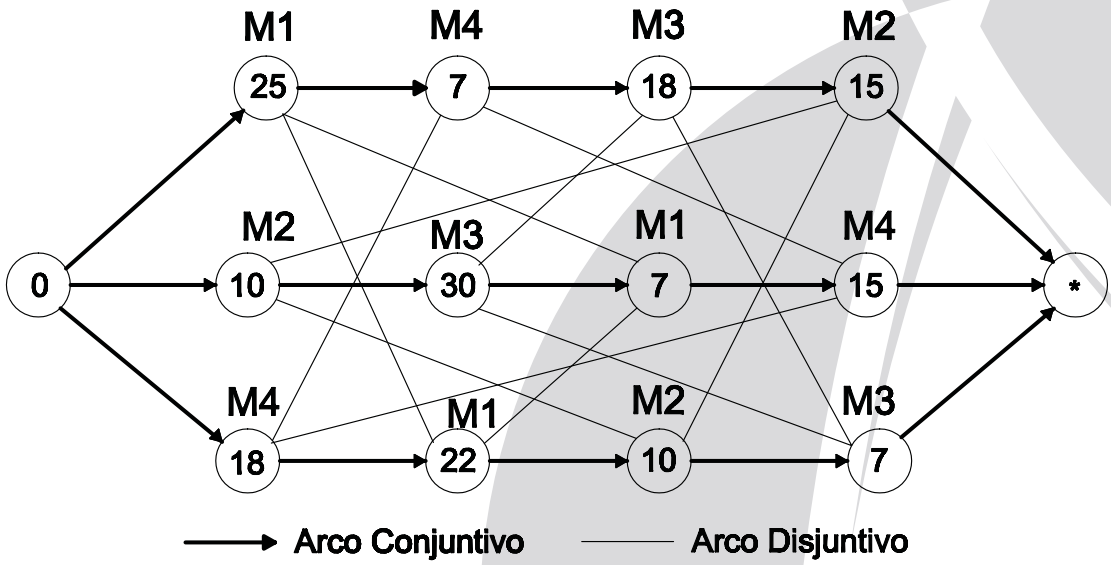


**Figura 2. Grafo disjuntivo: representação de um escalonamento. O objetivo é obter caminho hamiltoniano em cada clique de tarefas relacionada a cada máquina.**

Fonte: Adaptado de (Morales, 2012).

No JSP, considera-se um único *job* para ser o bloco de construção nesta fase inicial. Ou seja, é gerado um escalonamento viável alocando operações individuais, uma por vez, até que todas as operações sejam inseridas no escalonamento. A próxima operação a ser agendada é escolhida aleatoriamente da LRC, na qual todos os candidatos possuem a mesma probabilidade de serem selecionados.

### 3.2. Busca Local

A geração da vizinhança de soluções e, consequentemente, a lista de possíveis movimentos a serem realizados, são determinadas a partir do caminho crítico (*critical path*), definido como o maior caminho entre o início de processamento dos *jobs* até a conclusão de todos os *jobs* em todas as máquinas. Ao calcular o *makespan*, a partir do caminho crítico, conforme descrito por Taillard (1994), guarda-se o caminho de vértices mais longo do grafo, conforme apresentado na Figura 3. Realiza-se trocas na sequência de operações, respeitando as ordens de precedências. Dessa forma, foi adotada a estratégia de escolher o melhor movimento dentre os movimentos possíveis do caminho crítico, para obter a nova solução.

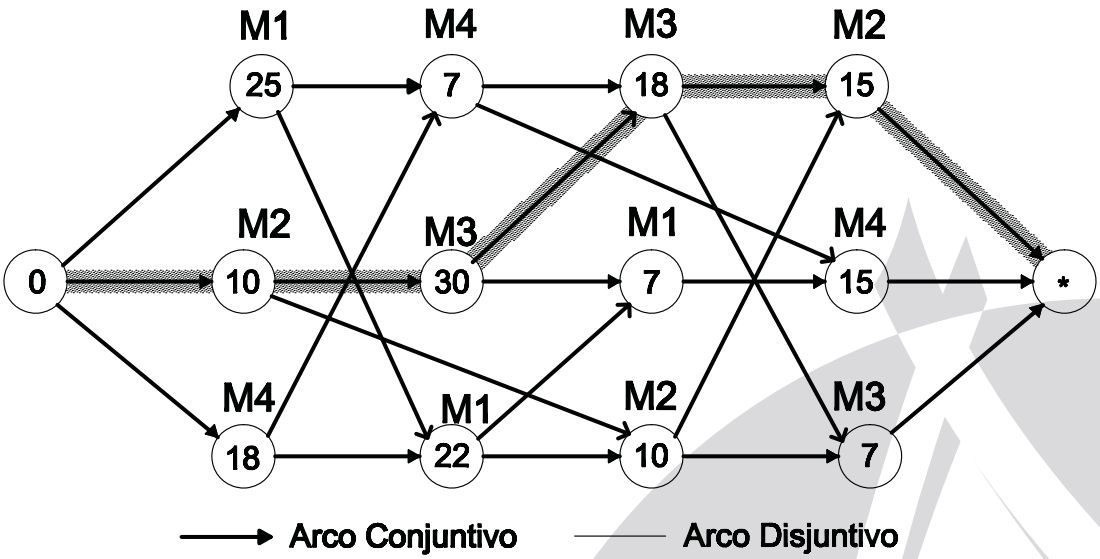


**Figura 3. Grafo disjuntivo: caminho crítico (*Makespan* = 73).**
Fonte: Adaptado de (Morales, 2012).

A Figura 4 apresenta uma solução após a aplicação da busca local, no qual foi minimizado o valor da Função Objetivo de 73 para 62. A seguir são apresentados os experimentos computacionais.

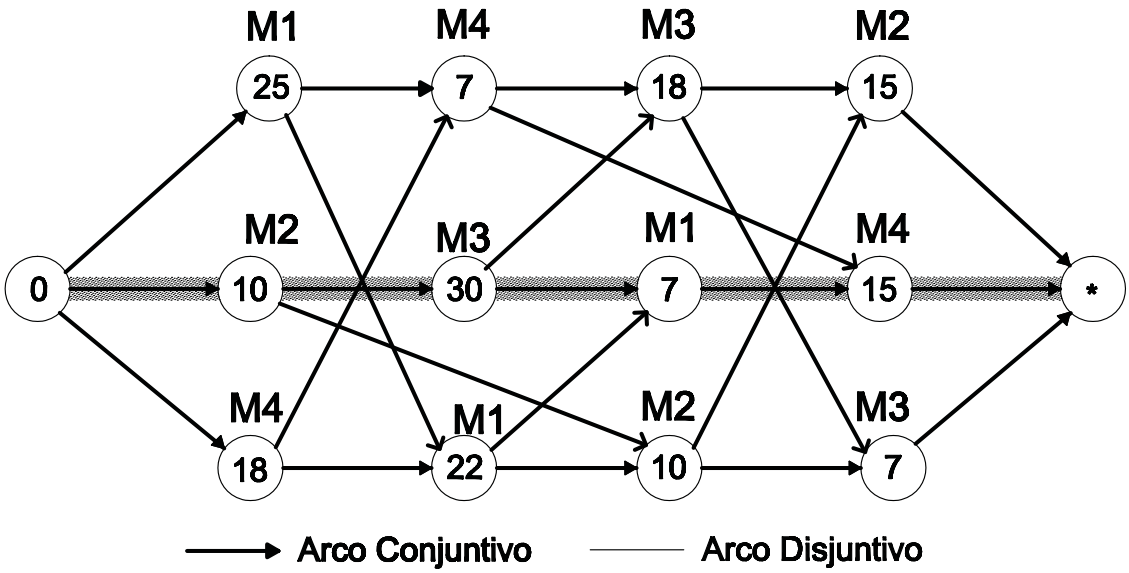


**Figura 4. Grafo disjuntivo: caminho crítico minimizado após aplicação da busca local (*Makespan* = 62).**

Fonte: Adaptado de (Morales, 2012).

## 4. Experimentos Computacionais

Para ilustrar a eficácia do algoritmo GRASP descrito neste artigo, considera-se 43 instâncias tradicionais do JSP: Instâncias FT06, FT10 e FT20 de Fisher e Thompson (1963) e instâncias LA01 a LA40 de Lawrencen (1984). Esse conjunto de instâncias apresenta-se em várias configurações desde 6 a 30 *jobs* e 5 a 15 máquinas.

O algoritmo GRASP foi implementado em linguagem C e compilado com o compilador GCC e os testes foram executados em um computador com processador Intel Core i5-2450M com 2.50Ghz e sistema operacional Windows 8.1 Pro.

Em relação à parametrização do algoritmo GRASP, o parâmetro $\alpha$ foi definido como 0,5 e o número de iterações como 10000. O algoritmo GRASP foi executado dez vezes, para cada instância.

A Tabela 1 apresenta os resultados da seguinte forma: Coluna I apresenta o nome da instância utilizada; a quantidade de *jobs* e a quantidade de máquinas são representados por J e M, respectivamente; A coluna BKS (*best known solution*) representa a melhor solução conhecida; A coluna AGH apresenta as melhores soluções geradas pelo algoritmo genético híbrido (AGH) em Gonçalves et al. (2005); A coluna S2 apresenta as melhores soluções encontradas pelo algoritmo GRASP proposto; A média das 10 soluções geradas para cada instância pelo algoritmo GRASP proposto é relacionada na coluna Média S2; O desvio médio entre as dez soluções geradas pelo algoritmo GRASP proposto apresenta-se na coluna Desvio; Os Gap's entre S2 e BKS e AGH e BKS são apresentados nas colunas Gap S2 e Gap AGH, respectivamente.

A partir da Tabela 1, percebe-se que tanto o GRASP quando o AGH atingem o valor da melhor solução conhecida, ou próximo dele, com uma ligeira vantagem para o

**Tabela 1. Resultados Computacionais**

| I | J | M | BKS* | AGH | S2 | Média S2 | Desvio | Gap S2 | Gap AGH |
|---|---|---|---|---|---|---|---|---|---|
| ft06 | 6 | 6 | 55 | 55 | 55 | 55 | 0,0% | 0,0% | 0,0% |
| ft10 | 10 | 10 | 930 | 930 | 973 | 982,7 | 1,0% | 4,6% | 4,6% |
| ft20 | 20 | 5 | 1165 | 1165 | 1196 | 1207,8 | 1,0% | 2,7% | 2,7% |
| la01 | 10 | 5 | 666 | 666 | 666 | 666,00 | 0,0% | 0,0% | 0,0% |
| la02 | 10 | 5 | 655 | 655 | 656 | 662,70 | 1,0% | 0,2% | 0,2% |
| la03 | 10 | 5 | 597 | 597 | 628 | 633,70 | 0,9% | 5,2% | 5,2% |
| la04 | 10 | 5 | 590 | 590 | 598 | 598,00 | 0,0% | 1,4% | 1,4% |
| la05 | 10 | 5 | 593 | 593 | 593 | 593,00 | 0,0% | 0,0% | 0,0% |
| la06 | 15 | 5 | 926 | 926 | 926 | 926,00 | 0,0% | 0,0% | 0,0% |
| la07 | 15 | 5 | 890 | 890 | 890 | 894,00 | 0,4% | 0,0% | 0,0% |
| la08 | 15 | 5 | 863 | 863 | 863 | 863,00 | 0,0% | 0,0% | 0,0% |
| la09 | 15 | 5 | 951 | 951 | 951 | 951,00 | 0,0% | 0,0% | 0,0% |
| la10 | 15 | 5 | 958 | 958 | 958 | 958,00 | 0,0% | 0,0% | 0,0% |
| la11 | 20 | 5 | 1222 | 1222 | 1222 | 1222,00 | 0,0% | 0,0% | 0,0% |
| la12 | 20 | 5 | 1039 | 1039 | 1039 | 1039,00 | 0,0% | 0,0% | 0,0% |
| la13 | 20 | 5 | 1150 | 1150 | 1150 | 1150,00 | 0,0% | 0,0% | 0,0% |
| la14 | 20 | 5 | 1292 | 1292 | 1292 | 1292,00 | 0,0% | 0,0% | 0,0% |
| la15 | 20 | 5 | 1207 | 1207 | 1210 | 1224,00 | 1,2% | 0,2% | 0,2% |
| la16 | 10 | 10 | 945 | 945 | 979 | 979,80 | 0,1% | 3,6% | 3,6% |
| la17 | 10 | 10 | 784 | 784 | 795 | 804,00 | 1,1% | 1,4% | 1,4% |
| la18 | 10 | 10 | 848 | 848 | 861 | 874,20 | 1,5% | 1,5% | 1,5% |
| la19 | 10 | 10 | 842 | 842 | 856 | 863,80 | 0,9% | 1,7% | 1,7% |
| la20 | 10 | 10 | 902 | 907 | 914 | 919,90 | 0,6% | 1,3% | 0,8% |
| la21 | 15 | 10 | 1046 | 1046 | 1133 | 1154,10 | 1,9% | 8,3% | 8,3% |
| la22 | 15 | 10 | 927 | 935 | 1020 | 1032,20 | 1,2% | 10,0% | 9,1% |
| la23 | 15 | 10 | 1032 | 1032 | 1057 | 1068,80 | 1,1% | 2,4% | 2,4% |
| la24 | 15 | 10 | 935 | 953 | 1029 | 1040,30 | 1,1% | 10,1% | 8,0% |
| la25 | 15 | 10 | 977 | 986 | 1064 | 1089,20 | 2,4% | 8,9% | 7,9% |
| la26 | 20 | 10 | 1218 | 1218 | 1348 | 1374,10 | 1,9% | 10,7% | 10,7% |
| la27 | 20 | 10 | 1235 | 1256 | 1375 | 1394,30 | 1,4% | 11,3% | 9,5% |
| la28 | 20 | 10 | 1216 | 1232 | 1330 | 1365,40 | 2,7% | 9,4% | 8,0% |
| la29 | 20 | 10 | 1157 | 1196 | 1366 | 1382,50 | 1,2% | 18,1% | 14,2% |
| la30 | 20 | 10 | 1355 | 1355 | 1417 | 1443,80 | 1,9% | 4,6% | 4,6% |
| la31 | 30 | 10 | 1784 | 1784 | 1822 | 1837,10 | 0,8% | 2,1% | 2,1% |
| la32 | 30 | 10 | 1850 | 1850 | 1911 | 1944,70 | 1,8% | 3,3% | 3,3% |
| la33 | 30 | 10 | 1719 | 1719 | 1747 | 1786,70 | 2,3% | 1,6% | 1,6% |
| la34 | 30 | 10 | 1721 | 1721 | 1827 | 1841,30 | 0,8% | 6,2% | 6,2% |
| la35 | 30 | 10 | 1888 | 1888 | 1915 | 1951,20 | 1,9% | 1,4% | 1,4% |
| la36 | 15 | 15 | 1268 | 1279 | 1398 | 1412,30 | 1,0% | 10,3% | 9,3% |
| la37 | 15 | 15 | 1397 | 1408 | 1503 | 1517,20 | 0,9% | 7,6% | 6,7% |
| la38 | 15 | 15 | 1196 | 1219 | 1303 | 1325,30 | 1,7% | 8,9% | 6,9% |
| la39 | 15 | 15 | 1233 | 1246 | 1329 | 1349,80 | 1,6% | 7,8% | 6,7% |
| la40 | 15 | 15 | 1222 | 1241 | 1309 | 1319,10 | 0,8% | 7,1% | 5,5% |
| **Média** | | | | | | | **0,9%** | **4,0%** | **3,6%** |

*Fonte: Binato et al. (2001).

AGH. Porém, ressalta-se a simplicidade do algoritmo GRASP proposto nesse estudo e a robustez do mesmo, considerando que o desvio médio entre as soluções geradas a partir das dez execuções para cada instância ficou em menos de $1\%$, no geral.

A Tabela 2 apresenta-se da seguinte forma: Coluna I representa o nome da instância utilizada; a quantidade de *jobs* e a quantidade de máquinas são listados em J e M, respectivamente; O tempo total referente as dez execuções do GRASP proposto é apresentado na coluna T(s) AGH, e o tempo total de execução do AGH de Gonçalves et al. (2005) para cada instância é informado na coluna T(s) AGH.

O tempo do AGH consiste em uma única execução do algoritmo, e é apresentado na Tabela 2 apenas como referência pois como os algoritmos foram executados em computadores com configurações diferentes, esses tempos não podem ser diretamente comparados.

A parametrização informada foi escolhida por ter apresentado a melhor média no conjunto total de instâncias, sendo que em alguns casos, ao variar o parâmetro $\alpha$ em algumas instâncias, a melhor solução foi melhor do que a apresentada na Tabela 1, mas consequentemente piorando o desvio entre as execuções.

## 5. Conclusões

Este trabalho apresenta uma simples e robusta implementação do GRASP para o JSP clássico. A fase construtiva gera uma solução viável, um elemento por vez, sendo que a próxima operação a ser agendada é escolhida aleatoriamente da LRC, na qual todos os candidatos possuem a mesma probabilidade de serem selecionados. Por conseguinte, é aplicada uma busca local, que consiste em calcular o *makespan*, e ao obter o caminho mais longo do grafo disjuntivo, que representa o escalonamento de maior custo, escolhe o melhor movimento dentre os possíveis, para obter a nova solução.

O algoritmo GRASP proposto foi testado em um conjunto de 43 instâncias padrões e comparado com o algoritmo genético híbrido (AGH) proposto por Gonçalves et al. (2005). Os resultados computacionais mostram que o algoritmo GRASP atinge os mesmos valores das melhores soluções conhecidas (BKS) ou próximos em todas as instâncias testadas. De maneira geral, o algoritmo GRASP produz resultados com desvio relativo médio de $3,6\%$ em relação ao BKS.

## 6. Agradecimentos

Dayan de Castro Bissoli agradece a Bolsa de Doutorado da Fundação de Amparo à Pesquisa do Espírito Santo (FAPES).

## Referências

**Aiex, R. M., Binato, S., and Resende, M. G. C.** (2003). Parallel grasp with path-relinking for job shop scheduling. *Parallel Computing*, 29:393–430.

**Binato, S., Hery, W., Loewenstern, D., and Resende, M. G. C.** (2001). A grasp for job shop scheduling. In Ribeiro, C. and Hansen, P., editors, *Essays and Surveys on Metaheuristics*, pages 59–79. Kluwer Academic Publishers.

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

**Festa, P. and Resende, M. G. C.** (2009). An annotated bibliography of grasp – part i: Algorithms. *International Transactions in Operational Research*, 16:1–24.

**Tabela 2. Tempos computacionais**

| I | J | M | T (s) S2* | T (s) AGH** |
|---|---|---|---|---|
| ft06 | 6 | 6 | 9,32 | 13 |
| ft10 | 10 | 10 | 79,14 | 292 |
| ft20 | 20 | 5 | 174,02 | 204 |
| la01 | 10 | 5 | 25,57 | 37 |
| la02 | 10 | 5 | 26,31 | 51 |
| la03 | 10 | 5 | 24,93 | 39 |
| la04 | 10 | 5 | 25,91 | 42 |
| la05 | 10 | 5 | 26,82 | 32 |
| la06 | 15 | 5 | 64,35 | 99 |
| la07 | 15 | 5 | 49,37 | 86 |
| la08 | 15 | 5 | 64,25 | 99 |
| la09 | 15 | 5 | 60,12 | 94 |
| la10 | 15 | 5 | 60,14 | 91 |
| la11 | 20 | 5 | 120,41 | 197 |
| la12 | 20 | 5 | 128,54 | 201 |
| la13 | 20 | 5 | 133,68 | 189 |
| la14 | 20 | 5 | 99,37 | 189 |
| la15 | 20 | 5 | 110,98 | 189 |
| la16 | 10 | 10 | 57,12 | 232 |
| la17 | 10 | 10 | 55,58 | 216 |
| la18 | 10 | 10 | 54,86 | 219 |
| la19 | 10 | 10 | 55,95 | 235 |
| la20 | 10 | 10 | 60,67 | 235 |
| la21 | 15 | 10 | 171,95 | 602 |
| la22 | 15 | 10 | 170,53 | 629 |
| la23 | 15 | 10 | 152,29 | 594 |
| la24 | 15 | 10 | 152,56 | 578 |
| la25 | 15 | 10 | 170,86 | 609 |
| la26 | 20 | 10 | 321,62 | 1388 |
| la27 | 20 | 10 | 337,13 | 1251 |
| la28 | 20 | 10 | 423,75 | 1267 |
| la29 | 20 | 10 | 646,81 | 1350 |
| la30 | 20 | 10 | 340,64 | 1260 |
| la31 | 30 | 10 | 1004,45 | 3745 |
| la32 | 30 | 10 | 999,82 | 3741 |
| la33 | 30 | 10 | 828,68 | 3637 |
| la34 | 30 | 10 | 947,88 | 3615 |
| la35 | 30 | 10 | 782,04 | 3716 |
| la36 | 15 | 15 | 261,01 | 1826 |
| la37 | 15 | 15 | 266,55 | 1860 |
| la38 | 15 | 15 | 273,35 | 1859 |
| la39 | 15 | 15 | 242,00 | 1869 |
| la40 | 15 | 15 | 289,14 | 2185 |

*Computador com CPU Intel(R) Core(TM) i5-2450M 2,50GHz e sistema operacional MS Windows 8.1 Pro.
**Computador com CPU AMD Thunderbird 1,333 GHz e sistema operacional MS Windows Me.

**Fisher, H. and Thompson, G. L.** (1963). Probabilistic learning combinations of local job-shop scheduling rules. In Muth, J. F. and Thompson, G. L., editors, *Industrial Scheduling*, pages 225–251. Prentice-Hall, Englewood Cliffs, NJ.

**Gonçalves, J. F., Mendes, J. J. M., and Resende, M. G. C.** (2005). Proposta de classificação hierarquizada dos modelos de solução para o problema de job shop scheduling. *Gestão & Produção*, 6(1):44–52.

**Laguna, M. and Martí, R.** (1999). Grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52.

**Lawrencen, B. J.** (1984). Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques. In *Supplement to resource constrained project scheduling*, pages 225–251. GSIA, Carnegie Mellon University, Pittsburgh, PA.

**Morales, S. W. G.** (2012). Formulações matemáticas e estratégias de resolução para o problema job shop clássico. Dissertação de mestrado, Universidade de São Paulo - USP.

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