# GBT: GRASP + BUSCA TABU EM PROBLEMAS DE SCHEDULING

**Luís Henrique Costa Bicalho**
Departamento de Informática
Universidade Federal de Viçosa – UFV
Av. P. H. Rolfs, s/n – DPI/UFV – 36570-000 – Viçosa, MG
luis.bicalho@ufv.br

**André Gustavo dos Santos**
Departamento de Informática
Universidade Federal de Viçosa – UFV
Av. P. H. Rolfs, s/n – DPI/UFV – 36570-000 – Viçosa, MG
andre@dpi.ufv.br

**José Elias Cláudio Arroyo**
Departamento de Informática
Universidade Federal de Viçosa – UFV
Av. P. H. Rolfs, s/n – DPI/UFV – 36570-000 – Viçosa, MG
jarroyo@dpi.ufv.br

## RESUMO

Este trabalho propõe uma metaheurística híbrida baseada em GRASP e Busca Tabu para tratar dois problemas de programação de tarefas: o RASDST (*Resource-assignable Sequence Dependent Setup Time*) e o WTSSDS (*Weighted Tardiness Scheduling Problem with Sequence-Dependent Setup Times*). No RASDST devemos programar as tarefas em um conjunto de máquinas paralelas distintas, considerando tempos de preparação (*setup times*) e recursos, com o objetivo de minimizar o *flowtime* e a quantidade de recursos utilizados para preparar as máquinas. Já no WTSSDS, as tarefas devem ser alocadas em uma única máquina, com tempo de preparação entre as tarefas, a fim de minimizar o atraso total ponderado, já que toda tarefa tem um peso associado. O algoritmo proposto, denominado GBT, foi testado em instâncias disponíveis na literatura, e, em ambos os problemas, os resultados obtidos foram melhores do que a maioria dos resultados já publicados.

**PALAVRAS CHAVE: Metaheurística. Busca Tabu. Scheduling.**
**Área principal: MH – Metaheurísticas.**

## ABSTRACT

This paper proposes a hybrid metaheuristic based on GRASP and Tabu Search to solve two scheduling problems: the RASDST (Resource-assignable Sequence Dependent Setup Time) and the WTSSDS (Weighted Tardiness Scheduling Problem with Sequence-Dependent Setup Times). In the RASDST we schedule the jobs in a set of unrelated parallel machines, considering setup times and resources, and the goal is minimize the flowtime and the number of resources used to prepare the machines. In the WTSSDS, the jobs should be allocated on a single machine, with setup times between the jobs, to minimize the weighted tardiness, since the jobs has an associate weight. The proposed algorithm, called GBT, was tested on instances from literature and, in both problems, the results were better than most of the results already published.

**KEYWORDS: Metaheuristics. Tabu Search. Scheduling.**
**Main area: MH - Metaheuristics**

## 1. Introdução

Os problemas de programação de tarefas (*job scheduling problems*) são estudados exaustivamente devido à sua aplicação em diversos processos produtivos, como, por exemplo, manufaturas e indústrias de serviços. Lawler et al. (1993) fazem uma análise da teoria, complexidade e algoritmos para problemas de *scheduling*. Como problemas desse tipo são da classe NP-Difícil, sugere-se o uso de heurísticas para a solução de problemas de *scheduling* em geral.

Neste trabalho, é proposta uma metaheurística que combina GRASP e Busca Tabu para solucionar dois problemas distintos de programação de tarefas. O primeiro problema (RASDST) consiste na programação de tarefas em máquinas paralelas e o segundo (WTSSDS) em uma única máquina. Ambos consideram tempos de preparação entre as tarefas, dependentes da sequência, mas no primeiro o tempo depende também da quantidade de recursos utilizada na preparação. São descritos os detalhes da aplicação do método em cada um desses problemas, e são feitos extensivos testes com instâncias da literatura.

As contribuições do trabalho são: um método facilmente adaptável a vários problemas de programação de tarefas; a efetiva aplicação do método em dois problemas distintos; a publicação de melhores valores de solução para várias instâncias da literatura.

Os detalhes dos problemas são descritos nas próximas seções (2 e 3), juntamente com um levantamento bibliográfico dos trabalhos mais relevantes sobre cada um. A seção 4 descreve o método proposto, e como ele foi aplicado em cada um dos problemas. A seção 5 mostra os resultados computacionais para conjuntos de instâncias da literatura, atestando a eficácia do método, que encontra soluções melhores que os publicados para a maioria das instâncias, em ambos os problemas. A seção 6 conclui o texto, citando também algumas extensões e trabalhos futuros.

## 2. RASDST

O Problema de Programação de Tarefas em Máquinas Paralelas com *Setup time* e Recursos (RASDST - *Resource-assignable Sequence Dependent Setup Time*) foi proposto por Ruiz e Andrés (2007). Nesse problema, deve-se determinar a melhor sequência de $n$ tarefas em $m$ máquinas paralelas distintas, onde cada tarefa deve ser processada em apenas uma máquina. Cada tarefa $j = 1, \ldots, n$ tem um tempo de processamento $p_{ij}$ em cada máquina $i = 1, \ldots, m$. Se a tarefa $k$ é processada depois da tarefa $j$ na máquina $i$ existe um tempo de preparação $S_{ijk}$ dependente da quantidade de recursos $R_{ijk}$. Como no RASDST o tempo de *setup* varia com a quantidade de recursos designados para a preparação, existem valores mínimos e máximos de tempo de *setup* ($S_{ijk}^{-}$; $S_{ijk}^{+}$) e recursos ($R_{ijk}^{-}$; $R_{ijk}^{+}$) que estão relacionados de maneira inversamente proporcional, quanto mais recursos utilizados, menor o tempo de *setup*, e vice-versa.

O objetivo é minimizar, concomitantemente, o tempo total de produção (*flowtime*) e a quantidade de recursos empregados para preparar as máquinas:

$$Z = \alpha \sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} + \beta \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{\substack{k=1 \\ k \neq j}}^{n} R_{ijk}$$

em que $C_{ij}$ é o tempo de conclusão da tarefa $j$ na máquina $i$ ($C_{ij} = 0$ se a tarefa $j$ não é alocada à máquina $i$) e $R_{ijk}$, como mencionado, é a quantidade de recursos alocados para preparar a tarefa $k$ depois da tarefa $j$ na máquina $i$ ($R_{ijk} = 0$ se $k$ não é uma tarefa consecutiva de $j$ em $i$). $\alpha$ e $\beta$ são constantes que indicam a importância ou custo de cada unidade de produção ou recurso, respectivamente.

Definida uma sequência em cada máquina, é possível determinar o número ótimo de recursos atribuídos para a preparação em cada máquina através do algoritmo de Atribuição Otimizada de Recursos definido por Ruiz e Andrés (2007). Dessa forma, a dificuldade do problema consiste em definir qual tarefa vai para cada máquina e em que ordem. Ruiz e Andrés

(2007) provaram que o RASDST é NP-Difícil reduzindo-o a outro problema de sequenciamento NP-Difícil, proposto por Webster (1997).

Existem na literatura vários trabalhos que tratam da programação de tarefas considerando tempos de *setup*, como por exemplo Allahverdi et al. (2008). E também problemas envolvendo máquinas paralelas, por exemplo em Mokotoff (2001). Outros trabalhos, Zhu e Heady (2007), utilizam modelos de Programação Inteira Mista (*Mixed Integer Programming* - MIP) para solucionar alguns problemas de pequeno porte, mas devido a alta complexidade combinatória do problema, heurísticas como *Simulated Annealing*, Busca Tabu, GRASP e Algoritmos Genéticos vem sendo utilizadas.

Foram encontrados somente os seguintes três trabalhos na literatura que abordam a programação de tarefas RASDST. Em Ruiz e Andrés (2007), onde o problema foi proposto, também foi proposto um modelo MIP que soluciona algumas instâncias de pequeno porte, além de três heurísticas construtivas baseadas em regras de despacho. Essas heurísticas são utilizadas em outros trabalhos, por exemplo em Kampke et al. (2009) e Kampke et al. (2010), incorporadas respectivamente nas metaheurísticas GRASP e ILS – *Iterated Local Search*.

## 3. WTSSDS

O segundo problema tratado nesse trabalho, denominado WTSSDS (*Weighted Tardiness Scheduling Problem with Sequence-Dependent Setup Times*), é um problema de programação de tarefas com *setup times* dependente da sequência que visa minimizar o atraso ponderado. Nesse problema, existe um conjunto $J = \{j_1, j_2, \ldots, j_n\}$ de $n$ tarefas para serem processadas em uma única máquina. Cada tarefa possui um tempo de processamento $p_j$, uma data de entrega $d_j$, um peso $w_j$ e um tempo de preparação $s_{ij}$ entre cada par de tarefas distintas $i$ e $j$. Alem disso, cada tarefa $j$ possui um tempo $s_{0j}$ que informa o tempo de preparação da tarefa $j$ caso ela seja a primeira tarefa a ser processada.

Seja $C_j$ o tempo de conclusão da tarefa $j$. Uma tarefa está atrasada se $C_j > d_j$, e seu atraso $T_j = C_j - d_j$. Se a tarefa não estiver atrasada $T_j = 0$. O objetivo é sequenciar as tarefas na máquina de forma a minimizar o atraso ponderado, que é dado por:

$$T = \sum_{j \in J} w_j T_j = \sum_{j \in J} w_j \max(C_j - d_j, 0)$$

Como o objetivo é calcular o atraso total ponderado, definida uma ordem das tarefas, o valor ótimo para essa sequência pode ser determinado de maneira simples processando todas as tarefas sequencialmente, sem tempo de espera (exceto o *setup time*). Dessa forma, a dificuldade do problema se restringe em determinar a melhor ordem das tarefas. Entretanto, o problema é NP-Difícil, pois é uma generalização de uma versão sem tempos de *setup* que é provado ser NP-Difícil em Lenstra e Kan (1980).

Algumas heurísticas foram propostas na literatura para solucionar o WTSSDS. Cicirello (2006) propôs um Algoritmo e em Cicirello (2007) foi proposto uma versão do algoritmo *Simulated Annealing* que melhora os resultados de algumas instâncias. Cicirello e Smith (2005) investigaram o uso de randomização para melhorar o desempenho das heurísticas com um algoritmo de amostragem estocástica, denominado *Value-Biased Stochastic Sampling* (VBSS), obtendo novos resultados para o problema. Cicirello propôs outros algoritmos, como *Limited Discrepancy Search* em Cicirello (2003). Lee et al. (1997) propuseram uma regra de despacho para ordenar as tarefas, e usaram o algoritmo em um problema real. Essa regra de despacho é citada e usada nos trabalhos de Cicirello. Ying et al. (2009) utilizaram *Iterated Greedy Heuristic* para solucionar essa e outras versões multi-objetivo desse problema.

## 4. Metaheurística Proposta

A metaheurística proposta neste trabalho, denominada GBT, consiste na combinação de duas heurísticas conhecidas na literatura: GRASP e Busca Tabu (*Tabu Search* - TS). Soluções iniciais são geradas pelo GRASP e, em seguida, a melhor solução é submetida à heurística de Busca Tabu para ser melhorada (Figura 1). O mesmo método é aplicado a dois problemas distintos, existindo algumas diferenças para se adaptar ao RASDST e ao WTSSDS, principalmente na fase construtiva. Essas diferenças serão detalhadas adiante.

### 4.1. Representação da solução

A representação da solução no problema WTSSDS consiste, simplesmente, em um vetor de $n$ posições onde a posição 0 indica a primeira tarefa a ser programada, a posição 1 a segunda tarefa, e assim por diante. Já no RASDST, a representação se dá através de um vetor com ($n$+$m$-1) posições. Existem $m$-1 elementos (-1) que dividem o vetor em $m$ partes, uma para cada máquina. Por exemplo, a sequência $s$ = {2, -1, 4, 1, 3} indica que a máquina 1 processa somente a tarefa 2 e a máquina 2 processa a tarefa 4, 1 e 3, necessariamente nessa ordem.

### 4.2. GRASP

A metaheurísica GRASP (*Greedy Randomized Adaptive Search Procedure*) foi proposta por Feo e Resende (1989) e é composta de duas fases: a fase de Construção, que gera uma solução inicial e a fase de Busca Local (BL), que visa melhorar a solução obtida na fase anterior. O GRASP possui várias iterações e a cada iteração uma nova solução é gerada pelo procedimento de Construção, que depois é submetida à BL. Essa metaheurística recebe como entrada o parâmetro $\lambda$, usado na fase de construção, e o critério de parada, que, neste caso, é o número de iterações.

A seguir serão detalhadas as fases de Construção e Busca Local do GRASP. Além da representação da solução, a fase de Construção do GRASP é diferente nos dois problemas. Logo, são detalhadas separadamente.

#### 4.2.1. Construção

A fase de Construção do GRASP para a geração de uma solução factível é feita iterativamente: inicialmente há uma sequência vazia e a cada iteração uma tarefa é adicionada à sequência. Para definir a próxima tarefa a ser acrescentada, define-se uma Lista de Candidatos (LC) com todas as tarefas ainda não sequenciadas. A Lista de Candidatos é ordenada considerando algum critério, e, após a ordenação, uma tarefa é selecionada aleatoriamente da Lista Restrita de Candidatos (LRC) composta dos $\theta = MAX(1, \lambda \times |LC|)$ primeiros elementos da LC, sendo $\lambda$ o parâmetro de aleatoriedade. Um pseudocódigo genérico é mostrado na Figura 2.

```
Procedimento GBT (time_gbt)
inicio
    s ← GRASP()
    time_ts = time_gbt – Tempo atual
    s* ← Busca_Tabu (s, time_ts)

    retorne s*
fim
```

**Figura 1: Pseudocódigo do algoritmo GBT**

```
Procedimento Construção (λ)
início
    LC ← {j1, j2, ..., jn}
    Sequencia ← Ø
    enquanto |LC| ≠ 0 faça
        ordene (LC)
        θ ← MAX (1, λ x |LC|)
        pos ← Random(1, θ)
        Sequencia ← Sequencia ∪ {jpos}
        LC ← LC – {jpos}
    fim-enquanto

    retorne Sequencia
fim
```

**Figura 2: Pseudocódigo da fase de Construção do GRASP**

#### 4.2.1.1. Fase de Construção no RASDST

Para determinar o valor do acréscimo de cada tarefa na fase de Construção para o RASDST, foi utilizada a heurística construtiva DJASA (*Dynamic Job Assignment with Setups Resource Assignment*), proposta por Ruiz e Andrés (2007), onde a alocação de tarefas é feita dinamicamente, sendo então apropriada para ser usada na fase de Construção. A cada iteração, várias simulações são feitas considerando a inclusão de uma tarefa não sequenciada em todas as máquinas. A simulação que resultar no menor incremento no valor da função objetivo é incluída na sequência. O que difere a fase Construtiva do GRASP do método DJASA é que este inclui sempre a melhor simulação aparente, enquanto a Construção do GRASP permite que inclusões aparentemente não tão boas sejam feitas, podendo resultar em melhores inserções futuras.

#### 4.2.1.2. Fase de Construção no WTSSDS

Podemos utilizar a mesma idéia da fase de Construção do RASDST para o problema WTSSDS, mudando somente a função para avaliar as sequências parciais. Porém, encontramos melhores resultados utilizando uma heurística gulosa (o que equivale ao GRASP quando $\lambda = 0$). A heurística gulosa utilizada é a regra de despacho ATCS (*Apparent Tardiness Cost with Setups*) proposta por Lee et al. (1997), a mesma utilizada por Cicirello (2003) em seus algoritmos. A regra de despacho ATCS é definida como:

$$ATCS_j(t,l) = \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - p_j - t, 0)}{k_1 \bar{p}} - \frac{s_{lj}}{k_2 \bar{s}}\right)$$

onde $t$ é o tempo corrente, $l$ é o índice da última tarefa seqüenciada, $w_j$, $p_j$ e $d_j$ são, respectivamente, o peso, tempo de processamento e a data de entrega da tarefa $j$ que está sendo avaliada, $s_{lj}$ é o *setup time* decorrente do processamento da tarefa $j$ após a tarefa $l$; $\bar{p}$ e $\bar{s}$ são a média do tempo de processamento e dos *setup times*, respectivamente. Os valores $k_1$ e $k_2$ são parâmetros de ajuste, cujos valores são calculados como definido em Cicirello (2007). A próxima tarefa $j$ que deve ser adicionada à sequência é aquela com o maior ATCS.

### 4.2.2. Busca Local

O procedimento de Busca Local consiste em explorar a vizinhança de uma solução inicial à procura de um mínimo local. Soluções vizinhas são geradas a partir de movimentos em uma solução inicial. O movimento considerado nessa busca local consiste na permutação de duas tarefas consecutivas numa mesma máquina (no caso do RASDST não é permitido a permutação de tarefas pertencentes a máquinas distintas). A idéia é trabalhar não somente com uma, mas com *size_list* melhores soluções da vizinhança. Esse método foi proposto por Santos et al. (2010) para outro problema de programação de tarefas. O procedimento começa adicionando a solução proveniente da fase de Construção em uma lista de soluções (de tamanho *size_list*). Ela é então marcada e sua vizinha é explorada. Uma solução vizinha é adicionada à lista se esta não estiver cheia ou se houver uma solução na lista com o valor da função objetivo maior do que o encontrado. Para cada solução não marcada na lista, esta é então marcada e sua vizinhança enumerada, analogamente ao processo da solução inicial. O algoritmo termina quando todas as soluções da lista estiverem marcadas. O pseudocódigo da BL utilizado nesse trabalho é mostrado na Figura 3.

## 4.1. Busca Tabu

O algoritmo de Busca Tabu (BT), proposto em Glover (1996), é um algoritmo que visa explorar o espaço de soluções além de ótimos locais. Para isso, usa-se uma estrutura de memória para guardar a(s) característica(s) de uma solução vizinha, impedindo que o algoritmo retorne a esta solução por um período de tempo. Neste trabalho, é utilizado o movimento de inserção de uma tarefa em todas as posições da sequência para explorarmos a vizinhança de uma solução.

A BT requer a definição de alguns componentes, como o movimento que gera uma nova solução, o atributo (característica) que deve ser armazenado de cada solução, a regra de

proibição, a duração tabu (número de iterações), o critério de aspiração e o critério de parada. O movimento que gera uma nova solução é o movimento de inserção, como mencionado anteriormente. A próxima solução escolhida é a melhor solução encontrada na vizinhança. No caso do RASDST, serão guardados como atributos que caracterizam a solução a tarefa *j* selecionada para inserção e a máquina *i* na qual a tarefa foi inserida. Para o WTSSDS, foi guardado somente a tarefa selecionada para inserção, já que o problema envolve apenas uma máquina. A regra de proibição é a não inserção dessa tarefa em algumas iterações posteriores. A duração tabu é definida da seguinte forma: a cada 20 iterações um novo valor é gerado considerando uma distribuição uniforme a partir de um intervalo dependente no número de tarefas *n*. Esses intervalos foram tomados com base em alguns trabalhos da literatura.

```
Procedimento Busca_Local (s)
início
    Lista ← Lista ∪ {s}
    enquanto Existe solução não expandida na Lista faça
        SolucaoCorrente ← primeira solução não expandida da Lista
        Marque SolucaoCorrente como expandida
            para i = 0 até i < n+m-1 faça
                se SolucaoCorrente[i] ≠ -1 and SolucaoCorrente [i+1] ≠ -1 então
                    Permute as entradas i and i+1 de SolucaoCorrente
                    se SolucaoCorrente <  Pior solução da Lista então
                        Lista ← Lista –  Pior solução da Lista
                                Lista ← Lista ∪ { SolucaoCorrente }
                        ordene (Lista)
                    fim-se
                    Permute as entradas i and i+1 de SolucaoCorrente
                        fim-se
            fim-para
    fim-enquanto

    s* ← melhor solução da Lista

    retorne s*
fim
```

**Figura 3: Pseudocódigo do algoritmo de Busca Local**

O critério de aspiração permite que, em certos momentos, um movimento tabu seja executado. Essa função permite a escolha de uma solução vizinha se o valor dessa solução for menor que o melhor valor encontrado até o momento, mesmo que o movimento que gerou esse vizinho seja tabu. A Figura 4 mostra o pseudocódigo do método de BT. O critério de parada é o tempo de CPU. Foi definido um tempo limite para o algoritmo GBT que varia de acordo com o número de tarefas *n* e o número de máquinas *m* (*time_gbt* = *nm*/2), o mesmo utilizado em outros trabalhos, como por exemplo, Kampke et al. (2009) e Kampke et al. (2010). Assim, o tempo limite para a Busca Tabu, denominado *time_ts*, depende do tempo gasto pelo GRASP (*time_ts* = *time_gbt – time_grasp*).

Nesse trabalho a Busca Tabu recebe como entrada a melhor solução retornada pelo GRASP e o tempo permitido (*time_ts*). O método realiza a Busca Tabu nessa solução e retorna a melhor solução encontrada no processo.

```
Procedimento Busca_Tabu (s, time_ts)
início
    TabuList[ ] ← 0
    s* ← s                    // Melhor solução
    b* ← f(s)                 // Melhor valor da função objetivo, calculado pela função f()
    k ← 0

    enquanto tempo de CPU < time_ts faça
        s' ← melhor vizinho de s em que o movimento m (que gerou este vizinho) não é
             tabu (TabuList[m] ≤ k) ou, se tabu, satisfaça o critério de aspiração (f(s') < f(s*))
        s ← s'
        TabuList[m] ← k + duração tabu
        se f(s) < b então
            s* ← s
            b ← f(s)
        fim-se
        k ← k + 1
        Atualize a duração tabu, se for o caso
    fim-enquanto

    retorne s*
fim
```

**Figura 4: Pseudocódigo do procedimento de BT (O tamanho de TabuList varia dependendo do problema. No RASDST, TabuList é uma matriz e |TabuList| = mn. No WTSSDS, |TabuList| = n)**

## 5. Resultados

O algoritmo GBT foi implementado na linguagem de programação C++. Os experimentos foram testados em uma máquina Intel® Core™2 Quad 2.4GHz com 2GB de memória RAM. Para o RASDST utilizamos as instâncias geradas por Ruiz e Andrés (2007), disponíveis em http://soa.iti.es/problem-instances. As instâncias estão divididas em dois grupos: *small* e *large*. O grupo *small* contém todas as combinações de $n$ = {6, 8, 10} tarefas e $m$ = {3, 4, 5} máquinas. O grupo *large* é composto pelas combinações de $n$ = {50, 75, 100} tarefas e $m$ = {10, 15, 20} máquinas. Juntando a combinação desses e de outros parâmetros envolvendo *setup times* e recursos são disponíveis 720 casos de teste, 360 para cada grupo. Para o cálculo da função objetivo, é utilizado os mesmos valores usados pelos outros autores: $\alpha$ =1 e $\beta$ = 50.

Para as instâncias do grupo *small* soluções ótimas foram encontradas para as instâncias com 6 e 8 tarefas, usando, respectivamente, 5 e 60 minutos como critério de parada na solução de um modelo MIP. Para este conjunto, o GBT alcançou a melhor solução disponível na literatura em 328 das 360 instâncias. Para as 32 instâncias restantes a diferença média foi muito baixa, menor que 0,2%. Já as instâncias do grupo *large* o GBT superou os outros métodos da literatura em 251 das 360 instâncias, o que representa 69,7% do conjunto, conforme detalhado a seguir.

Para avaliar os resultados, utilizamos, para cada instância, o Desvio Relativo Percentual (DRP), que é calculado pela expressão:

$$DRP = \frac{(Método_{sol} - Melhor_{sol})}{Melhor_{sol}} \times 100$$

onde $Método_{sol}$ é o valor da função objetivo (F.O.) da solução obtida por um determinado algoritmo e $Melhor_{sol}$ é o valor da F.O. da melhor solução já encontrada. Os valores de DRP para o grupo *large* são mostrados na Tabela 1. A primeira coluna indica o nome da instância. Cada linha da Tabela 1 mostra a média dos DRP's das 10 instâncias do tipo I_*n*_*m*_S($S_{ijk}^-$; $S_{ijk}^+$)_R($R_{ijk}^-$; $R_{ijk}^+$). As próximas colunas indicam a média dos DRP's para cada método: GBT é o algoritmo Grasp-Busca Tabu proposto neste trabalho, GR é o GRASP reativo com *Path Relinking* proposto por Kampke et al. (2009), ILS é o algoritmo *Iterated Local Search*

com *Path Relinking* proposto por Kampke et al. (2010), e a coluna GR+ILS representa o melhor resultado desses dois últimos métodos, que são os melhores encontrados na literatura até o momento. A última coluna indica o número de instâncias melhoradas pelo GBT, considerando as 10 instâncias do mesmo tipo.

**Tabela 1: Valore dos DRP's para o GBT e outros métodos da literatura para o RASDST**

| Instância | Média dos DRP's | | | | Melhor (de 10) |
|---|---|---|---|---|---|
| | GBT | GR | ILS | GR+ILS | |
| I_50_10_S(1-50;50-100)_R(1-3;3-5) | 0,049 | 0,914 | 1,226 | 0,646 | 8 |
| I_50_10_S(1-50;50-100)_R(1-5;5-10) | 0,240 | 0,935 | 0,600 | 0,390 | 5 |
| I_50_10_S(50-100;100-150)_R(1-3;3-5) | 0,000 | 0,994 | 0,691 | 0,548 | 10 |
| I_50_10_S(50-100;100-150)_R(1-5;5-10) | 0,001 | 1,386 | 1,150 | 1,056 | 9 |
| I_50_15_S(1-50;50-100)_R(1-3;3-5) | 0,384 | 0,819 | 0,337 | 0,268 | 5 |
| I_50_15_S(1-50;50-100)_R(1-5;5-10) | 0,255 | 1,896 | 0,983 | 0,948 | 6 |
| I_50_15_S(50-100;100-150)_R(1-3;3-5) | 0,117 | 0,963 | 0,390 | 0,273 | 7 |
| I_50_15_S(50-100;100-150)_R(1-5;5-10) | 0,008 | 1,827 | 1,086 | 1,017 | 9 |
| I_50_20_S(1-50;50-100)_R(1-3;3-5) | 0,111 | 1,566 | 0,604 | 0,421 | 7 |
| I_50_20_S(1-50;50-100)_R(1-5;5-10) | 0,666 | 1,571 | 1,000 | 0,480 | 5 |
| I_50_20_S(50-100;100-150)_R(1-3;3-5) | 0,450 | 0,795 | 0,141 | 0,137 | 4 |
| I_50_20_S(50-100;100-150)_R(1-5;5-10) | 0,615 | 1,449 | 0,883 | 0,672 | 5 |
| I_75_10_S(1-50;50-100)_R(1-3;3-5) | 0,455 | 0,980 | 0,705 | 0,347 | 4 |
| I_75_10_S(1-50;50-100)_R(1-5;5-10) | 0,502 | 1,775 | 0,300 | 0,284 | 5 |
| I_75_10_S(50-100;100-150)_R(1-3;3-5) | 0,035 | 1,087 | 0,482 | 0,482 | 9 |
| I_75_10_S(50-100;100-150)_R(1-5;5-10) | 0,020 | 1,610 | 0,756 | 0,699 | 8 |
| I_75_15_S(1-50;50-100)_R(1-3;3-5) | 0,315 | 1,324 | 1,266 | 0,595 | 7 |
| I_75_15_S(1-50;50-100)_R(1-5;5-10) | 0,290 | 1,824 | 1,249 | 1,097 | 8 |
| I_75_15_S(50-100;100-150)_R(1-3;3-5) | 0,045 | 1,301 | 0,693 | 0,693 | 9 |
| I_75_15_S(50-100;100-150)_R(1-5;5-10) | 0,048 | 1,864 | 1,031 | 0,995 | 9 |
| I_75_20_S(1-50;50-100)_R(1-3;3-5) | 0,477 | 1,487 | 0,288 | 0,258 | 3 |
| I_75_20_S(1-50;50-100)_R(1-5;5-10) | 0,441 | 3,031 | 1,414 | 1,302 | 7 |
| I_75_20_S(50-100;100-150)_R(1-3;3-5) | 0,018 | 1,805 | 0,854 | 0,854 | 9 |
| I_75_20_S(50-100;100-150)_R(1-5;5-10) | 0,000 | 2,082 | 1,324 | 1,203 | 10 |
| I_100_10_S(1-50;50-100)_R(1-3;3-5) | 0,699 | 1,927 | 0,072 | 0,072 | 3 |
| I_100_10_S(1-50;50-100)_R(1-5;5-10) | 0,494 | 1,550 | 0,743 | 0,582 | 6 |
| I_100_10_S(50-100;100-150)_R(1-3;3-5) | 0,002 | 0,974 | 0,395 | 0,355 | 8 |
| I_100_10_S(50-100;100-150)_R(1-5;5-10) | 0,270 | 1,303 | 0,386 | 0,346 | 5 |
| I_100_15_S(1-50;50-100)_R(1-3;3-5) | 0,629 | 1,884 | 0,817 | 0,606 | 7 |
| I_100_15_S(1-50;50-100)_R(1-5;5-10) | 0,356 | 2,409 | 0,748 | 0,748 | 5 |
| I_100_15_S(50-100;100-150)_R(1-3;3-5) | 0,171 | 1,705 | 0,882 | 0,858 | 8 |
| I_100_15_S(50-100;100-150)_R(1-5;5-10) | 0,000 | 2,050 | 1,320 | 1,290 | 10 |
| I_100_20_S(1-50;50-100)_R(1-3;3-5) | 0,091 | 1,715 | 0,813 | 0,768 | 9 |
| I_100_20_S(1-50;50-100)_R(1-5;5-10) | 0,515 | 2,138 | 0,820 | 0,809 | 5 |
| I_100_20_S(50-100;100-150)_R(1-3;3-5) | 0,107 | 1,564 | 0,679 | 0,617 | 7 |
| I_100_20_S(50-100;100-150)_R(1-5;5-10) | 0,000 | 2,126 | 1,144 | 1,064 | 10 |
| **Média** | **0,246** | **1,573** | **0,785** | **0,661** | **6,97** |

Pela Tabela 1, última coluna, observa-se que para o problema RASDST, o algoritmo GBT encontrou soluções melhores em 251 das 360 instâncias do grupo *large* (cerca de 69,7% do conjunto) com relação aos outros métodos da literatura. É importante ressaltar que, em alguns casos, o GBT não encontra uma melhor solução do que o GR+ILS, mas melhora o resultado quando comparado com cada método separadamente. Para visualizar melhor a superioridade do GBT em relação aos outros métodos, aplicamos uma análise de variância (ANOVA) sobre os DRP's de cada solução do grupo *large* encontrada por cada algoritmo. A comparação das variações dos DRP's é mostrada na Figura 5.

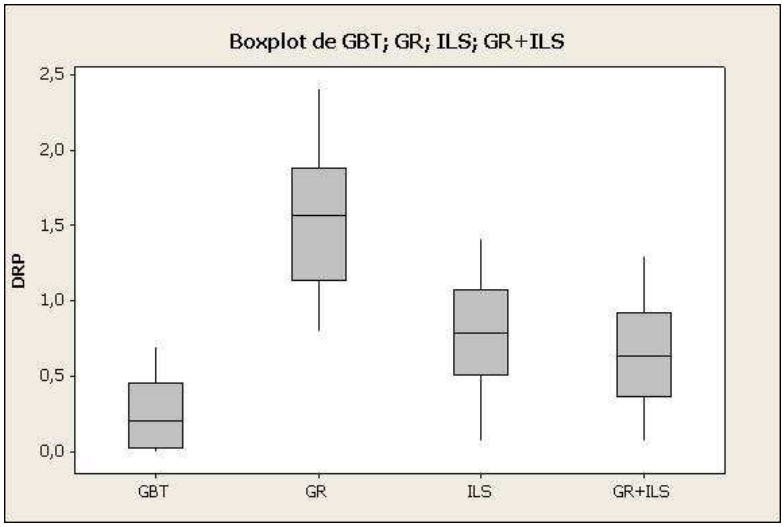


**Figura 5: Comparação entre os métodos utilizando um gráfico boxplot para o RASDST**

Observa-se pela Figura 5 que a mediana dos DRP's das soluções encontradas pelo GBT é significativamente menor do que a mediana dos DRP's dos valores encontrados pelos algoritmos propostos Kampke et al. (2009) e Kampke et al. (2010), o que indica que os valores obtidos por estes métodos tendem a ser maiores do que os encontrados pelo algoritmo proposto neste trabalho. Além disso, é verificado através do Teste de Tukey (nível 0,05) que a diferença entre os resultados encontrados pelo GBT e os resultados encontrados pelos outros métodos é estatisticamente significativa. Portanto, pode-se afirmar que o método GBT, na média, é melhor que os outros métodos.

Com relação ao problema WTSSDS, utilizamos um conjunto de 120 instâncias proposto por Cicirello (2003), que está disponível *online*. Ele também mantém uma lista (disponível em http://loki.stockton.edu/_cicirelv/benchmarks/bestknown.txt) com o valor da melhor solução já encontrada para cada instância. Nós não encontramos uma análise do tempo de execução em algoritmos da literatura. Eles usaram outros critérios de parada, como, por exemplo, o número de gerações em Algoritmos Genéticos, o que não se aplica diretamente ao nosso método. Então utilizamos o limite de tempo correspondente a $nm/2$, o mesmo usado no problema RASDST (como, para todas as instâncias, temos $n = 60$ e $m = 1$, o tempo limite é de 30 segundos). Assim, a BT é executada durante $30 - t_g - t_{bl}$ , sendo $t_g$ o tempo de execução do método guloso e $t_{bl}$ o tempo de execução da BL. O tempo utilizado pela BT é de fato dividido por 4, e 4 execuções em paralelo são feitas, uma em cada núcleo de processamento do Intel Quad. Logo, o tempo total do algoritmo permanece 30 segundos. Para cada instância, o método foi executado 10 vezes, e o melhor resultado foi utilizado, o mesmo feito em Ying et al. (2009).

A Tabela 2 mostra o DRP dos resultados encontrados pelo GBT (representados pela coluna "DRP-GBT") para o WTSSDS. A coluna ID é o número de identificação da instância. As colunas "DRP-A" e "Algoritmo" mostram o DRP do melhor valor da função objetivo conhecido na literatura e o algoritmo que o encontrou primeiro, respectivamente. Dezoito instâncias foram omitidas, já que o GBT encontrou os mesmos resultados já publicados na literatura (*DRP-GBT* = *DRP-A* = 0). Dessas 18 instâncias, 17 são soluções ótimas para o problema.

**Tabela 2: Valores dos DRP's para o GBT e os outros métodos da literatura problema WTSSDS**

| ID | Algoritmo | DRP-A | DRP-GBT | ID | Algoritmo | DRP-A | DRP-GBT |
|---|---|---|---|---|---|---|---|
| 1 | SA-Heur | 14,9927 | 0 | 69 | SA-Heur | 0,5849 | 0 |
| 2 | SA-Heur | 8,2126 | 0 | 70 | SA-R | 0,0749 | 0 |
| 3 | GA | 2,8693 | 0 | 71 | SA-R | 1,6956 | 0 |
| 4 | SA-Heur | 0,4405 | 0 | 72 | SA-Heur | 6,4325 | 0 |
| 5 | SA-R | 6,2924 | 0 | 73 | SA-Heur | 0 | 8,8436 |
| 6 | SA-Heur | 5,2745 | 0 | 74 | SA-R | 2,8928 | 0 |
| 7 | SA-Heur | 2,4752 | 0 | 75 | SA-R | 0 | 16,2716 |
| 8 | SA-R | 93,5065 | 0 | 76 | SA-Heur | 2,3566 | 0 |
| 9 | SA-Heur | 8,2669 | 0 | 77 | SA-Heur | 0 | 0,0254 |
| 10 | SA-Heur | 0 | 0,5647 | 78 | SA-Heur | 0 | 0,0280 |
| 11 | SA-Heur | 4,7992 | 0 | 79 | SA-Heur | 3,9603 | 0 |
| 13 | VBSS-HC,2500,5 | 6,5338 | 0 | 80 | SA-Heur | 9,5395 | 0 |
| 14 | SA-R | 8,6680 | 0 | 81 | SA-Heur | 0 | 0,6380 |
| 15 | SA-R | 0,9906 | 0 | 82 | SA-Heur | 0 | 0,4784 |
| 16 | SA-R | 7,4826 | 0 | 83 | SA-R | 0 | 2,0396 |
| 17 | SA-R | 78,3410 | 0 | 84 | SA-R | 0 | 0,5349 |
| 18 | SA-R | 10,1033 | 0 | 85 | SA-R | 0 | 0,1185 |
| 19 | SA-Heur | 35,7955 | 0 | 86 | SA-R | 0,2324 | 0 |
| 20 | SA-R | 6,2552 | 0 | 87 | SA-R | 0 | 0,1167 |
| 24 | SA-Heur | 2,1515 | 0 | 88 | SA-R | 0 | 0,4040 |
| 27 | SA-R | 100 | 0 | 89 | SA-Heur | 0 | 0,8477 |
| 30 | SA-R | 0 | 24,1860 | 90 | SA-R | 0 | 0,7457 |
| 37 | SA-R | 0,9009 | 0 | 91 | SA-R | 0 | 0,3118 |
| 41 | SA-Heur | 0,1603 | 0 | 92 | SA-R | 0 | 0,4852 |
| 42 | SA-Heur | 1,3665 | 0 | 93 | VBSS,100,19 | 0 | 2,0077 |
| 43 | SA-Heur | 0 | 0,0379 | 94 | SA-Heur | 0 | 3,5224 |
| 44 | SA-Heur | 0,5267 | 0 | 95 | SA-R | 0 | 1,4914 |
| 45 | SA-R | 0,4493 | 0 | 96 | LDS-all-two | 0 | 0,1113 |
| 46 | SA-R | 0,6315 | 0 | 97 | SA-Heur | 0,0767 | 0 |
| 47 | SA-Heur | 0 | 0,5444 | 98 | VBSS,100,12 | 0 | 0,6850 |
| 48 | SA-R | 0 | 1,1070 | 99 | SA-R | 0,3246 | 0 |
| 49 | SA-R | 0,9940 | 0 | 100 | VBSS-HC,500,5 | 0,2907 | 0 |
| 50 | SA-R | 0 | 0,1464 | 101 | SA-R | 0 | 0,2596 |
| 51 | SA-Heur | 2,1434 | 0 | 102 | SA-Heur | 0 | 0,6094 |
| 52 | SA-Heur | 0 | 0,8086 | 103 | VBSS,100,18 | 0,3447 | 0 |
| 53 | SA-R | 4,3500 | 0 | 104 | SA-R | 0 | 0,9135 |
| 54 | VBSS,2500,5 | 0 | 1,9254 | 106 | SA-Heur | 0 | 0,1883 |
| 55 | SA-R | 6,1146 | 0 | 107 | SA-Heur | 0 | 0,8406 |
| 56 | SA-Heur | 0,0299 | 0 | 108 | SA-R | 0 | 0,3199 |
| 57 | SA-Heur | 0 | 1,6254 | 109 | SA-Heur | 0,0667 | 0 |
| 58 | SA-R | 0 | 4,1275 | 110 | SA-R | 0 | 0,3912 |
| 59 | SA-Heur | 0,9542 | 0 | 111 | SA-Heur | 1,4803 | 0 |
| 60 | SA-R | 1,2374 | 0 | 112 | SA-Heur | 0 | 0,9405 |
| 61 | SA-R | 0 | 0,1576 | 113 | SA-Heur | 0,0260 | 0 |
| 62 | SA-Heur | 0 | 0,4913 | 114 | SA-Heur | 0 | 0,2064 |
| 63 | SA-Heur | 0,1172 | 0 | 115 | VBSS,100,10 | 0 | 2,1659 |
| 64 | SA-Heur | 0 | 2,2078 | 116 | SA-Heur | 0 | 1,0409 |
| 65 | SA-R | 0 | 2,1016 | 117 | SA-Heur | 0 | 0,3874 |
| 66 | SA-Heur | 0 | 1,0868 | 118 | LDS-all-two | 0 | 0,6866 |
| 67 | SA-R | 0 | 1,2213 | 119 | SA-Heur | 0 | 0,2359 |
| 68 | SA-R | 0,0177 | 0 | 120 | VBSS-HC,10000,5 | 0 | 0,2997 |

Observa-se, pela Tabela 2, que o GBT melhorou 51 das 120 instâncias testadas (DRP-GBT = 0). Contando as 17 soluções ótimas alcançadas, o GBT encontrou valores melhores ou iguais aos da literatura em quase 60% das instâncias, incluindo um novo valor ótimo encontrado (instância 27), cujo DRP-GBT = 0 e DRP-A = 100. Além disso, para quase todas as instâncias em que o método GBT não conseguiu melhorar os resultados da literatura o *DRP* é pequeno. Na Figura 6 mostra-se o bloxpot da variação dos DRP's do GBT e dos algoritmos da literatura para os resultados do WTSSDS.

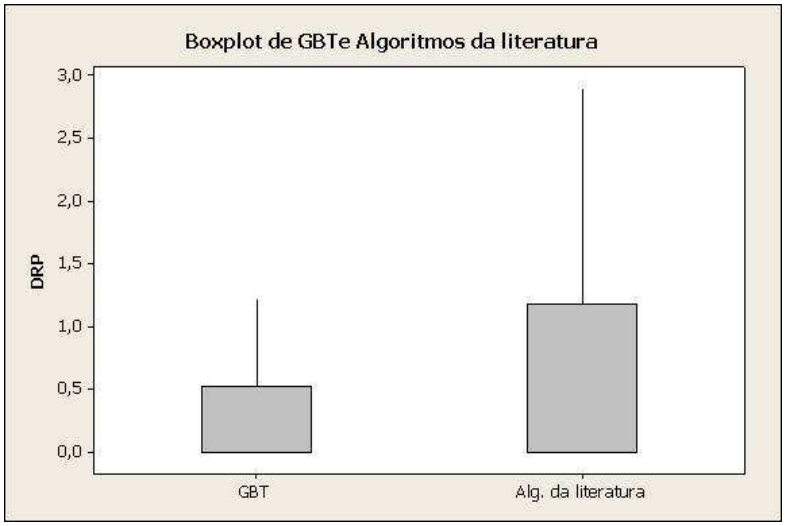


**Figura 6: Comparação entre os *DRP's* das soluções do GBT da literatura para o WTSSDS**

Através da Figura 6 observa-se que o GBT possui uma menor variação nas soluções obtidas se comparado aos algoritmos propostos na literatura para o WTSSDS.

## 6. Conclusão

Nesse trabalho nós apresentamos uma metaheurística, chamada GBT, baseada em GRASP e Busca Tabu para solucionar dois problemas de programação de tarefas, o RASDST e o WTSSDS. Em ambos os problemas, o algoritmo proposto conseguiu melhorar a maioria dos resultados publicados na literatura.

Para o RASDST, nós aplicamos o GBT em 720 casos de teste de portes variados. Para o grupo *small* (instâncias de pequeno porte), o GBT conseguiu alcançar cerca de 92% das soluções já publicadas. Já para o grupo *large* (instâncias de grande porte) o GBT conseguiu melhorar a solução de 69,7% das instâncias.

No caso do WTSSDS, o método foi aplicado a um conjunto padrão de instâncias da literatura, usado por diversos outros autores. O GBT encontrou soluções iguais ou melhores às já publicadas em 60% dos casos de teste. Nas demais instâncias, que não tiveram o resultado melhorado, a diferença entre a qualidade dos resultados já publicados e do GBT é geralmente muito pequena.

O algoritmo é geral o suficiente para ser empregado em vários outros problemas de sequenciamento, e também pode ser facilmente paralelizado. Resultados preliminares mostram que essas atividades são válidas e em breve teremos resultados comparativos.

## Agradecimentos

Os autores agradecem ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e à Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) pelo financiamento desta pesquisa.

## Referências

**Allahverdi, A. Ng, C.T., Cheng, T.C.E. e Kovalyov, M.Y.** (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3): 985-1032.

**Cicirello, V. A.** (2003). Boosting stochastic problem solvers through online self-analysis of performance. Technical Report CMU-RI-TR-03-27, The Robotics Institute, Carnegie Mellon University.

**Cicirello, V. A.** (2006). Non-wrapping order crossover: an order preserving crossover operator that respects absolute position. In *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO'06), volume 2: 1125–1131.

**Cicirello, V. A.** (2007). On the design of an adaptive simulated annealing algorithm. In *Proceedings of the First International Workshop on Autonomous Search* (CP 2007).

**Cicirello, V. A. e Smith, S. F.** (2005). Enhancing stochastic search performance by value-biased randomization of heuristics. *Journal of Heuristics*, 11(1):5–34.

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

**Glover, F.** (1996). Tabu search and adaptive memory programming - advances, applications and challenges. In: Barr, R.S., Helgason, R.V. and Kenngington, J.L. editors. *Interfaces in Computer Science and Operations Research*, 1-75.

**Kampke, E.H., Arroyo, J.E.C., e Santos, A.G.** (2009).Reactive GRASP with path relinking for solving parallel machines scheduling problem with resource-assignable sequence dependent setup times. In *Proceedings of the 8th International Conference on Computer Information Systems and Industrial Management Applications* (CISIM '09) Coimbatore, India, 924-929.

**Kampke, E.H., Arroyo, J.E.C., e Santos, A.G.** (2010). Iterated local search with path relinking for solving parallel machines scheduling problem with resource-assignable sequence dependent setup times. In *Proceedings of the 10th European Conference on Evolutionary Computation in Combinatorial Optimisation* (EvoCOP '10), Istanbul, Turkey, 107-118.

**Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G. e Shmoys D.B** (1993). Sequencing and Scheduling: Algorithms and Complexity, chapter 9, em *Logistics of Production and Inventory*, Graves S. C., Zipking P.H. (editores), North Holland.

**Lee, Y. H., Bhaskaran K., e Pinedo M.** (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions*, 29:45–52.

**Lenstra, J. K. e Kan, A. H. G.** (1980). Complexity results for scheduling chains on a single machine. *European Journal of Operational Research*, 4:270–275.

**Mokotoff, E.** (2001). Parallel machine scheduling problems: a survey. *Asia-Pacific Journal of Operational Research*, 18(2):193-242.

**Pinedo, M.** (2008). Scheduling: theory, algorithms, and systems, 3ª ed. Springer Verlag.

**Ruiz, R. e Andrés, C.** (2007). Unrelated parallel machines scheduling with resource-assignable sequence dependent setup times. In *Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications* (MISTA) Paris, France, 439-446.

**Santos, A.G., Araujo, R.P., e Arroyo, J.E.C.** (2010). A combination of evolutionary algorithm, mathematical programming, and a new local search procedure for the just-in-time job-shop scheduling problem. In *Proceedings of the 4th international conference on Learning and intelligent optimization (LION'10)*, 10-24.

**Webster, S.T.** (1997). The complexity of scheduling job families about a common due date. *Operations Research Letters*, 20(2): 65-74.

**Ying , K. C., Lin, S. W., e Huang , C. Y.** (2009). Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic. *Expert Systems with Applications*, 36:7087–7092.

**Zhu, Z. e Heady, R.** (2000). Minimizing the sum of earliness/tardiness in multi-machine scheduling: a mixed integer programming approach, *Computers & Industrial Engineering*, 38(2): 297-305.