UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Metaheurística Híbrida GRASP e Busca Tabu Aplicada ao Problema de Escalonamento de Tarefas

Dissertação de Mestrado, submetida ao Curso de Mestrado em Informática — como parte dos requisitos para a obtenção do título de Mestre.

Mestranda: Cláudia Rossana Cunha

Orientador: Prof. Dr. Lucídio dos Anjos F. Cabral

João Pessoa – PB
Julho de 2010.

CLÁUDIA ROSSANA CUNHA

Metaheurística Híbrida GRASP e Busca Tabu Aplicada ao Problema de Escalonamento de Tarefas

Dissertação de Mestrado, submetida ao Curso de Mestrado em Informática — como parte dos requisitos para a obtenção do título de Mestre.

Orientador: Prof. Dr. Lucídio dos Anjos F. Cabral

João Pessoa – PB
Julho de 2010.

C972m Cunha, Cláudia Rossana.
Metaheurística híbrida GRASP e Busca Tabu aplicada ao problema de escalonamento de tarefas / Cláudia Rossana Cunha.- João Pessoa, 2010.
66f. : il.
Orientador: Lucídio dos Anjos F. Cabral
Dissertação (Mestrado) – UFPB/CCEN
1. Informática. 2. Escalonamento de tarefas. 3. GRASP. 4. Algoritmo de Coffman Gramah. 5. Busca Tabu.

UFPB/BC CDU: 004(043)

ATA DA SESSÃO PÚBLICA DE DEFESA DE DISSERTAÇÃO

Ata da Sessão Pública de Defesa de Dissertação de Mestrado da CLÁUDIA ROSSANA CUNHA, candidata ao Título de Mestre em Informática na Área de Sistemas de Computação, realizada em 16 de julho de 2010.

Aos dezesseis dias do mês de julho do ano dois mil e dez, às dezesseis horas, na Escola de Redes da Universidade Federal da Paraíba, reuniram-se os membros da Banca Examinadora constituída para examinar a candidata ao grau de Mestre em Informática, na área de "*Sistemas de Computação*", na linha de pesquisa "*Computação Distribuída*", a Srª. Cláudia Rossana Cunha. A comissão examinadora composta pelos professores doutores: Lucídio dos Anjos Formiga Cabral (DI - UFPB), Orientador e Presidente da Banca Examinadora, José Antonio Gomes de Lima (DI-UFPB), como examinador interno e Plácido Rogério Pinheiro (UNIFOR), como examinador externo. Dando início aos trabalhos, o Prof. Lucídio dos Anjos Formiga Cabral, cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra a candidata para que a mesma fizesse, oralmente, a exposição do trabalho de dissertação intitulado "*METAHEURÍSTICA HÍBRIDA GRASP E BUSCA TABU APLICADA AO PROBLEMA DE ESCALONAMENTO DE TAREFAS*". Concluída a exposição, a candidata foi argüida pela Banca Examinadora que emitiu o seguinte parecer: "*aprovado*" Assim sendo, deve a Universidade Federal da Paraíba expedir o respectivo diploma de Mestre em Informática na forma da lei e, para constar, a professora Tatiana Aires Tavares, Srª. Coordenadora do PPGI, lavrou a presente ata, que vai assinada por ela, e pelos membros da Banca Examinadora. João Pessoa, 16 de julho de 2010.

Tatiana Aires Tavares

Prof. Dr. Lucídio dos Anjos Formiga Cabral
Orientador (DI-UFPB)

Prof. Dr. José Antonio Gomes de Lima
Examinador Interno (DI-UFPB)

Prof. Dr. Plácido Rogério Pinheiro
Examinador Externo (UNIFOR)

Aos meus amados pais,
José Mário e Vera Lúcia,
bênçãos em minha vida.

AGRADECIMENTOS

Tudo que almejamos na vida depende sempre de muito trabalho e persistência, principalmente, nos momentos difíceis... É aí que contamos com a força que vem do amparo amigo, do incentivo e do companheirismo... Que faz toda a diferença... E pelo que muito venho a agradecer...

A Deus, por tudo que me foi permitido alcançar e viver até este momento... E, sobretudo, por ter colocado, em meu caminho, pessoas tão especiais como as que encontrei aqui.

A minha família, pelo apoio, incentivo e, principalmente, pela confiança que sempre me depositaram.

Ao meu orientador, Prof. Lucídio dos Anjos Formiga Cabral, pelas preciosas orientações durante todo meu trajeto neste curso e, também, por seu apoio e dedicação.

Ao meu co-orientador, Prof. Gilberto Farias de Sousa Filho, pelas imprescindíveis contribuições ao meu trabalho, bem como, pela dedicação e acompanhamento incessante.

Ao colega Alexander de Almeida Pinto, pelo apoio e colaboração.

A todos os professores que tive até hoje, que muito contribuíram para a minha formação.

Aos amigos (de ontem, de hoje e de sempre) que, na alegria ou na tristeza, estão comigo... E torcem por mim.

A todos vocês, meus sinceros agradecimentos.

RESUMO

Este trabalho aborda o problema de escalonamento de tarefas (*Job Shop Scheduling*) através da combinação das metaheurísticas GRASP e Busca Tabu. O estudo consiste em utilizar o GRASP na fase de construção da solução inicial, sugerindo, para tanto, um procedimento específico baseado no algoritmo de Coffman Gramah. Tal procedimento foi o grande diferencial deste trabalho, visto que oferece uma solução inicial qualitativamente superior, permitindo a redução do tempo de busca local, onde foi utilizada a metaheurística Busca Tabu, a qual demonstrou proporcionar melhores resultados, comparando-se com outras metaheurísticas que seguem a mesma base estrutural. A combinação GRASP e Busca Tabu foi avaliada sob duas implementações diferenciadas, sendo uma com a Busca Tabu em sua forma mais simplificada e outra com a Busca Tabu desenvolvendo um processo de busca otimizada — com o auxílio de um modelo matemático —, a qual demonstrou grandes progressos quanto ao tempo de processamento e a obtenção de resultados. Os resultados computacionais obtidos, quando comparados com os existentes na literatura, mostraram que a combinação GRASP e Busca Tabu é capaz de produzir boas soluções.

Palavras-chave: Escalonamento de processos, GRASP, Busca Tabu.

ABSTRACT

This work approaches the problem of the tasks scheduling (Job Shop Scheduling) through the combination of the GRASP and Tabu Search metaheuristics. The study consists of using the GRASP (*Greedy Randomized Adaptive Search Procedure*) in the construction phase of the initial solution, suggesting for that, a specific procedure based on Coffman Gramah algorithm. However, such procedure was the great differential in this study, since it offers an initial solution qualitatively better, what allows the reduction of the local search time, where it was utilized the Tabu Search metaheuristic in the local search phase, which provided best results when it was compared with other metaheuristics that have the same structural base. The combination GRASP and Tabu Search were evaluated under two differentiated implementations, being one with Tabu Search in its form more simplified and another one with Tabu Search developing a process of optimized search, using an aditional mathematical model, which demonstrated great advancements in relation at processing time and the obtained results. The computational results, when compared with the existing ones in literature, they had shown that GRASP and Tabu Search combination are capable to produce good solutions.

Word-key: Job Shop Scheduling, GRASP, Tabu Search.

ÍNDICE ANALÍTICO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

1. INTRODUÇÃO

Com o desenvolvimento, veio a crescente automação de processos em diversas áreas da indústria; tanto que, a cada dia, surgem máquinas capazes de executar tarefas antes não imaginadas. E nesse contexto estão as máquinas que desenvolvem desde tarefas simples, como parafusar uma peça em outra, até a modelagem e fabricação desta mesma peça. Com a possibilidade de uso desta tecnologia, logo surgiu a necessidade de se obter um melhor aproveitamento das máquinas e, por consequência, do tempo empregado num conjunto de tarefas. Donde sobrevém a exigência por uma definição de escalas de processos, que coordene satisfatoriamente o desempenho das máquinas. O cenário delineado trata do problema do escalonamento de processos, mais especificamente, do *Job Shop Scheduling Problem* — JSSP, tema central desta pesquisa.

Em um *Job Shop Scheduling*, as máquinas podem ser reais ou representativas, ou seja, podem ser máquinas operacionais, sistemas computacionais ou entidades vivas que se aliam ao processo desenvolvendo uma tarefa específica, como explica Alvarez-Valdes *et al* [03], esclarecendo que trabalhadores qualificados que executam tarefas manuais podem ser modelados como máquinas especiais. E as operações, por eles desenvolvidas, também podem ser estimadas como variáveis para o sistema. Considerando que o planejamento do escalonamento das tarefas acarreta na previsão de ocorrência destas, o tempo de execução de cada operação tem uma importância relevante, bem como, seus requisitos — se houverem, para que se estabeleça uma formulação dos possíveis escalonamentos a fim de se encontrar aquele que represente uma solução ótima.

É importante observar que este tipo de problema atende a uma estrutura básica identificada de modo próprio, tendo em vista que o cenário do problema seja um conjunto de máquinas que contribuem na execução de tipos diversos de produtos. A produção de cada produto constitui um *job* e ocorre em etapas, as quais recebem o nome de operações. Assim, a característica básica deste tipo de problema é a necessidade de se alocar operações (processos) às máquinas (os executores dos processos) ao longo do tempo, visando reduzir o tempo de parada das máquinas, em função da troca de produtos, na linha de produção, ou seja, o tempo total até a conclusão de todas as tarefas. O que faz da busca por um agendamento de tarefas o alvo a ser alcançado.

Um agendamento, ou escalonamento (*Scheduling*), pode ser entendido como uma forma de escalonar um conjunto de operações (processos) as quais envolvem planejamento, pois obedecem a múltiplas restrições.

Para tratar o ordenamento deste conjunto de operações, faz-se necessário estabelecer critérios que avaliem adequadamente as variáveis do problema, que envolve, principalmente, o melhor uso dos executores (máquinas, pessoas, conjunto de práticas, etc.). Tal descrição constitui-se num problema clássico de produção, o *Scheduling*, que é a programação do desenvolvimento de operações num tempo de duração previsto, tendo como objetivo principal a redução dos custos e o aumento da produtividade. De maneira que, no caso específico deste tipo de escalonamento, cada máquina executa uma única operação por vez e, tendo sido iniciado o processamento de uma dada operação, este não pode ser interrompido, ou seja, não ocorre a preempção.

Os métodos mais comuns de escalonamento proporcionam soluções de problemas que exigem programação otimizada (planejamento e controle) das operações que constituem os *jobs*.

Problemas de escalonamento (*scheduling problems*) envolvem questões de produção de bens e serviços, de definição de rotas, de organização de horários e muitos outros. A definição de tais escalas ou programações (*schedules*) é feita pelas características do problema, isto é, por sua forma de ser executada em conformidade com o cenário (máquinas) que a envolve.

Para resolver tais tipos de problemas, considerando instâncias de médio porte, é comum o uso de metaheurísticas que conduzem a soluções aceitáveis e diferenciadas, mas não únicas. Neste trabalho, duas metaheurísticas foram tratadas com este fim, o GRASP (*Greedy Randomized Adaptive Search Procedure*), amplamente discutido por Resende [20]; e a Busca Tabu — proposto inicialmente por Glover [14] —, utilizando-se, inclusive, de um método inédito para construir uma solução inicial aplicado às metaheurísticas e adaptado ao *job shop*, como será exposto nos capítulos subsequentes.

As metaheurísticas GRASP e Busca Tabu se desenvolvem em etapas, onde uma solução inicial é gerada e trabalhada para alcançar uma solução melhorada — otimizada, de modo que a implementação pode ser realizada de diversas formas. Nesse estudo, as duas metaheurísticas se combinam, o GRASP gera a solução inicial e a busca por uma solução otimizada é realizada pela Busca Tabu, sob duas formas diferenciadas de implementação. Na primeira implementação, a Busca Tabu foi realizada de modo simplificado. Na segunda, a Busca Tabu foi desenvolvida com o objetivo de realizar uma busca mais apurada por uma

solução otimizada. De maneira que, a partir da solução inicial — gerada pelo GRASP —, soluções aproximadas — ditas vizinhas—, são formuladas e examinadas pela Busca Tabu, só que, nesta segunda implementação tal busca foi conduzida com o auxílio do modelo matemático para o problema do *job shop*.

A Busca Tabu Otimizada desenvolvida nesse trabalho faz uso de técnicas de diversificação realizadas com melhoria da solução, diferentemente da diversificação encontrada na literatura, que faz a diversificação sem necessariamente otimizar a solução. Esta diversificação otimizada é auxiliada por uma ferramenta que permite o amplo processo de análise combinatória entre as soluções, como será exposto mais adiante.

1.1 O PROBLEMA

Como supra mencionado, a programação de processos interdependentes constitui-se num alvo de pesquisa a ser buscado por fábricas de diversas designações, as quais disponham de máquinas destinadas a papéis específicos com o fim de produzir, comumente, mais de um tipo de produto e, às vezes, utilizando uma mesma máquina em alguma etapa do processo; com o objetivo de minimizar o tempo ocioso e maximizar a produtividade.

Em relação a produtividade de bebidas, por exemplo, destacam-se entre suas operações: *i)* a modelagem (fabrico) de garrafas de vários tamanhos e formas; *ii)* a geração do líquido refrigerante, desde a criação de seu composto básico (xarope) até a gaseificação; *iii)* e o envase do líquido produzido. Cada *job* constitui-se na produção de um tipo de refrigerante e necessita passar por cada uma das máquinas executoras das operações descritas.

Outro exemplo de um *job shop* passível de ser tratado pelos algoritmos propostos que serão apresentados é o de uma fábrica de roupas masculinas. Tendo em vista que a produção de cada tipo de produto, entre camisa, calça comprida e bermuda, constitui um *job* diferente. Neste caso a produção de cada peça obedece a uma linha de produção específica, isto é, deve ser executada através de um conjunto de operações, seguindo uma sequência própria.

O perfil do problema apresentado se encaixa em outros já estudados. Despojando-se dos detalhes de caracterização, o que sobra são números relativos ao tempo de execução e identificação das respectivas máquinas executoras. De modo que o problema será tratado com seu perfil geral, a exemplo do que tem sido realizado em estudos pretéritos e, por conseguinte, aplicado às suas especificações para aferições finais.

Neste trabalho é abordado o problema de escalonamento de tarefas — *job shop* (JSSP) — através das metaheurísticas GRASP e Busca Tabu e suas respectivas variações, confrontando resultados obtidos, através de uma nova formulação do algoritmo, com os da literatura.

Esta dissertação está organizada em seis capítulos.

No Capítulo 2, são apresentados os tipos de *Job Schedule* existentes e as variantes do problema de escalonamento de processos abordado. No Capítulo 3, pode-se encontrar o embasamento conceitual para tratar o problema, iniciando com a formulação matemática, apresentando, em seguida, abordagens resolutivas para o problema na forma de revisão de literatura. No Capítulo 4, serão apresentados os métodos propostos para execução da pesquisa. No Capítulo 5, serão exibidos os resultados obtidos com cada método de tratamento, e também, um paralelo entre os métodos aplicados. Para concluir, no Capítulo 6 encontram-se as considerações finais.

2. VARIANTES DO PROBLEMA DE ESCALONAMENTO DE PROCESSOS

Conforme seja a classificação em que se enquadre um dado problema, também será a classificação dos *Jobs Schedules*, a qual ocorre pela forma de tratar tal problema; ou seja, de acordo com a adequação destes *jobs* às máquinas que as executam. Desta forma, há três tipos básicos de escalonamento: o *open shop*, o *flow shop* e o *job shop*. Onde a diferença entre estes problemas de escalonamento pode ser resumida da seguinte forma:

- *open-shop*: a sequência de máquinas não é especificada;
- *flow-shop*: toda tarefa tem a mesma sequência de máquinas;
- *job-shop*: cada tarefa tem a sua sequência de máquinas.

Uma vez que se identifique o tipo de problema, pode-se escolher uma estratégia para abordá-lo e buscar soluções que melhor considerem o perfil de cada um. Isto posto, pode-se observar algumas destas características a seguir.

2.1 OPEN SHOP SCHEDULING

O *Open Shop Scheduling* caracteriza os problemas cujas operações dos *jobs* não precisam de uma ordem para serem executadas. Desta forma, as operações são totalmente independentes, não existindo restrições quanto à ordem de execução destas.

Através deste tipo de escalonamento, as tarefas precisam ser processadas por todas as máquinas e isto pode ocorrer em qualquer ordem, objetivando, também, minimizar o tempo total de execução (*makespan*) e o tempo total de fluxo (*flowtime*).

Brasel *et al* [09] avaliam o uso de várias heurísticas construtivas para tratar deste problema, fazendo a relação entre o número de *jobs* e o número de máquinas.

2.2 FLOW SHOP SCHEDULING

O *Flow Shop Scheduling* caracteriza-se pela exigência de uma ordem de execução das operações pré-estabelecida, ou seja, seguindo uma sequência de operações estritamente

ordenadas. De modo que os *jobs* devem seguir uma direção uniforme, sendo executados de uma só vez, por configurar o desenvolvimento (de um mesmo produto) de forma repetitiva. Neste tipo de problema, o processamento das tarefas ocorre em linha, ou seja, as tarefas existentes passam por todas as máquinas que compõem o sistema, exatamente na mesma ordem. Assim, a *i*-ésima operação de cada uma das tarefas deverá ser sempre feita na máquina *i* e o tempo de processamento de cada tarefa em cada máquina é conhecido e imutável. Cada tarefa (*job*) tem de ser processada primeiro pela máquina 1, depois pela máquina 2 e assim por diante.

Percebe-se que ao invés de se projetar o escalonamento de operações, faz-se o planejamento das tarefas em relação ao uso das máquinas.

Alguns autores descrevem o *Flow Shop Scheduling* como um tipo particular de *Job Shop Scheduling*, onde a sequência das operações não depende dos *jobs*.

2.3 JOB SHOP SCHEDULING

O *Job Shop Scheduling* caracteriza-se por diferenciar as operações de cada um dos *jobs*, direcionando sua execução para máquinas específicas, o que restringe a execução de uma dada operação a uma só máquina. Este tipo de escalonamento (*scheduling*) permite que se estabeleçam diversas escalas, vez que pode haver flexibilidade quanto à ordem de processamento das operações — cuja execução é realizada apenas uma vez —, bem como, quanto à ordem de utilização das máquinas, que deve objetivar o tempo mínimo de ociosidade e o mínimo tempo total de execução (*makespan*). O que implica em buscar arranjos de operações cuja ordem de execução em cada máquina leve ao menor tempo possível.

Na visão de conjuntos, o *job shop scheduling* consiste de um conjunto de Jobs ($J = \{j_1, j_2, ..., j_p\}$), onde um *job* é composto por um conjunto de operações ($j_i = \{o_{i1}, o_{i2}, ..., o_{ip}\}$), e cada operação tem seu tempo de processamento. Sendo possível formar um conjunto de tempos de processamento ($t_i = \{t_{i1}, t_{i2}, ..., t_{ip}\}$) passíveis de serem identificados em relação às respectivas operações, como será melhor apresentado adiante. Assim, t_{ip} é o tempo de processamento da *p*-ésima operação do *i*-ésimo *job*.

Busca-se, portanto, o escalonamento tendo em vista os requisitos de execução de cada operação, os quais consistem em considerar a prioridade de execução das operações que estão prontas para serem executadas. O que envolve duas características principais, o estado de pronto e a prioridade. A primeira, definida pela inexistência de pendências (todas as

operações antecessoras já foram executadas) e, a segunda, por regras pré-estabelecidas. Como o escalonamento pode ser efetivado de diversas formas — ou seja, pode ocorrer adotando-se várias ordens de sequenciamento destas operações —, o escalonamento ideal consiste na opção que melhor otimize o conjunto de processos, tanto em relação ao tempo de execução das tarefas quanto em relação ao uso das máquinas. O *Job Shop* tradicional é caracterizado por permitir diferentes fluxos das ordens entre as máquinas e diferentes números de operações por ordem, que são processadas apenas uma vez em cada máquina, segundo relata Oliveira [18]

Além destas variáveis, algumas outras poderiam ser consideradas e estudadas por caracterizarem ainda mais as tarefas. Como exemplo, as datas de disponibilidade, tempos de preparação, restrições de precedência, restrições de elegibilidade, permutações, bloqueios e a soma das folgas (intervalos entre execuções). Contudo, o objetivo do escalonamento pode ser alcançado através da minimização das variáveis mais comuns, que são *makespan* e a soma dos atrasos ponderados.

2.3.1 REPRESENTAÇÃO DO PROBLEMA DE ESCALONAMENTO JOB SHOP (JSSP)

Em linhas gerais, um *Job Shop Scheduling* é a representação de uma programação de *jobs*, compostos por operações interdependentes (podendo haver precedência entre a execução das operações). Onde cada operação de um *job* deve ser desenvolvida por uma máquina diferente.

Para se representar um problema de *Job Shop Scheduling* (JSS) pode-se fazer uso de uma tabela onde variáveis componentes do problema estão presentes, seja nos dados, seja nas cores diferenciadas ou, mesmo, na distribuição dos *jobs* ao longo de uma escala de tempo. O mesmo ocorre em relação a representação gráfica da solução do JSSP.

Inicialmente, será vista a representação do JSSP através de tabela (ver Tabela 1), que destaca os *jobs* e suas operações. Onde, a primeira coluna indica as tarefas (*jobs*) existentes e as demais colunas relacionam as respectivas operações; cujas células informam a máquina em que deve ser executada tal operação e o tempo correspondente (entre parênteses). E o índice indicado para as operações (op) apontam o *job* a que pertence a operação (o primeiro) e a ordem de ocorrência (segundo índice). Assim, op_{31}:2 (3), por exemplo, significa que a primeira operação do *job* "3" será executada na máquina "2" e consumirá "3" unidades de

tempo. Além disso, pode-se perceber que para um dado *job*, suas operações serão executadas em máquinas distintas.

Tabela 1: Representação de um JSSP por tabela.

Trabalho (*Job*)	Operações a serem processadas (op: máquina (tempo))		
1	op_{11}:1 (3)	op_{12}:2 (3)	op_{13}:3 (3)
2	op_{21}:1 (2)	op_{22}:3 (3)	op_{23}:2 (4)
3	op_{31}:2 (3)	op_{32}:1 (2)	op_{33}:3 (1)

2.3.2 REPRESENTAÇÃO DA SOLUÇÃO USANDO GRÁFICO DE GANTT

Uma forma de se representar uma solução para o problema de JSS exibe um possível sequenciamento de execução das operações por parte das máquinas. Trata-se do gráfico de Gantt, (Figura 1), que é uma das ilustrações mais conhecidas para se representar uma solução para o problema de JSS. Neste gráfico existe, na vertical, a indicação das máquinas e, na horizontal, a linha de tempo; os retângulos representam as operações componentes dos *jobs*, encaixadas de forma que seu tempo de execução seja marcado na linha de tempo. Os *jobs* ficam claramente destacados pela cor do retângulo, o qual pode conter a indicação do *job* (índice único) ou do *job* e da ordem da respectiva operação (quando indicada por duplo índice). Com base na solução abaixo, percebe-se que o tempo total (*makespan*) de conclusão das operações é igual a 12 unidades de tempo.

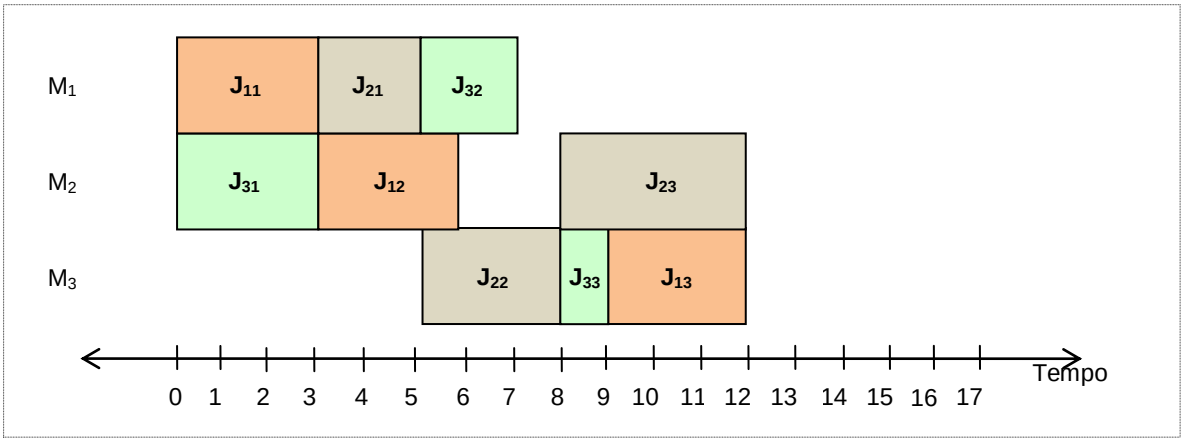

Figura 1: Representação de solução do JSSP no gráfico de Gantt.

2.3.3 GRAFOS DISJUNTIVOS

Tanto o problema quanto sua solução podem ser representados por grafos disjuntivos, pois seus conceitos auxiliam também na modelagem de ambos.

Representação do Problema

Um grafo parcialmente orientado constituído por outros grafos é uma representação para o JSSP que reúne as características mais relevantes do projeto de escalonamento. Tal representação é a mais usada na literatura, tendo sido, também, adotada neste trabalho.

Tal grafo tem uma formação conjuntiva e outra disjuntiva, como explica Blazewicz *et al* [08]. O grafo dispõe as tarefas (*jobs*) a serem processadas separadamente (em relação ao grafo, visualmente, em formação linear), destacando sua composição pelas respectivas operações constituintes; as quais são representadas pelos vértices do grafo e vêm identificadas por um rótulo ou por seu tempo de processamento. As operações de um mesmo *job* estão interligadas por uma dependência de prioridade de execução. Esta ligação é representada por arestas horizontais orientadas (num único sentido), as quais não podem sofrer modificações (de sentido) — a composição dos arcos horizontais forma o grafo conjuntivo —. Como cada operação de uma tarefa deve ser executada por uma máquina diferente, a interconexão não-orientada destas operações forma o grafo disjuntivo, o qual permite relacionar as operações pela máquina que as executa. Opcionalmente, pode-se representar a identificação da máquina por cores diversificadas, a exemplo do que se vê na Figura 2, subsequente.

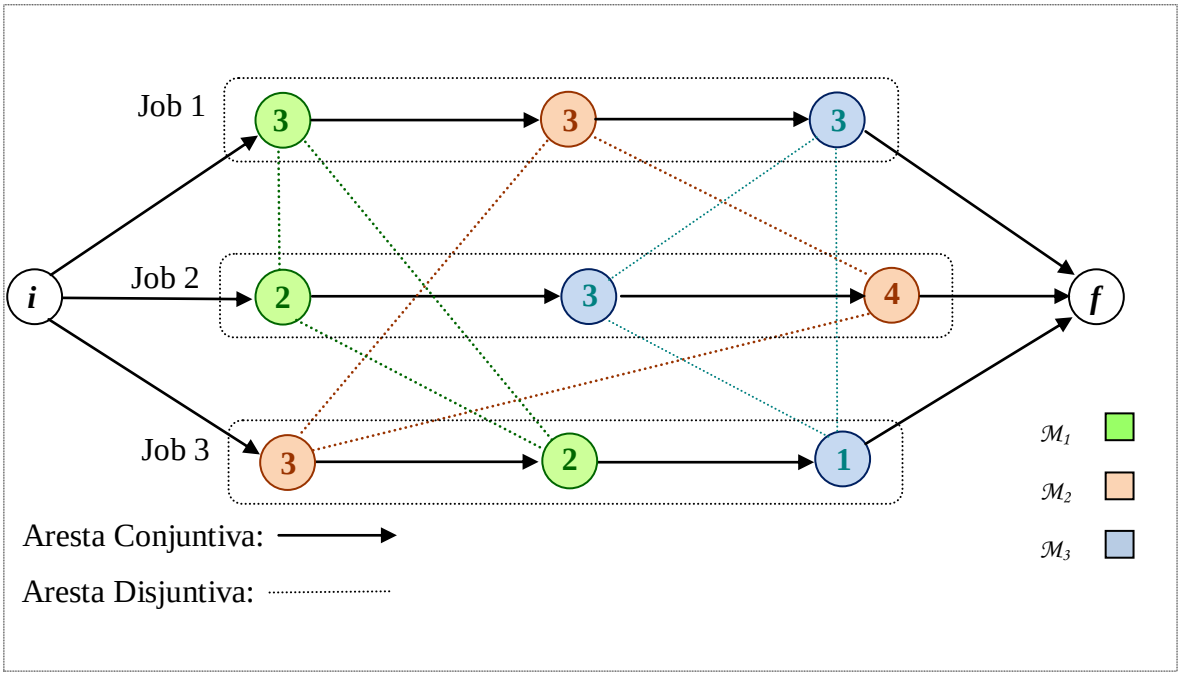

Figura 2: Grafo disjuntivo inicial (não orientação).

Representação da Solução

A simples orientação do grafo disjuntivo vem representar uma possível solução para o problema, logicamente, considerando o conjunto de arestas conjuntivas e disjuntivas. Contudo, apenas as disjuntivas podem sofrer modificações, já que as conjuntivas representam a restrição de precedência entre as operações. Desta forma, como exposto anteriormente, tendo em vista um conjunto de tarefas que configurem uma instância de problema, buscar uma solução significa propor um escalonamento das operações constituintes. E, como existe certa flexibilidade no arranjo das operações, pode-se alcançar diversas possibilidades de execução, isto devido as múltiplas ordenações passíveis de serem obtidas; a exemplo do que se observa com as opções geradas pela mudança nas arestas disjuntivas (no sentido da execução das operações), desde que, sejam mantidas acíclicas (ver Figura 3). A necessidade de manter o grafo disjuntivo acíclico, deve-se ao fato de uma operação só poder passar por uma máquina apenas uma vez.

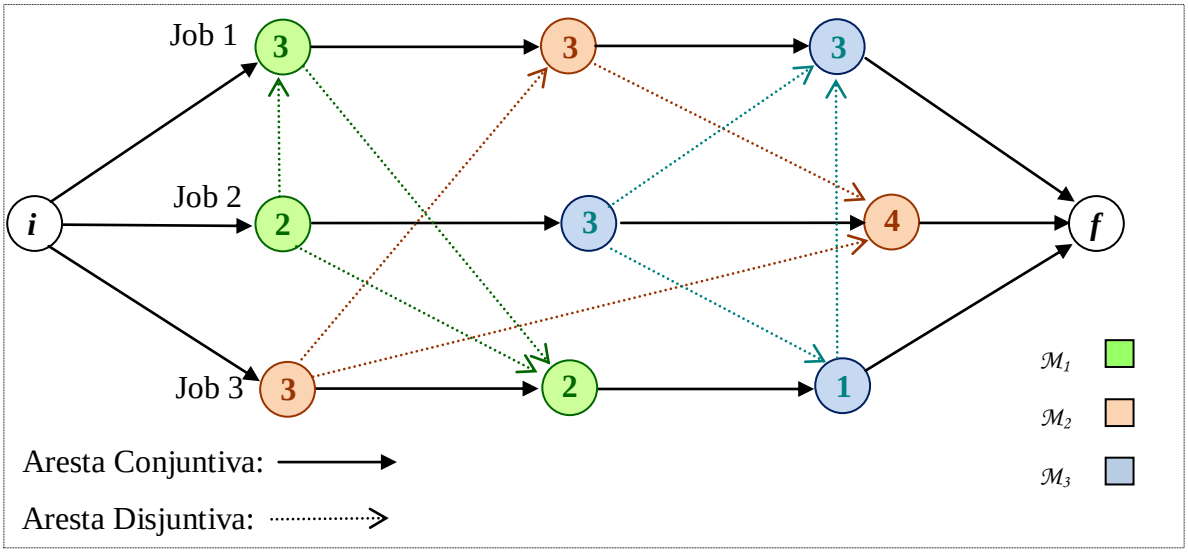

Figura 3: Grafo disjuntivo orientado.

Avaliação da Solução Encontrada

Uma vez encontrada uma solução para o problema de *job shop scheduling* existe a necessidade de se avaliar essa solução para que se possa compará-la com outras soluções. Uma forma de se efetuar tal tarefa é se analisar o tempo de execução das operações mais demoradas. Dessa forma, analisa-se o subconjunto de operações que implicam em maior tempo para o término dos processos, o que consiste em se avaliar o tempo do caminho crítico.

Método do Caminho Crítico

Em relação ao JSSP, o método do caminho crítico consiste em encontrar o maior tempo de execução para que operações críticas vinculadas se realizem. Ou seja, busca-se conhecer a ordem de execução das operações que levam o maior tempo para serem executadas para, então, encontrar-se as possíveis soluções ótimas. Tendo em vista a representação do problema através de grafos disjuntivos, sabe-se que haverá entre as operações arestas conjuntivas (intermediando as operações de um mesmo *job*) e arestas disjuntivas (relacionando operações que comungam de uma mesma máquina). Uma vez definida a solução inicial, define-se também o direcionamento das arestas disjuntivas e, assim, pode-se encontrar o caminho crítico desta solução.

Uma particularidade do tratamento adotado do JSSP através de grafos disjuntivos é a adição de duas operações virtuais (sem tempo de execução) ao modelo, como apoio, para representar o início (*head* - *cabeça*) e o fim (*tail* - *cauda*) da escala de processos.

Desta forma, partindo-se da operação virtual de início, busca-se encontrar um valor para cada uma das arestas, tanto conjuntivas quanto disjuntivas — o que ocorre no estudo proposto — através de uma função seletiva que avalia o tempo de execução das operações paulatinamente. Tal seletividade avalia as operações que se encontram no mesmo nível, ou seja, em condição de independência para ser executada. De modo que as arestas que saem de tais operações recebam o valor do tempo destas. As demais arestas devem considerar não apenas o tempo das operações que lhe dão origem, mas também, o valor da aresta que lhe antecede (incide na referida operação) com maior valor. Ou seja, o valor dado a referida aresta é gerado pela soma dos dois valores. Em relação às últimas operações (dos *jobs*), faz-se a ligação entre estas e a operação virtual final, considerando-se o valor da operação de incidência como nulo e os valores de tais arestas (ver Figura 4).

Ao final do procedimento, ao se observar a formação de uma solução em relação ao caminho crítico, pode-se perceber que este consiste exatamente das arestas de maior valor encontrado que ligam o ponto virtual inicial até o ponto virtual final. Outro aspecto a ser observado é que, ao se definir uma ordem de execução entre duas operações quaisquer, define-se também o direcionamento das arestas que ligam as operações. A mudança no sentido de uma destas arestas implica em uma nova solução e precisa atender a um conjunto de restrições para ser considerada válida. Como as arestas conjuntivas estabelecem a necessária precedência entre as operações para que uma tarefa seja realizada, fica implícito que esta ordem não pode ser quebrada e que, portanto, tais arestas não são passíveis de mudanças.

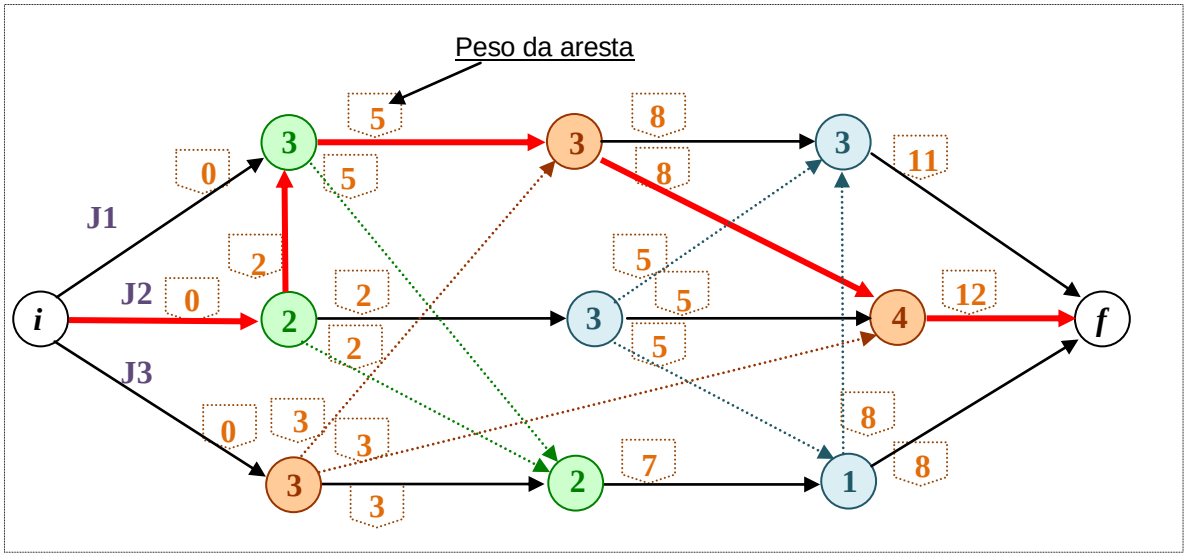

Figura 4: Caminho crítico da solução apresentada (CP = 12).

3. FUNDAMENTAÇÃO TEÓRICA

Considerando a possibilidade de se analisar um caso prático específico, são encontrados inúmeros trabalhos que utilizam os conceitos de *Job Shop Scheduling* para resolver problemas de diversas áreas do conhecimento, sempre promovendo a otimização dos processos. Assim, encontram-se exemplos na Biologia (Bioinformática), na Computação — com a arquitetura de computadores (modelando problemas de sequenciamento) —, na Engenharia de Produção (buscando aumentar a produção e por consequência, os lucros) e em muitas outras áreas, cabendo ainda a observação das soluções específicas.

Os métodos de resolução para problemas do tipo JSSP podem ser enquadrados em duas linhas de desenvolvimento: por métodos exatos e por métodos aproximativos. Os métodos exatos oferecem soluções exatas, objetivas, de eficiente resolução. São métodos que, normalmente, exigem muita capacidade computacional e, portanto, demandam um tempo de processamento que cresce em proporção exponencial em relação ao tamanho da entrada de dados. Assim sendo, são aplicáveis apenas para problemas de pequeno porte [05, 10, 11]. Já os métodos aproximativos necessitam de métodos não-exatos, heurísticos, dado que sua resolução exata é computacionalmente intratável. Ou seja, buscam por uma solução ou por um conjunto de soluções, sem garantir, necessariamente, uma solução ótima, mas tão somente uma solução aproximada. São métodos que se utilizam de heurísticas conhecidas e trabalham bem com instâncias maiores, como relata um *survey* sobre métodos heurísticos para JSSP em [19].

A seguir será apresentada a formulação matemática para o problema, bem como as abordagens aproximativas utilizadas, uma vez que são consideradas as instâncias maiores para este problema, que é reconhecido na literatura como NP-Difícil [15].

3.1 FORMALIZAÇÃO MATEMÁTICA DO JSSP

Será utilizada a definição de conjuntos, tendo como base conceitos de grafos disjuntivos (vistos anteriormente), para definir o problema e destacar as características dos elementos constituintes do sistema, tratando-os individualmente.

Definição Geral:

Um problema de *Job Shop Scheduling* pode ser definido por uma instância descrita da seguinte forma:

$P = \langle M, J \rangle$ onde,

M – representa o conjunto de máquinas diferentes;

J – representa o conjunto de *jobs*;

Expandindo tal representação, tem-se:

$M = \{m_1, m_2, ..., m_m\}$

$J = \{j_1, j_2, ..., j_n\}$, donde $j_i = \{o_{i1}, o_{i2}, ..., o_{in}\}$, sendo j_i o conjunto de operações de um dado *job* e $p_i = \{p_{i1}, p_{i2}, ..., p_{in}\}$, que representa o tempo de processamento de cada uma destas operações respectivamente.

Conforme já explicado, o objetivo é achar uma sequência de operações para cada máquina a fim de minimizar o tempo total de processamento (*makespan*) $C_{max} = \max_{j=1,n} C_j$ — após o processamento do último *job*, C_j denota o tempo total desta última operação de um *job* J_j (j = 1, ..., n).

Lembrando que: *i)* cada operação de *j* utiliza uma das *m* máquinas com um tempo de processamento fixo; *ii)* cada máquina processa uma operação por vez, sem interrupções; *iii)* o processamento das operações obedece a um roteiro pré-estabelecido.

O JSSP modelado por um grafo disjuntivo pode ser representado matematicamente por $\mathcal{G} = (G, D)$, onde G = (X, U) é um grafo conjuntivo relativo à sequência dos *jobs.* Formado por um conjunto (X) de vértices (representantes das operações) e por um conjunto (U) de arestas (representantes das conjunções). D é o conjunto de disjunções, onde uma disjunção [i, j] = {(i, j), (j, i)} está associada a um par de operações processadas pela mesma máquina. Uma característica que deve ser lembrada é a adição das duas operações virtuais ao grafo, para marcar o início e o fim do processo.

No grafo disjuntivo o esquema $\mathcal{G} = (G, D)$ possui um conjunto de tempos iniciais T, onde

$T = \{t_i \mid i \in X\}$ tal que:

- as restrições conjuntivas são satisfeitas por:

$t_j - t_i \geq p_i, \forall (i, j) \in U$ – que demonstra que operações pertencentes ao mesmo *job* têm restrições de prioridade, onde uma só inicia com o término de sua antecessora.

- as restrições disjuntivas são satisfeitas por:

$t_j - t_i \geq p_i \ \vee \ t_i - t_j \geq p_j$, $\forall$ $(i, j) \in D$ – que demonstra a não sobreposição de execução de duas operações, onde, a execução pode se dar tanto com início em uma como em outra; contudo, a segunda só se inicia com o término da primeira.

Detalhando a formulação matemática apresentada anteriormente, temos um conjunto de fórmulas que especificam melhor as condições do grafo disjuntivo. Seja uma variável binária x_{ij} para representar cada disjunção, onde a operação i precede a operação j, tem-se:

(1) min C_{max}

(2) $\forall$ i $\in$ X, $t_i \geq 0$

(3) $\forall$ i $\in$ X, $t_i + p_i \leq C_{max}$

(4) $\forall$ (i, j) $\in$ U, $t_i + p_i \leq t_j$

(5) $\forall$ [i, j] $\in$ D, $x_{ij} \in \{0, 1\}$

(6) $\forall$ [i, j] $\in$ D, $t_i + p_i - M(1 - x_{ij}) \leq t_j$ $\vee$ $\forall$ [i, j] $\in$ D, $t_j + p_j - M x_{ij} \leq t_i$

Fazendo-se uma releitura sobre as fórmulas anteriormente apresentadas, podem-se extrair todas as regras que devem estar contidas em um grafo disjuntivo. O objetivo do problema, que é a minimização do *makespan*, pode ser expresso na minimização do *makespan* da última operação a ser executada (1). E, relacionando-se os vértices do grafo às operações, estas terão, no mínimo, o tempo nulo (o que ocorre no caso dos vértices extremos) que representam as operações virtuais (2). Considerando-se uma operação qualquer, seu tempo de processamento pode ser relacionado ao *makespan*, pois será, no máximo, igual a soma do tempo de início de sua execução e seu respectivo tempo de processamento. E no caso da última operação a ser executada, seu tempo de início somado a seu tempo de processamento será equivalente ao *makespan* (3). Em se tratando de uma conjunção, sendo considerado um par de operações — onde uma não se sobrepõe a outra, ou seja, a segunda operação só tem início quando a primeira termina —, o tempo de início desta segunda será igual ou maior a soma do tempo inicial da primeira e seu respectivo tempo de processamento (4). Em relação a uma disjunção, tem-se um parâmetro para auxiliar na avaliação de pertinência de um par de operações — x_{ij} representa o indicativo de existência da referida disjunção no sentido de *i* para *j* (se $x_{ij} = 1$ trata-se de uma disjunção de *i* para *j* e $x_{ij} = 0$, indica que a disjunção não ocorre no sentido indicado) (5). Dadas duas operações disjuntivas, a não sobreposição e a sequência de execução dessas operações será comprovada quando representada pelas fórmulas disjuntivas (6), que podem sinalizar o sentido de ocorrência da disjunção. Isto é, se a

operação i precede a operação j ou o caso contrário. Senão houver disjunção no sentido indicado, as referidas fórmulas demonstrarão que o tempo de início da segunda operação é maior do que a soma do tempo de início com o tempo de processamento da primeira, para tanto, existe a constante M de dimensão superior que ressalta a inexistência da disjunção no sentido indicado, sendo definida como equivalente ao valor do *makespan* no pior caso de escalonamento possível.

3.2 ABORDAGEM APROXIMATIVA

Como anteriormente mencionado, estudos revelam que problemas de *Job Shop Scheduling* podem ser tratados utilizando-se de diversas heurísticas e, principalmente, não unicamente de uma heurística apenas, mas de algoritmos híbridos, os quais formulam suas soluções tendo como base um algoritmo e se utilizam, em alguma fase, de uma técnica comum a outro algoritmo. Tais variações acabam por oferecer melhorias que proporcionam uma otimização no tempo de processamento, ao que se confirmam estudos comparativos.

A metaheurística GRASP [02, 07] é apontada como modelo de algoritmo em pesquisas sobre JSSP, sendo, inclusive, utilizada em composição com outros modelos, em alguma fase do desenvolvimento [21, 02]. O mesmo ocorrendo com metaheurísticas como Busca Tabu [25], Algoritmos Genéticos [12], ILS [21] e *Simulated Annealing* [23, 26], entre outros. Tendo sido encontrado na literatura, estudos correlatos com o problema abordado [07, 25], cujas idéias influenciaram o desenvolvimento do presente trabalho, principalmente, os trabalhos destacados neste capitulo. Primeiro, pela inspiração em relação a modelagem do problema usando grafos disjuntivos, segundo, pela estruturação das vizinhanças adotadas e, por último, pelo uso de técnicas de intensificação no processo de busca por melhores soluções.

3.2.1 A GRASP FOR JOB SHOP SCHEDULING

Binato *et al* [07] propuseram uma solução para o JSSP adotando o GRASP como base e utilizando grafos disjuntivos na modelagem do problema. Empregaram o método do caminho crítico ao grafo disjuntivo para avaliar as soluções encontradas.

Neste trabalho o JSSP foi representado em função das operações, seu tempo de processamento e suas respectivas máquinas. A construção da solução inicial analisa o tempo

inicial de processamento de uma operação relativo ao de outra, ou seja, o tempo inicial de processamento de uma operação pode ser conhecido pelo tempo inicial de processamento de uma operação antecessora e seu respectivo tempo de processamento. Foi utilizada também uma técnica de intensificação na fase de construção, que consiste em analisar uma lista elite de soluções segundo um parâmetro obtido de um paralelo entre dois *makespans* (de uma solução tida como ótima e outra candidata) e a variação causada pela inclusão de uma operação em certa ordem de execução. Observa-se, contudo, que nesse caso, a variação analisa um bloco de iterações. Nesta modelagem, a busca local utiliza a técnica do caminho crítico aplicada ao grafo disjuntivo, com relação à solução produzida na fase de construção, para gerar uma nova solução através da troca de ordem de execução entre operações que compartilham a mesma máquina. Os estudos de Binato *et al* [03] demonstraram que a análise do caminho crítico também poderia ser utilizada já na fase de construção da solução elite, interferindo na função heurística gulosa em tal fase, na formação da lista restrita de candidatos (RCL — *Restricted Candidate List*).

A relevância desta pesquisa se deve, portanto, às variações realizadas para executar o algoritmo, tanto através do uso de uma estratégia de intensificação na fase de construção, como adotando a técnica de POP (*Proximate Optimality Principle*) na fase de busca local. Essa técnica consiste em procurar por uma solução melhor tomando-se uma instância e avaliando-a parcialmente, como se efetuasse a composição paulatinamente, sem, necessariamente, testá-la em seu formato final. Dado que uma solução é formada pelo conjunto ordenado de operações existentes, ao se aplicar este procedimento na fase de construção, o escalonamento das operações é feito um a um até que todas as operações tenham sido escalonadas, respeitando-se a prioridade das operações de cada *job*. Dessa forma, considera-se um conjunto parcial de operações escalonadas na fase de construção e realiza-se a busca local (parcial) com o objetivo de minimizar o tempo total de execução desta solução parcial. Para tanto, a busca local é realizada com uma leve modificação do grafo disjuntivo, o qual é construído usando apenas operações já escalonadas, deixando as operações remanescentes concentradas junto ao nó final, e considerando para a troca apenas as arestas disjuntivas que interligam tais operações.

A vantagem deste método encontra-se na possibilidade de se avaliar quão melhor ou pior uma solução pode se tornar com a inclusão de uma (nova) operação em dada ordem de posição. O POP aplicado acaba por corrigir imperfeições geradas na fase de construção. Nessa aplicação, a escolha da lista RCL, considera a possibilidade de não se adotar a aleatoriedade absoluta, aplicando-se um escalonamento parcial onde apenas parte das operações é

considerada na seleção. Dessa forma, a busca local pode ser realizada para uma solução apenas parcial com o objetivo de se reduzir o *makespan*, o que pode abreviar o tempo em busca de uma solução. Contudo, observa-se que o POP não é aplicado a cada iteração de inclusão de operação, mas sim, em um estado definido e condicional em relação, por exemplo, ao número de operações já escalonadas. O critério de parada utilizado foi a qualidade da solução, ficando comprovado que o GRASP incrementado pela intensificação e pelo POP oferece melhores resultados do que sem. Tais variações foram analisadas em relação as mudanças no número de *jobs* e de máquinas.

Outra técnica de intensificação muito utilizada é a do Path Relinking (reconexão por caminhos), que foi empregada na fase de pós-otimização por Armentano e Araújo [06]; bem como na fase de iterações, por Cravo, Ribeiro e Lorena [12]. Nestas duas pesquisas o uso da técnica de Path Relinking demonstrou a obtenção de melhores resultados. O uso do Path Relinking em relação ao GRASP melhora substancialmente o conjunto-solução, tanto sendo aplicado como estratégia de pós-otimização entre os pares de soluções elite, quanto como estratégia de intensificação a cada ótimo local obtido após a fase de busca local. Aiex, Binato e Resende [02] destacam a relevante colaboração do Path Relinking ao GRASP como estratégia de pós-otimização, aplicada a um conjunto elite de soluções encontradas (entre a solução encontrada pelo GRASP e um conjunto gerado na busca local), só executando o Path Relinking em intervalos de 500.000 iterações. Esta pesquisa gerou duas implementações paralelas, do GRASP com Path Relinking para JSSP e do GRASP com intensificação e POP (com POP sendo executado a 40% e 80%). As estimativas probabilísticas demonstraram que a última implementação tem menor custo para encontrar as melhores soluções.

3.2.2 A TABU SEARCH ALGORITHM WITH A NEW NEIGHBORHOOD STRUCTURE FOR THE JOB SHOP SCHEDULING PROBLEM

O trabalho de Zhang *et al* [25] direcionado à resolução do JSSP apresenta o método do caminho crítico aplicado juntamente com a metaheurística Busca Tabu (*Tabu Search*) para se avaliar, principalmente, a vizinhança. A forma de se estruturar a vizinhança foi modelada, também, em função de grafos disjuntivos. Contudo, aqui o algoritmo proposto utiliza a estrutura de blocos (o caminho crítico é constituído por blocos) na movimentação de vizinhança para atingir uma "nova" solução vizinha à existente. Assim, a partir do caminho crítico, identificam-se seus blocos formadores, onde um bloco é a sequência máxima de

operações críticas adjacentes que sejam processadas na mesma máquina. Na proposta de Zhang *et al* [25], o algoritmo de busca local aplica pequenas perturbações a soluções aceitáveis obtidas, o que se realiza através de perturbações no caminho crítico, com a reordenação de uma sequência de operações pertencentes ao mesmo bloco. Esta prática pode vir a gerar um vizinho (solução) com o *makespan* melhor do que a solução corrente.

Várias propostas de solução foram lançadas utilizando-se deste mesmo conceito, tendo sido identificadas simplesmente de N (de *neighborhood*) — N1, N4, N5 e N6. E com base nestas, o trabalho em questão trouxe uma inovação, em relação às existentes, anteriormente citadas. Onde N1 gera a vizinhança pela troca (*swapping*) em pares adjacentes de operações críticas na mesma máquina, que não permite a geração de soluções inaceitáveis, a partir de uma solução exequível; bem como, determina que a permutação entre operações não-críticas não pode criar uma solução aceitável. Contudo, foi através dos modelos N4, N5 e N6 (Figuras 5, 6 e 7, respectivamente) que foram encontrados os melhores resultados, os quais têm em comum o fato de exigir que o *swapping* se realize dentro de um mesmo bloco crítico, seja pela inserção de uma operação no início ou no fim do bloco. A forma como ocorre a troca de ordem de execução das operações diferencia a vizinhança gerada. De modo que, as movimentações do padrão N4 se configuram por trocas entre operações mais distanciadas — é a menos restrita das três. As movimentações do padrão N5 envolvem apenas trocas entre operações sucessivas nas extremidades de um bloco crítico, as quais devem ocorrer no início ou no fim do bloco. Já movimentações do padrão N6 ocorrem sempre entre a primeira metade ou entre a segunda metade de um bloco, e é considerada uma extensão de N5. Onde, para gerar uma vizinhança, cada padrão faz uso isolado de cada um dos movimentos demonstrados nas figuras a seguir.

O destaque deste trabalho está na inovação apresentada, que consiste em considerar a troca de uma operação interna para o início ou para o fim de um bloco, o que conduz a uma vizinhança maior, uma vez que investiga um espaço maior de possibilidades. Trata-se da estrutura de vizinhança estendida adicional proposta por Zhang *et al* [25], como mostra a Figura 8.

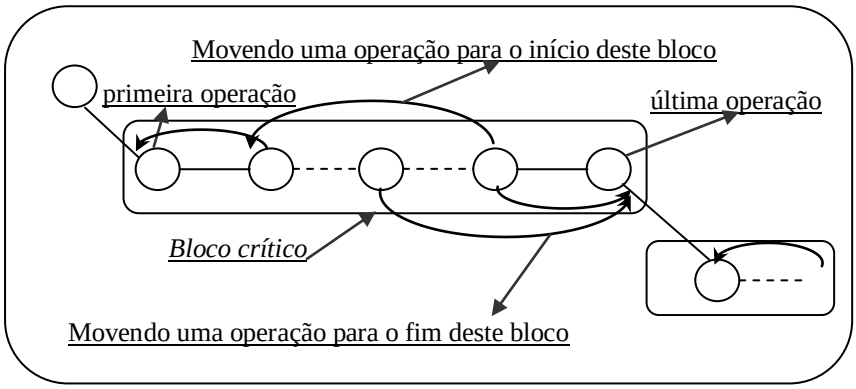

Figura 5: Movimentos de vizinhança N4

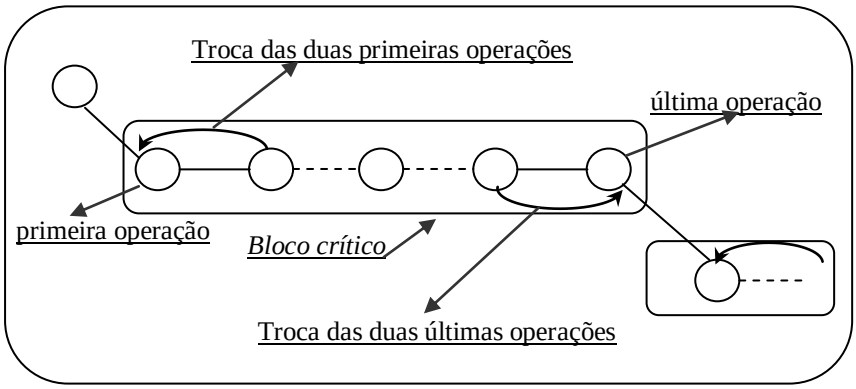

Figura 6: Movimentos de vizinhança N5

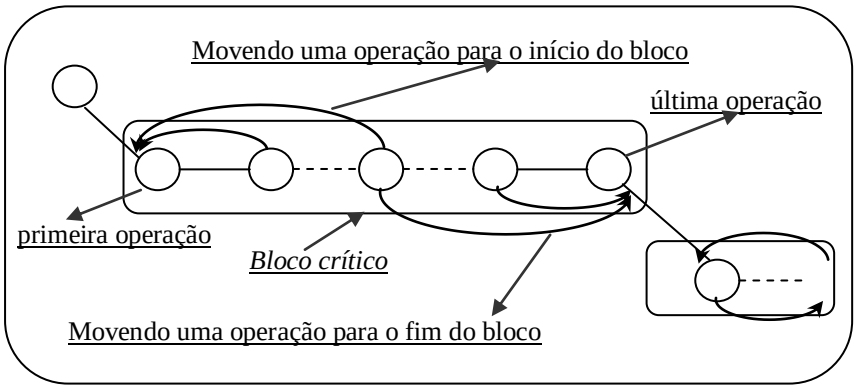

Figura 7: Movimentos de vizinhança N6

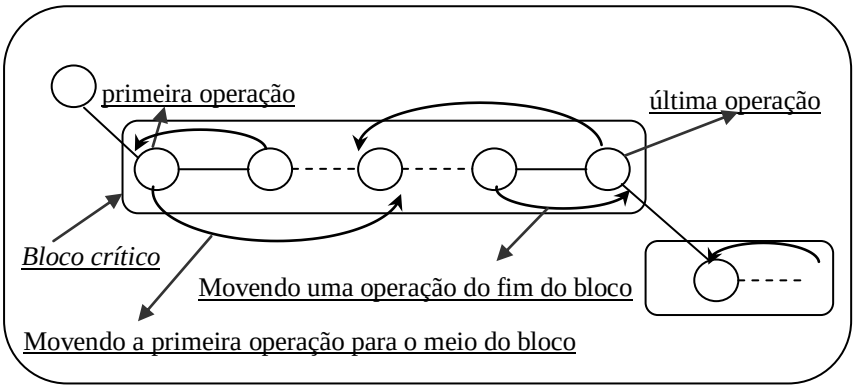

Figura 8: Movimentos de vizinhança estendida proposto por Zhang et al.

3.2.3 OUTROS TRABALHOS

Em todos os processos subsequentes a geração da solução inicial pode ser mais bem sucedida, quanto melhor seja a referida solução encontrada. Técnicas de intensificação ou o simples procedimento de busca local são mais bem avalizados com esta condicional, isto é, a qualidade da solução inicial e o hibridismo favorecem a construção de propostas resolutivas.

Estratégias de intensificação, como o *path relinking*, podem ser inseridas ao algoritmo com o objetivo de concentrar as buscas em regiões consideradas mais promissoras. Sendo possível guardar informações sobre os movimentos não-promissores, com o intuito de evitar que tais movimentos voltem a fazer parte de alguma solução. A busca por melhores ótimos locais pode prosseguir até que um número de iterações seja alcançado.

Moreira *et al* [16] também procuraram destacar (e comprovar) as melhorias obtidas através das técnicas de intensificação comprovando resultados empíricos paralelos. A pesquisa desenvolvida teve como objeto o JSS bi-objetivo. A minimização do *makespan* e do atraso total são considerados na avaliação e na busca por uma solução ótima, tendo como referência o Algoritmo Genético (AG) aplicado ao JSSP, sob três nuances em suas estratégias de solução. Os desempenhos comparativos são apresentados nas seguintes versões do algoritmo: AG Básico (AGB); AG com Busca Local (AGBL) e AG com Busca Local e com pós-otimização usando *Path Relinking* (AGBLPR).

O artigo dispõe sobre um AG que usa mecanismos de busca local, como forma de adaptação da população de soluções, e um procedimento de pós-otimização baseado na técnica *path relinking*. A adaptação da população consiste na inserção de soluções dominantes (soluções de elite) na população.

Em linhas gerais, nesta pesquisa a seleção é feita por torneio. Ou seja, classifica-se a população em sub-grupos disjuntos, calculando-se, em seguida, a distância entre tais soluções (*crowding*) com a finalidade de compará-las. O *crowding* define que tanto as soluções extremas quanto as mais dispersas possuem as maiores distâncias. E a mutação consiste na troca de duas tarefas aleatórias.

4. MÉTODO PROPOSTO

Embora haja muitos meios para se encontrar soluções para o JSSP, os métodos heurísticos apontam para uma construção da solução aceitável, que ocorre em etapas. Após se gerar uma solução inicial, essa é formulada e depois refinada até que se encontre outra solução (chamada de vizinhança) que seja ainda melhor, quando comparada com as outras soluções conhecidas de uma mesma execução. De modo que, serão adotados alguns modelos de heurística como base para o projeto, objeto desta pesquisa, do qual serão extraídos os parâmetros que fundamentarão os resultados a serem apresentados.

As heurísticas irão apoiar a resolução de cada etapa do problema, seja na sua forma pura ou híbrida — heurística usada em conjunto com outra(s) —. Conforme exposto nos capítulos anteriores, o problema proposto foi formulado utilizando grafos disjuntivos, o que proporciona a base para os métodos resolutivos. A resolução do problema nesta pesquisa se restringe ao uso das metaheurísticas GRASP e Busca Tabu, as quais se desenvolvem, essencialmente, em duas fases. A escolha por tais metaheurísticas se deve dada a indicação de bons resultados em relação ao problema, como foi visto na literatura encontrada; bem como, pelo aspecto construtivo da primeira fase que vai ao encontro do modelo de criação da solução inicial proposto.

Ambas as metaheurísticas se baseiam em uma heurística construtiva na primeira fase e uma heurística de busca local, na segunda fase. As referidas metaheurísticas foram usadas sob uma formulação híbrida, a fim de obter comparativos que possam fundamentar as considerações finais, comprovadas por resultados apresentados nos capítulos subsequentes.

E, como se verá, após encontrar a solução inicial, os métodos usados na busca por uma solução melhorada consistem em aplicar movimentos de vizinhança e métodos de otimização de busca às metaheurísticas e comparar os resultados experimentais obtidos por cada uma com os conhecidos na literatura, uma vez que partem da mesma solução inicial, cuja formulação é o fator diferenciador deste trabalho. A forma como tal solução é obtida tem a finalidade de, a princípio, partir de uma solução de qualidade e, assim, reduzir o trabalho de busca por uma solução melhor, resultante de um processo de busca local ou de intensificação de busca, como exposto neste capítulo.

4.1 MODELAGEM DO PROBLEMA

Neste trabalho, o JSSP foi modelado em função do método dos grafos disjuntivos, o que proporciona não apenas uma forma de modelar o problema, como também de representar a solução, além de ser uma estrutura que favorece a avaliação da solução encontrada. Como demonstrado nos capítulos anteriormente apresentados, a estrutura articulada permite tanto a formulação matemática como a visualização gráfica da solução, como escreve Blazewicz *et al* [08].

4.2 GRASP

O GRASP é um método iterativo proposto por Feo e Resende [13] e se desenvolve em sua forma básica em duas fases: a fase de construção e a fase de busca local (Figura 9). Embora, a seleção por uma solução para um problema não fica restrita a estas duas fases, uma vez que, qualquer uma delas pode ser incrementada e melhor desenvolvida, de forma independente. Tanto que técnicas de aprimoramento vêm sendo inseridas no processo, a fim de constituir propostas de resolução mais eficazes em algum aspecto, seja através de técnicas de intensificação ou de pós-otimização. E como o GRASP, normalmente, trata de problemas da ordem de complexidade NP-Difícil, poderia ficar indefinidamente em busca de uma solução ótima, o que exige a adoção de um critério de parada na fase de busca local, a qual pode se referir a definição do número de iterações.

```
procedimento GRASP( )
1       enquanto (iter ≤ MAX_ITER) faça
2               s0 ← Construção (g (.),α, s0);
3               s ← Busca_Local (s0, Avalia_solucoes (.),Vizinhança( ) );
4       fim-enquanto;
5       Retorne s;
fim GRASP;
```

Figura 9: Procedimento GRASP.

Na fase de construção, uma solução inicial é formada elemento a elemento a partir de uma Lista de Candidatos (LC). A cada etapa de construção a LC se renova com os elementos

remanescentes, sendo estes candidatos avaliados através de uma função adaptativa gulosa e mantidos de forma ordenada segundo o seu valor guloso. Da LC será gerada uma Lista Restrita de Candidatos (LRC), que contém os melhores candidatos, também formada iterativamente com a solução desenvolvida. Da lista restrita será extraído o próximo elemento da solução, com base numa componente de aleatoriedade, o parâmetro α. A componente α indica o nível de gulosidade e aleatoriedade de LC e seu valor influencia diretamente a escolha do elemento de LRC. Assim, para que uma operação seja integrada à solução ela deve ser escolhida aleatoriamente da LRC. E a formação da LRC é baseada no *cálculo do filtro*, alimentado pela componente α, como se observa na Figura 10. O procedimento de escolha das operações remanescentes ocorre a medida que a LC vai sendo composta e por consequência a LRC. No caso, a escolha pela próxima opção ocorre de forma aleatória em relação aos melhores elementos que compõem a LRC.

```
procedimento Construção (Função_gulosa (.),α, s0);
1       s0 ← Ø;
2       Inicialize o conjunto LC de candidatos;
3       enquanto (LC ≠ Ø) faça
4               g(tmin) = min { g(t) | t ∈ LC };
5               g(tmax) = max { g(t) | t ∈ LC };
6               LRC = {t ∈ LC | g(t) ≥ g(tmin) – α (g(tmax) - g(tmin))};
7               {t} ← escolha_Aleatoria(LRC);
8               s0 ← s0 ∪ {t};
9               Atualize o conjunto LC de candidatos;
10      fim-enquanto;
11      Retorne s0;
fim Construção;
```

Figura 10: Procedimento GRASP.

Este procedimento pode ser gerado quantas vezes seja requisitado, podendo obter uma solução inicial diferente em cada uma. De modo que cada solução terá sua vizinhança a ser explorada em busca de uma melhor solução, já que tal solução pode não representar um ótimo local.

Na fase de busca local, serão realizadas tentativas de busca por melhores soluções a partir da solução gerada na fase anterior: trata-se da busca pelo ótimo local. Tendo em vista a solução aceitável, o algoritmo faz uso de técnicas para se construir uma vizinhança (soluções próximas a solução aceitável gerada inicialmente). No GRASP básico a solução inicial é, então, comparada às soluções encontradas (vizinhança). Contudo, tal comparação é realizada aos pares, abandonando-se sempre a pior solução, ou seja, aquela com maior *makespan*; já

que não se guarda o histórico das soluções encontradas, mas apenas da solução corrente e da nova solução. Conforme seja a técnica de obtenção da solução inicial aceitável, ter-se-á uma solução que não exija muito esforço computacional na fase de busca, desde que tenha obtido uma solução próxima da ótima.

4.2.1 CONSTRUÇÃO GRASP PROPOSTA

A metaheurística GRASP possui boa parte de seu algoritmo independente do tipo de problema a que este será aplicado para obter soluções. A construção do GRASP proposto consiste também de duas fases (de construção e de busca local), contudo, adotando-se um algoritmo próprio para a primeira fase.

A implementação da função gulosa adaptativa, base da função que gera a solução inicial, compõe o principal aspecto contributivo deste trabalho, na tentativa de obter um melhor escalonamento para o JSSP.

Nesta fase foi utilizado um procedimento denominado Rotulação Dinâmica, cujo funcionamento consiste numa adaptação do algoritmo de Coffman Gramah [04], para se construir um método de identificação das operações, o qual possa contribuir na escolha por uma dada operação, na construção de uma solução aceitável inicial. O referido procedimento avalia os elementos da LC, através de comparativos entre os rótulos atribuídos. O funcionamento de composição de LC é equivalente ao descrito no GRASP tradicional, contudo a atribuição de rótulos confere um recurso para se avaliar as operações. A criação dos rótulos é efetuada para todas as operações apenas uma vez, sendo, por conseguinte, utilizada na geração da LC.

Rotulação Dinâmica: *Regras para Atribuição de Rótulos*

Para cada operação a ser escalonada (elementos da LC) são atribuídos um Rótulo (*Label*) que tem a capacidade de avaliar qualitativamente as operações. Assim, durante a construção da solução inicial, embasada no GRASP, os elementos de maior valor em seus Rótulos terão maior chance de serem escolhidos primeiro, considerando que a preferência no escalonamento da operação-candidata é para aquela de maior Rótulo.

O processo de atribuição de rótulos deve ter início pela operação virtual final, dado que esta não possui sucessores, recebendo, assim, o rótulo de valor zero.

A atribuição do Rótulo segue uma regra básica bi-objetiva, a qual consiste em atribuir o menor Rótulo as operações segundo os seguintes critérios: *i*) A operação que possui o menor conjunto de Rótulos de seus sucessores diretos, observando-os em ordem lexicograficamente[1] invertida, isto é, decrescente; *ii*) Em caso de empate à primeira regra, escolher a operação de maior tempo de processamento (ver Figuras 11-14). Tal sistemática tem o objetivo de forçar a priorização da execução das operações mais rápidas (com menos tempo de processamento). Outro aspecto desta forma de rotulação é o intento de liberar, o quanto antes, as operações que têm mais chance de serem executadas e, por conseguinte, as operações que podem liberar mais processos sucessores. Busca-se, portanto, o melhor aproveitamento possível das máquinas com a redução de "espaços vazios" — tempo em que uma máquina poderia estar sendo utilizada e não está —; havendo algum processo esperando para ser executado, por não estar pronto. Um processo está pronto quando não há (processos) predecessores a serem executados que condicionem, com seu término, o início do mesmo.

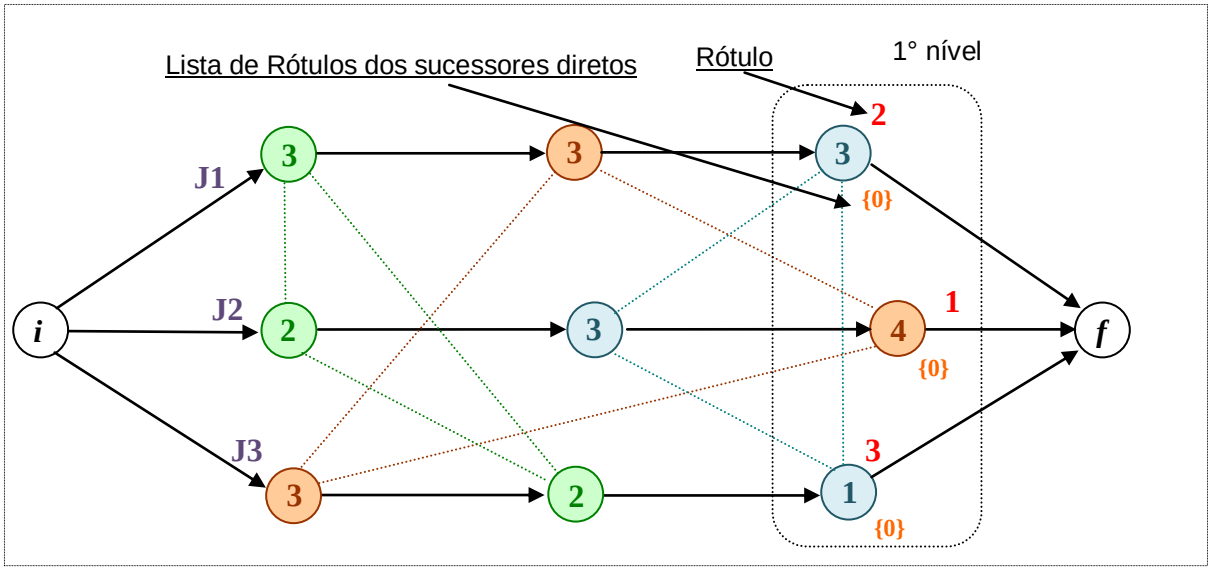

Figura 11: Atribuição de Rótulos no primeiro nível.

Encontram-se no primeiro nível as operações sem sucessores diretos. Neste caso, mesmo havendo a operação virtual final, esta tem tempo nulo, existindo apenas para auxiliar na modelagem do JSSP. Assim, neste nível as listas de rótulos dos sucessores diretos das operações têm o mesmo valor — ou seja, zero —, e a condição de avaliação lexicográfica

[1] Lexicograficamente: Forma gráfica que indica ordem em sua formação. O termo lexicografar, é referente a dicionarizar, ordenar. Tal expressão pode se referir a relação entre assemelhados entre si, por exemplo, palavras, textos ou listas numéricas.

deixa de ter relevância. O que leva à aplicação da segunda regra, a operação com maior tempo de processamento deve obter o menor rótulo. E todos os outros rótulos devem ser aplicados usando-se o mesmo princípio. Havendo, também, coincidência em relação ao valor de tempo de processamento de duas ou mais operações, estas deverão receber o mesmo rótulo. Esta coincidência indica que tais processos estão na mesma condição.

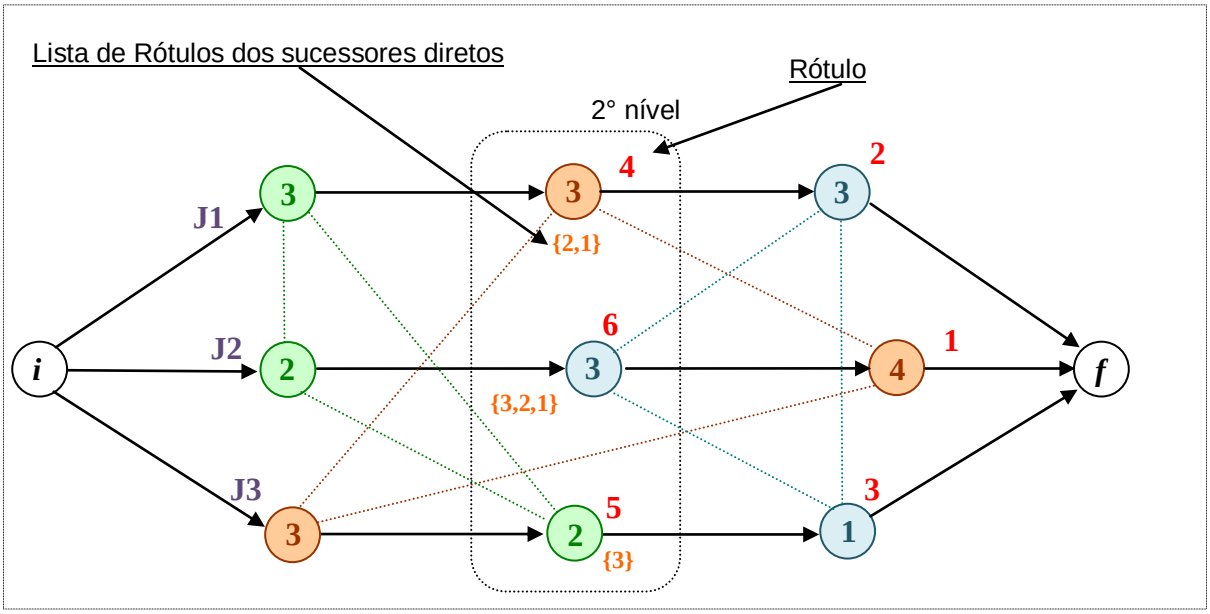

Figura 12: Atribuição de Rótulos no segundo nível.

No segundo nível, observa-se, inicialmente, os rótulos adotados para os sucessores diretos de cada uma das operações deste nível, listando-os em ordem decrescente. Ao se observar cada uma das listas formadas, deve-se atentar para a ordem lexicográfica inversa das listas das respectivas operações - {{2,1}, {3}, {3,2,1}}. Adotando-se para os menores (no caso, {2,1}), os menores rótulos (obedecendo a ordem crescente dos rótulos disponíveis — o subsequente de 3 — último valor utilizado). Em havendo coincidência entre as relações, observa-se o segundo fator de avaliação: o tempo de processamento. Priorizando-se o menor rótulo para os maiores tempos.

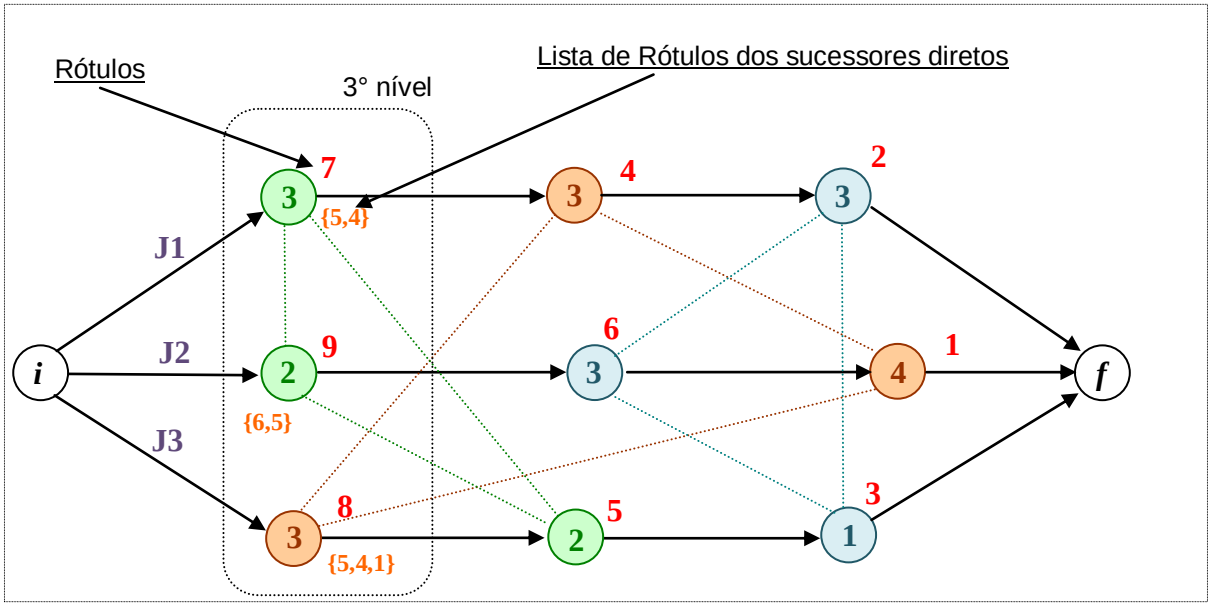

Figura 13: Atribuição de Rótulos no terceiro nível.

O procedimento anterior deverá se repetir até o último nível, representado neste exemplo pelo terceiro nível. Mais uma vez, observa-se a ordem lexicográfica inversa das listas de rótulos dos sucessores diretos das operações relacionadas ao nível - {{5,4}, {5,4,1}, {6,5}}. Adotam-se os menores rótulos para as operações com listas lexicograficamente invertidas menores.

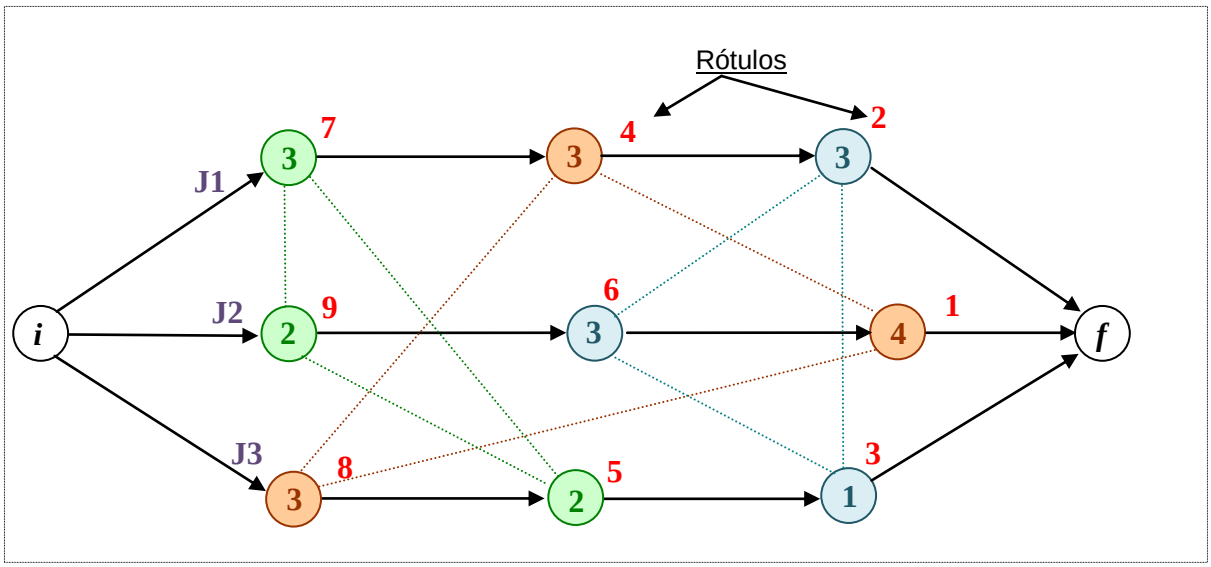

Figura 14: Resultado do algoritmo de Atribuição de Rótulos.

Observa-se assim que a prioridade dos menores rótulos é dada às operações que têm em seu conjunto de sucessores diretos os menores rótulos, pois estes deverão liberar as

operações mais distantes; ao contrário do que ocorre com as operações cujo conjunto de sucessores diretos envolve rótulos maiores, caracterizando a situação onde tal operação permitirá a liberação dos sucessores mais próximos, ou seja, o desimpedimento mais rápido de operações e um escalonamento mais ágil. A atribuição dos rótulos ressaltada nos passos apresentados constitui a base do algoritmo de construção da solução inicial.

Finalizado o procedimento de rotulação das operações, inicia-se o próximo passo na construção da solução, que é a execução da função gulosa adaptativa, a qual irá utilizar tais rótulos para avaliar as operações-candidatas da Lista de Candidatos — LC.

Ainda avaliando o mesmo exemplo, estão prontas para serem executadas as operações com maior rótulo — são operações sem antecessores — e que poderão liberar mais sucessores. Assim, temos no estado de pronto, as operações de rótulos {9}, {8} e {7}. A componente de aleatoriedade poderá escolher tanto uma quanto outra. Contudo, existe uma dependência a ser definida entre a {9} e a {7}. Isto é, ambas são executadas pela mesma máquina, de modo que se faz necessário definir qual a primeira operação a ser executada, observando que as duas têm a mesma chance e que tal escolha deva ser aleatória. Supondo que seja escolhida a {7}, o próximo estado de pronto constará das operações {9} e {8}, as quais podem, inclusive, iniciar ao mesmo tempo, pois são executadas por máquinas diferentes. Observa-se que ao se escolher as operações, conforme se perceba a sequência (de execução das operações) que se forma para cada máquina, tem-se o direcionamento das disjunções no grafo.

Uma vez definido o princípio que rege a composição da LC, o próximo passo é formar a LRC, o que ocorre sempre, considerando-se os rótulos então definidos. Na formação da solução inicial, para cada posição desta, a LC é recriada e tem seus elementos avaliados considerando-se o nível de gulosidade e os tempos de execução de cada um dos elementos para então formar a LRC, como explicado anteriormente. O procedimento continuará até que todas as operações tenham sido escalonadas, obtendo-se, por conseguinte, uma solução inicial.

4.2.2 BUSCA LOCAL

O conceito de vizinhança relativo à proximidade entre soluções pode ser aplicado ao processo de busca, com base numa função objetivo de prioridade, a depender do método heurístico escolhido. Tendo em vista que as soluções encontradas podem ser mapeadas, de

modo que a mínima variação as diferencie, diz-se, ao se analisar uma solução em relação a outra, que são soluções aproximadas (assemelhadas) e que pertencem a mesma vizinhança (o que provavelmente se comprove). Assim, a partir de uma solução, pode-se chegar a uma nova solução através da permutação de um elemento.

Neste trabalho, a fase de busca local se desenvolve com base nas características descritivas de grafos disjuntivos e se constitui na busca por uma solução melhorada na vizinhança da solução construída. Para tanto, o método do caminho crítico é aplicado a fim de se avaliar a referida solução, fazendo uso deste processo para avaliar também as soluções vizinhas. Em relação a construção da vizinhança, esta se baseia no uso de movimentos de vizinhança conhecidos na literatura, tendo-se aplicado vários padrões, entre eles, N1, N4, N5, N6 e a versão estendida de N6 [25]; contudo, o movimento mais promissor foi o de N6. Como visto anteriormente, o bloco crítico — constituinte do caminho crítico — é abordado e explorado através de movimentos em duas direções opostas; isto é, a partir da solução inicial, outras soluções — vizinhas — são geradas através de movimentos de operações centrais em direção as extremidades de um bloco crítico, fazendo com que operações mais internas sejam "exteriorizadas". A cada nova solução gerada, um teste entre esta e a solução corrente — melhor solução encontrada nessa construção — é realizado com base no método do caminho crítico (seção 2.3.3) para avaliá-la.

Estudos anteriores [02, 07, 08, 24, 25] têm comprovado que melhores soluções só podem ser obtidas através de variações no caminho crítico como, por exemplo, pela inversão de apenas uma das arestas disjuntivas pertencentes ao caminho critico, conforme analisa Binato *et al* [07], cabendo, entretanto, a necessária análise da vizinhança. Verifica-se, portanto, que as variações ocorrem em relação às operações que pertencem ao caminho crítico e que fazem parte de uma disjunção, isto porque as conjunções devem ser preservadas por caracterizar o JSSP. E, por fazer parte de uma disjunção, significa que os movimentos de vizinhança são restritos aos blocos críticos.

4.3 BUSCA TABU

A metaheurística Busca Tabu — BT (*Tabu Search*) teve origem a partir de uma solução para problemas de programação inteira; posteriormente foi dada uma descrição do método para uso geral em problemas de otimização combinatória [01]. Tendo sido aplicada ao JSSP, primeiramente por Taillard [22], cuja principal contribuição foi o uso de uma estrutura

de vizinhança usada por Van Laarhoven *et al* [23], pois foi observado que tal algoritmo era mais eficiente para instâncias retangulares.

A Busca Tabu é um procedimento adaptativo auxiliar o qual funciona, basicamente, como um método de busca local que consiste em explorar o espaço de busca, movendo-se de uma solução para outra, permitindo diversificar as soluções encontradas no processo de busca por uma solução melhorada, fazendo uso de um comportamento próprio para superar a otimalidade local e atingir um resultado ótimo ou próximo ao ótimo global. Para tanto, a partir de uma solução inicial, a busca move-se por sua vizinhança a fim de atingir uma solução melhor a cada iteração. Contudo, não aceitando movimentos que levem a soluções já visitadas por constarem em uma lista Tabu, tal lista permanece na memória guardando movimentos visitados por um dado tempo ou por um determinado número de iterações.

Para diversificar as soluções encontradas, a Busca Tabu faz uso de uma estratégia conhecida por diversificação. Essa estratégia, juntamente com a estrutura de memória para armazenar as soluções geradas, é que possibilita escapar dos ótimos locais com o objetivo de redirecionar a pesquisa por regiões ainda não-exploradas: algo que é feito através de um equilíbrio entre intensificação e diversificação. Isso evitando-se que o algoritmo cicle, ou seja, volte a um ótimo local (solução) já visitado, o que é resguardado por uma lista de movimentos proibidos (últimos movimentos realizados e que, partindo de uma dada solução, levam a soluções melhores), trata-se da lista Tabu. Os referidos movimentos são formações que levam uma solução ***s*** a uma solução ***s'***. O procedimento de busca por uma solução melhor pode ser interrompido ao se atingir um dado número de iterações sem melhora ou, ainda, a melhor solução conhecida.

O uso de memória é essencial a essa metaheurística. Primeiro, porque sem ela não seria possível prevenir a ciclagem (*looping*) e, depois, é o uso da memória que permite intensificar a busca em regiões promissoras e, bem como, diversificar a busca para regiões inexploradas. A lista Tabu pode ser usada com finalidades diversas e a forma como ela é gerenciada expressa aplicabilidades diferentes da Busca Tabu, entre elas a Lista Tabu Dinâmica, a exploração por regiões planas, a intensificação, a diversificação e o *Path Relinking*.

É importante observar que a lista Tabu pode ser estática ou dinâmica. A Lista Tabu Dinâmica tem tamanho variável em função de melhoras obtidas no processo iterativo de busca e cujos limites, mínimo e máximo, são definidos no algoritmo em conformidade com o tipo de problema.

4.3.1 BUSCA TABU PROPOSTO

Conforme exposto, a proposta deste trabalho é fazer uso de uma metaheurística híbrida, o que se revela, em relação à Busca Tabu (Figura 16), com sua associação ao GRASP, já que a solução inicial é por este construída.

A fase de busca local através da Busca Tabu teve um processo mais complexo, sendo realizada em três fases de desenvolvimento. Primeiro, a adoção de movimentos de vizinhança; em seguida, o uso de estratégias de intensificação na busca e, por último, a inclusão de uma tática de diversificação. Isso, tomando-se como ponto de partida a solução inicial gerada pelo GRASP.

Em relação aos movimentos de vizinhança, a Busca Tabu também fez uso dos movimentos N6 (já que este mostrou melhores resultados). O uso desta metaheurística foi motivado pela obtenção de melhores resultados com os movimentos de vizinhança N6, comparados aos encontrados na literatura.

Como esclarecido anteriormente, a Busca Tabu faz uso de uma lista de movimentos proibidos, a qual guarda informações de movimentos usados para evitar que estes sejam desfeitos, isto é, usados novamente e levem a vizinhos já visitados. Recriar vizinhos significa gerar ciclos. Tal controle é feito pelo uso da lista Tabu, cujo tamanho — neste trabalho — é variável de acordo com a dimensão da instância. Os parâmetros dimensionais da lista Tabu seguiram um estudo realizado por Zhang *et al* [25], que delimita o tamanho desta lista por um valor aleatório entre valores extremos (L_{min} e L_{max}), definidos com base no número de *jobs* e de máquinas, como mostra o pseudocódigo apresentado a seguir (Figura 15).

Seja

(1) $L_{min} = 10 + n/m$, onde n = *jobs*, m = *machines*

(2) E para L_{max}, tem-se:

Se n <= 2m

$L_{max} = 1{,}4 * L_{min}$

Senão

$L_{max} = 1{,}5 * L_{min}$

Figura 15: Definição do tamanho da lista Tabu.

Como a adoção da lista Tabu pode impedir que soluções ainda não geradas nunca sejam visitadas, foi usado um critério de aspiração na Busca Tabu, o qual permite fazer um movimento mesmo que este tenha sido classificado como proibido (conste na lista Tabu), desde que seja constatado que tal movimento melhora a "melhor" solução conhecida. Desta forma, tal critério tanto evita a ciclagem como permite a renovação da solução com movimentos já utilizados.

```
procedimento BT(s0)
1       enquanto (num_iterações < num_max_iterações) faça
2               m_v = melhor_vizinhança (Lista_tabu);
3               se m_v nao foi encontrado (não foi possível) início
4                       limpa_lista(Lista_tabu);
5                       num_iterações ++;
6               fim;
7               aplica (m_v, s);
8               adiciona_mov_lista (m_v);
9               se f(s) < f(m_s) início
10                      salva_solução;
11                      num_iterações = 0;
12              senão
13                      num_iterações ++;
14              fim;
15      fim;
```

Figura 16: Algoritmo Busca Tabu proposto.

4.3.2 BUSCA TABU OTIMIZADO

Tendo em vista o porte das instâncias utilizadas e a possibilidade de não se atingir a solução ótima, o processo de busca local através da Busca Tabu foi aprimorado com o auxílio de programação inteira. Tal procedimento antecede a terceira fase da busca local descrita anteriormente, ou seja, a diversificação. De modo que, executando-se a etapa de intensificação e não sendo mais possível a obtenção de melhores resultados, inicia-se o método de busca com base em programação inteira, isto é, a exploração de vizinhança via modelo matemático com dimensão reduzida. Caso sejam obtidos melhores resultados deste processo, a busca local é retomada com a intensificação, caso não, processa-se a diversificação que deverá oferecer necessariamente uma solução melhorada para o processo de busca local ou prosseguirá até que se atinja a condição de parada. Onde a condição de

parada utilizada é um parâmetro de entrada que estabelece o número de iterações sem melhora [ver Figura 17].

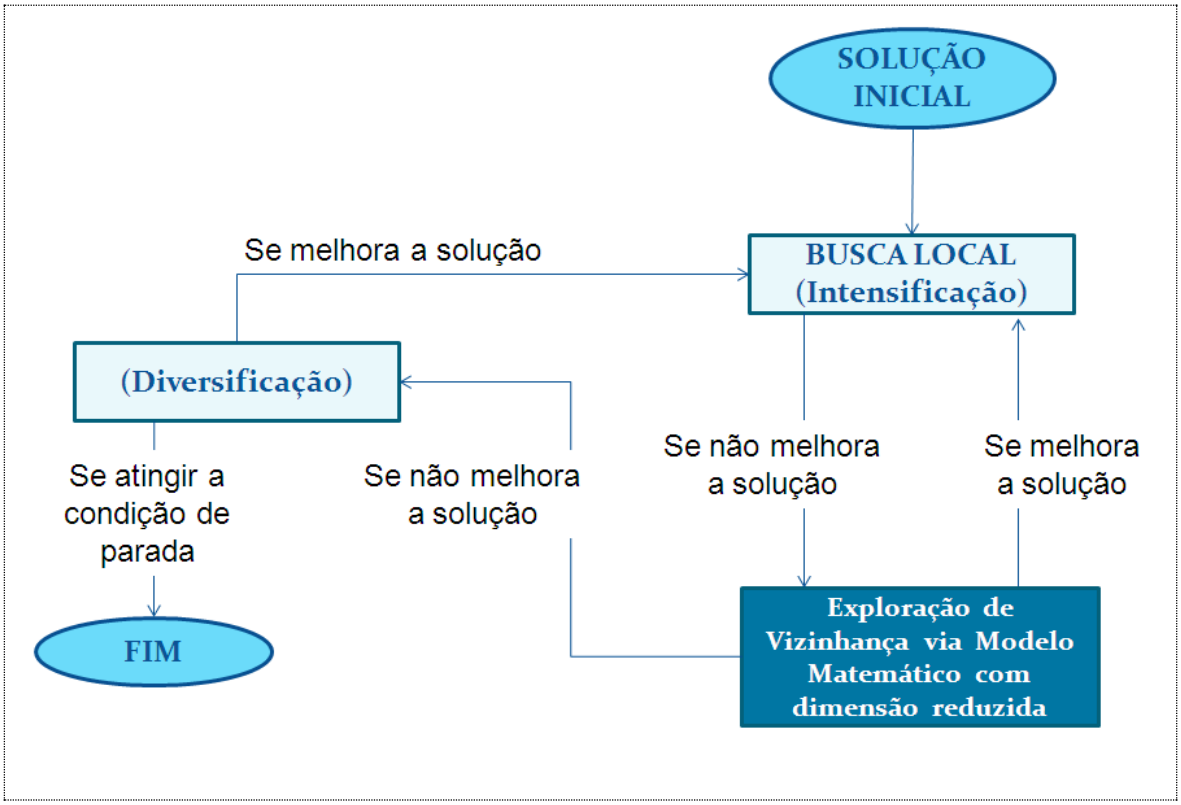

Figura 17: Busca Tabu Otimizada.

Tal método consiste em realizar um processo de busca por soluções passando por um processo de análise combinatória, o qual toma como variáveis todas as operações que se relacionem com o caminho crítico direta ou indiretamente. Isto é, são consideradas pelo método as operações pertencentes ao caminho crítico ou nele incidentes. As demais operações não são relacionadas como sujeitas a re-arrumação. Considerando que a re-arrumação se constitui na base do movimento de vizinhança e pode ser traduzido como um direcionamento diferenciado para as disjunções. Esta fase é auxiliada pelo CPLEX — ferramenta da IBM capaz de realizar a otimização matemática sobre um amplo conjunto de instâncias.

Com base no modelo matemático, exposto no capítulo 3, as permutações realizadas entre as operações, nos movimentos de vizinhança, se realizam apenas em relação as disjunções o que é definido pela variável de decisão x_{ij}. Assim, no momento da diversificação, é efetuada a análise da solução inicial, ou da melhor solução encontrada a partir desta, com o método do caminho crítico para se definir as disjunções pertencentes a este. O otimizador atua em cima da variável x_{ij} gerando uma matriz quadrada cuja ordem é igual ao número total de

operações. Para toda disjunção não pertencente ao Caminho Crítico, ou que não esteja ligada a uma operação crítica pertencente a uma disjunção crítica, são fixadas as variáveis da matriz X_{ij} correspondentes. Tal procedimento acaba por acrescentar novas restrições ao problema (a fixação dos valores de arestas) e tem a finalidade de diminuir o escopo do problema; o que garante uma vizinhança otimizada, não uma solução ótima.

Ao se observar, na Figura 18, o grafo disjuntivo no qual foram utilizados índices ilustrativos — para se identificar as operações em relação à variável x_{ij} —, tem-se a definição do problema (caracterizada por arestas disjuntivas sem direcionamento). E na sequência, na Figura 19, a representação da solução, inclusive com a definição do Caminho Crítico.

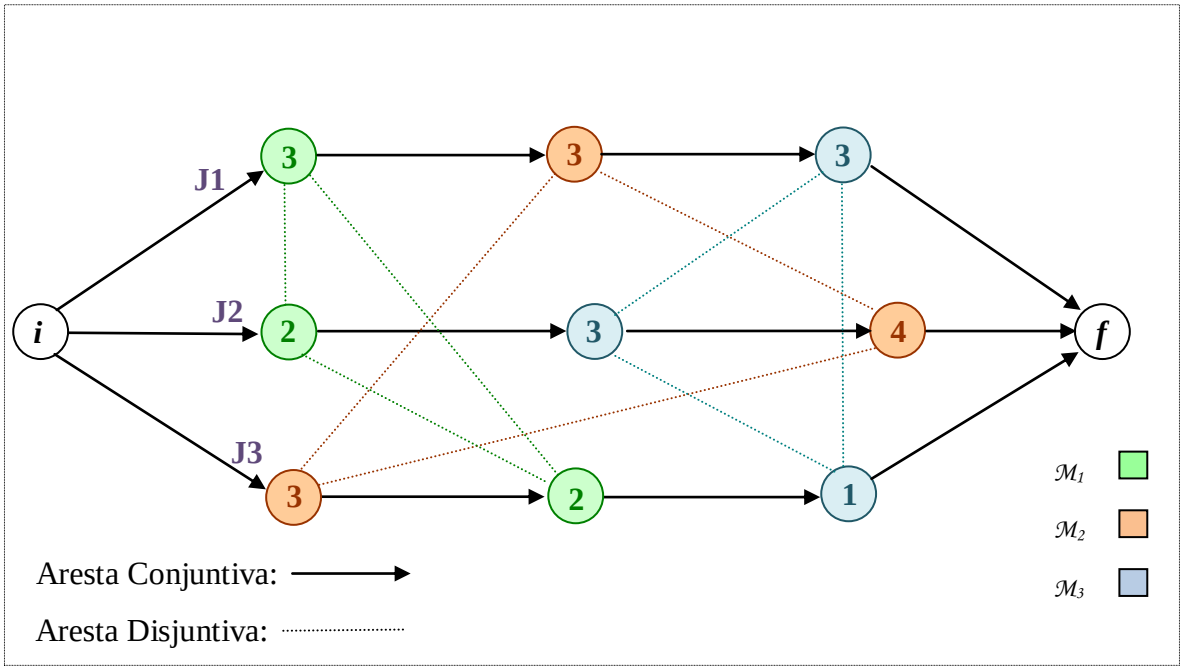

Figura 18: Representação do problema na otimização.

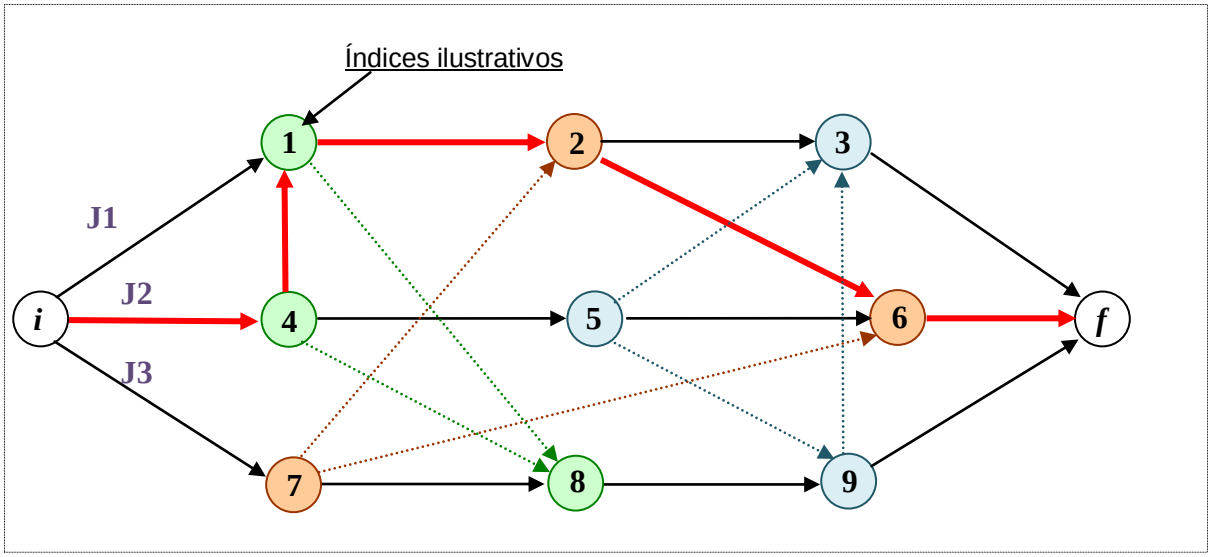

Figura 19: Representação de uma solução na otimização.

Como a variável x_{ij} denota a existência da aresta direcionada no sentido indicado por seus índices, isto é, x_{ij} igual 1 indica que a operação i é realizada primeiro que a operação j e igual a 0, caso contrário. A variável x_{ij} está definida, como já demonstrado, por:

$\forall$ [i, j] $\in$ D, $x_{ij} \in \{0, 1\}$

Assim, a matriz X_{ij}, cujo domínio é definido pelos valores 1 (expressa o indicativo positivo para sentido da disjunção representado pelos índices) e 0 (caso contrário), deverá estabelecer os respectivos valores de x_{ij} para as arestas disjuntivas; os quais passarão a ficar em aberto para as arestas pertinentes as disjunções críticas ou, que sejam a estas ligados como aresta disjuntiva posterior ou antecessora.

Exemplos de x_{ij}:

$x_{14} = 0$;

$x_{41} = 1$;

$x_{48} = 1$;

...

Para efeito de esclarecimento, os referidos conceitos podem ser representados pela seguinte matriz (Tabela 2), onde o não preenchimento de algumas células indica apenas a inexistência de disjunção entre as referidas operações.

Tabela 2: Representação da matriz X_{ij}

i \	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
1	j			0				1	
2						1	0		
3					0				0
4	1							1	
5			1						1
6		0					0		
7		1				1			
8	0			0					
9			1		0				

Para o processo de otimização, a busca por soluções melhores é realizada utilizando-se o Método do Caminho Crítico para avaliar a solução corrente e, por conseguinte, os movimentos que possam proporcionar melhores vizinhanças, uma vez que estudos encontrados na literatura [25] demonstraram que as mudanças mais propícias à otimização consistem em reduzir o tempo de processamento das operações que fazem parte do Caminho Crítico. Assim, para o melhor entendimento do processo de Exploração de Vizinhança via Modelo Matemático adotado neste trabalho, pode-se utilizar a matriz (apresentada anteriormente) com os valores de x_{ij} com o indicativo do possível sentido das arestas disjuntivas; contudo, como a possibilidade de melhoria está diretamente relacionada às operações que constituem o Caminho Crítico, o procedimento mantém os valores encontrados para as arestas disjuntivas, em relação a uma dada solução, e deixa em aberto a possibilidade de mudança dos valores de x_{ij} (acima destacados) das arestas disjuntivas críticas ou que com elas se relacione diretamente.

Desta forma, no exemplo usado, as células marcadas em azul assumiram, na solução encontrada, os valores indicados, mas terão seus valores em aberto e poderão ser modificados (podem assumir 0 ou 1) por estarem ligadas a disjunções críticas. Assim, a movimentação de vizinhança fica restrita a apenas algumas arestas disjuntivas e é eficaz, pois busca melhorias em torno do Caminho Crítico.

Após a realização do procedimento de diversificação — no qual se busca por uma solução otimizada mais distante das soluções preliminarmente obtidas e, portanto, permite um grande salto na escala das soluções —, se houver melhora da solução, o procedimento de intensificação é novamente aplicado com vistas a explorar a nova região de vizinhança alcançada.

Os procedimentos de intensificação, otimização via modelo matemático e diversificação são executados até que um limite, previamente definido, seja alcançado. Tal

limite pode ser temporal ou objetivo, isto é, uma variável que define o tempo de busca ou até que todas as possibilidades sejam avaliadas. Como a segunda opção (limite objetivo) exigiria muito tempo para executar o conjunto de instâncias de teste, a primeira opção (limite temporal) foi selecionada com o tempo mínimo de dois minutos para cada execução do procedimento do modelo matemático, juntamente com o valor de entrada que define o número máximo de iterações sem melhora na fase de diversificação, que representa, efetivamente, o meio de saída e pode vir a ser executado mais de uma vez para cada instância.

5. RESULTADOS COMPUTACIONAIS

5.1 AMBIENTE DE EXECUÇÃO

Como exposto anteriormente, o problema tratado é representado através de uma instância numérica que pode ser expressa numa matriz genérica, cujas definições denotam as características principais do problema. Assim, a primeira linha define as dimensões da matriz (*n* x *m*), onde o "*n*" indica o número de *jobs* e o "*m*" indica o número de máquinas. As demais linhas representam o escalonamento de cada *job*, com as colunas ímpares representando o número da máquina em que cada operação de um *job* deve ser processada e as colunas pares representando o tempo de processamento da respectiva operação em cada máquina. Na Figura 20, a variável "M" ilustra a máquina e a variável "T" ilustra o tempo de processamento.

n m

M_{11} T_{11} M_{12} T_{12} … M_{1m} T_{1m}
M_{21} T_{21} M_{22} T_{22} … M_{2m} T_{2m}
… … …
M_{n1} T_{n1} M_{n2} T_{n2} … M_{nm} T_{nm}

Figura 20: Estrutura Genérica de uma instância do JSSP.

As instâncias testadas foram obtidas no site OR-LIBRARY (Operations Research Library) < http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt>, um repositório de várias instâncias de problemas de otimização combinatória. Os testes foram realizados para as instâncias TA(1–30), LA(1–40), ABZ(5–9), ORB(1–10) e YN(1–4), das quais apenas o primeiro conjunto (TA(1–30)) foi gerado através de um algoritmo disponibilizado no site< http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop2.txt>, e as demais foram tomadas tal como descrito. Tais instâncias foram escolhidas por constarem na literatura da área.

Nesta seção, as instâncias são referenciadas pelo número de *jobs* e de máquinas, bem como pelo paralelo entre o BKS (*Best Known Solution*) e o melhor resultado encontrado. O comparativo entre os dois resultados gera o GAP (erro relativo), que é também referenciado juntamente com o tempo de execução, para obtenção deste resultado, em segundos.

A seguir serão apresentados os resultados computacionais obtidos com o hibridismo das metaheurísticas GRASP e Busca Tabu proposta para o *JSSP*. Os testes foram realizados inúmeras vezes e em diversas circunstâncias para cada instância com o fim de se verificar a variação dos resultados de cada execução.

A solução proposta foi implementada utilizando a linguagem C++. Todos os experimentos computacionais foram realizados em computadores do LAPORTE, com a seguinte configuração: AMD ATHLON de 2.4 Ghz, processador Intel Core(TM)2 Quad, 2 GB de memória RAM, 500 GB de HD e sistema operacional Ubuntu.

Como o objetivo foi testar a construção da solução inicial, foram feitas duas associações entre o GRASP e a Busca Tabu. Na primeira implementação, os algoritmos utilizados com base nas metaheurísticas tiveram uma formulação básica, tanto para uma quanto para outra metaheurística. Na segunda implementação, a formulação do GRASP foi mantida e o algoritmo da Busca Tabu foi otimizado. A razão de se manter o GRASP resume-se na pretensão de se analisar a influência da formulação da solução inicial, tanto pelo tempo de execução quanto pela qualidade das soluções obtidas. Enquanto que o uso da Busca Tabu foi motivada pela capacidade expansão de vizinhança, bem como, pela possibilidade de associação com a exploração via modelo matemático.

5.2 RESULTADOS OBTIDOS

Para avaliar a evolução da proposta deste trabalho, os testes foram realizados para um algoritmo híbrido básico e depois para um algoritmo híbrido otimizado, com a finalidade de se estabelecer um paralelo entre as duas propostas.

Tais testes foram efetuados várias vezes para cada instância com variação no número de repetições sem melhora para a execução do processo de busca local (realizado pela Busca Tabu). Foram estabelecidos, por fim, valores médios tanto para uma como para outra definição dos testes, já que fazem singular distinção para cada grupo de instâncias.

O perfil das instâncias utilizadas, apresentadas a seguir, é descrito essencialmente por suas dimensões, que assinalam também os números de *Jobs* e de máquinas, respectivamente. Havendo para cada perfil, mais de uma instância, sendo os perfis:

- TA (15 x 15), (20 x 15) e (20 x 20).
- LA (10 x 5), (15 x 5), (20 x 5), (10 x 10), (15 x 10), (20 x 10), (30 x 10), (15 x 15).

- ABZ (10 x 10), (20 x 15).
- ORB (10 x 10).
- YN (20 x 20).

Assim, cada uma das instâncias foi executada dez vezes e em dois ciclos, para cada implementação, a fim de se encontrar uma solução que melhore o GAP. Extraindo-se, para cada ciclo, o melhor e o pior resultado obtidos nos experimentos, para compará-los com o melhor resultado conhecido na literatura (BKS). No primeiro ciclo, após se gerar a solução inicial, executa-se a Busca Tabu até que se atinja a condição de parada de se efetuar 60000 iterações sem melhora. No segundo, como é uma nova execução, gera-se novamente a solução inicial e então se executa a Busca Tabu por até 80000 iterações sem melhora. Além destes valores, a média dos resultados de cada ciclo também foi calculada para efeito comparativo, em relação ao BKS.

A execução das instâncias por várias vezes teve o objetivo de analisar a variação dos valores obtidos, como será visto posteriormente. Já a execução das instâncias em dois ciclos, modificando-se o número de repetições de busca sem melhora, deve-se à necessidade de se analisar o nível de melhora da solução. Para tanto, serão expostos resultados de todas as instâncias utilizadas.

Os resultados estão apresentados nas Tabelas (3-12), em que constam, em paralelo, os resultados dos testes realizados divididos em dois grupos, o do algoritmo híbrido GRASP e Busca Tabu e o do algoritmo híbrido GRASP e Busca Tabu Otimizado. Onde, na primeira coluna, pode-se encontrar a denominação de cada instância com suas respectivas dimensões. Na segunda coluna, tem-se o *BKS* que indica o melhor valor de avaliação encontrado na literatura em relação à instância referenciada na primeira coluna. Daí, inicia-se a divisão grupal, para a qual apresentam-se os resultados obtidos com cada implementação proposta neste trabalho, respectivamente, o melhor valor, o pior valor, o valor médio, os respectivos *GAP* e tempo de execução do melhor resultado obtido em segundos. O GAP relaciona o BKS e o melhor resultado alcançado nos experimentos para cada instância de problema, através de cálculo centesimal (GAP = 100 * ((melhor_resultado_encontrado - BKS) / BKS)).

Para o grupo TA:

Repetições sem Melhora=60000

Tabela 3: Grupo TA (01 a 30) - execução do algoritmo com até 60000 repetições sem melhora.

		GRASP+BUSCA TABU					GRASP+BUSCA TABU OTIMIZADO				
Nome (JxM)	**BKS**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**
ta01(15x15)	1231	1248	1259	1253,40	1,38	254	1233	1257	1249,70	0,16	54
ta02(15x15)	1244	1260	1268	1265,00	1,29	253	1253	1269	1263,70	0,72	107
ta03(15x15)	1218	1224	1229	1227,00	0,49	295	1223	1248	1230,10	0,41	61
ta04(15x15)	1175	1189	1206	1197,80	1,19	280	1185	1207	1192,60	0,85	61
ta05(15x15)	1224	1235	1247	1239,80	0,90	411	1242	1251	1246,40	1,47	58
ta06(15x15)	1238	1245	1260	1255,00	0,57	297	1249	1266	1257,00	0,89	91
ta07(15x15)	1227	1228	1245	1232,80	0,08	307	1228	1249	1235,20	0,08	68
ta08(15x15)	1217	1229	1241	1233,40	0,99	288	1226	1236	1231,00	0,74	55
ta09(15x15)	1274	1291	1312	1302,20	**1,33**	**272**	1296	1313	1304,30	**1,73**	**80**
ta10(15x15)	1241	1263	1287	1274,20	**1,77**	**330**	1268	1282	1276,30	**2,18**	**53**
ta11(20x15)	1359	1373	1402	1390,80	**1,03**	**728**	1376	1562	1406,70	**1,25**	**92**
ta12(20x15)	1367	1385	1394	1390,60	1,32	467	1381	1402	1390,60	1,02	222
ta13(20x15)	1342	1375	1411	1385,00	2,46	436	1371	1389	1378,50	2,16	120
ta14(20x15)	1345	1347	1355	1351,20	0,15	497	1345	1359	1352,50	0,00	818
ta15(20x15)	1339	1370	1391	1377,60	2,32	855	1360	1399	1374,30	1,57	109
ta16(20x15)	1360	1375	1423	1400,20	1,10	495	1389	1421	1398,90	2,13	118
ta17(20x15)	1462	1478	1547	1505,40	1,09	634	1482	1556	1503,10	1,37	93
ta18(20x15)	1396	1438	1448	1441,80	3,01	619	1429	1449	1441,00	2,36	111
ta19(20x15)	1335	1362	1413	1382,20	2,02	611	1368	1388	1375,50	2,47	137
ta20(20x15)	1350	1383	1394	1387,00	2,44	493	1368	1397	1382,20	1,33	95
ta21(20x20)	1644	1668	1689	1676,20	1,46	1041	1661	1697	1677,30	1,03	165
ta22(20x20)	1600	1621	1638	1630,80	1,31	932	1612	1643	1631,30	0,75	279
ta23(20x20)	1557	1581	1594	1587,00	1,54	1251	1579	1621	1591,90	1,41	128
ta24(20x20)	1646	1661	1688	1673,60	0,91	938	1664	1694	1679,00	1,09	168
ta25(20x20)	1595	1623	1645	1636,00	1,76	842	1620	1654	1634,70	1,57	247
ta26(20x20)	1645	1678	1722	1694,40	2,01	686	1676	1701	1687,80	1,88	181
ta27(20x20)	1680	1708	1724	1716,20	1,67	889	1699	1736	1714,30	1,13	149
ta28(20x20)	1603	1622	1642	1634,60	1,19	839	1623	1633	1628,80	1,25	305
ta29(20x20)	1625	1641	1653	1645,80	0,98	981	1639	1680	1647,90	0,86	131
ta30(20x20)	1584	1597	1645	1626,00	0,82	812	1618	1641	1626,80	2,15	123
Média				**1433,77**	**1,35**	**601,10**			**1433,65**	**1,27**	**149,30**

Repetições sem Melhora=80000

Tabela 4: Grupo TA (01 a 30) - execução do algoritmo com até 80000 repetições sem melhora.

		GRASP+BUSCA TABU					GRASP+BUSCA TABU OTIMIZADO				
Nome (JxM)	**BKS**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**
ta01(15x15)	1231	1242	1259	1250,60	0,89	311	1248	1251	1249,20	1,38	90
ta02(15x15)	1244	1253	1266	1260,00	0,72	357	1250	1269	1262,10	0,48	102
ta03(15x15)	1218	1223	1228	1225,60	0,41	347	1222	1235	1226,40	0,33	88
ta04(15x15)	1175	1183	1200	1190,20	0,68	328	1181	1194	1189,30	0,51	76
ta05(15x15)	1224	1243	1247	1244,80	1,55	375	1233	1246	1241,30	0,74	83
ta06(15x15)	1238	1248	1262	1255,40	0,81	429	1246	1263	1254,00	0,65	132
ta07(15x15)	1227	1228	1246	1235,60	0,08	360	1228	1250	1238,30	0,08	204
ta08(15x15)	1217	1226	1232	1229,40	0,74	422	1220	1246	1230,10	0,25	138
ta09(15x15)	1274	1288	1314	1303,80	**1,10**	**396**	1291	1316	1301,10	**1,33**	**75**
ta10(15x15)	1241	1269	1284	1275,40	2,26	345	1267	1281	1273,90	2,10	66
ta11(20x15)	1359	1373	1398	1385,80	**1,03**	**939**	1381	1479	1407,60	**1,62**	**132**
ta12(20x15)	1367	1388	1398	1392,20	1,54	787	1380	1419	1390,30	0,95	131
ta13(20x15)	1342	1368	1386	1375,80	1,94	768	1362	1393	1377,90	1,49	117
ta14(20x15)	1345	1350	1356	1353,80	**0,37**	**579**	1345	1362	1353,00	**0,00**	**967**
ta15(20x15)	1339	1366	1377	1372,00	2,02	950	1363	1504	1385,80	1,79	140
ta16(20x15)	1360	1377	1392	1385,80	1,25	1056	1378	1406	1391,20	1,32	197
ta17(20x15)	1462	1490	1560	1521,60	1,92	601	1482	1535	1507,10	1,37	125
ta18(20x15)	1396	1441	1450	1446,00	3,22	853	1417	1531	1451,00	1,50	145
ta19(20x15)	1335	1360	1393	1374,80	1,87	853	1353	1380	1368,50	1,35	151
ta20(20x15)	1350	1372	1386	1378,20	1,63	780	1367	1388	1380,30	1,26	211
ta21(20x20)	1644	1655	1687	1672,00	0,67	974	1663	1690	1674,80	1,16	365
ta22(20x20)	1600	1622	1636	1630,20	1,38	898	1623	1642	1634,40	1,44	413
ta23(20x20)	1557	1577	1629	1593,40	1,28	781	1572	1597	1584,20	0,96	302
ta24(20x20)	1646	1678	1688	1681,20	1,94	1159	1661	1689	1675,60	0,91	430
ta25(20x20)	1595	1621	1640	1628,80	1,63	1196	1601	1647	1633,70	0,38	214
ta26(20x20)	1645	1678	1721	1691,20	2,01	1213	1680	1797	1701,50	2,13	144
ta27(20x20)	1680	1707	1714	1710,00	1,61	1219	1705	1729	1715,70	1,49	178
ta28(20x20)	1603	1624	1636	1630,20	1,31	881	1622	1638	1630,70	1,19	489
ta29(20x20)	1625	1637	1654	1646,40	0,74	847	1639	1646	1642,70	0,86	703
ta30(20x20)	1584	1615	1628	1618,00	1,96	983	1607	1636	1623,10	1,45	186
Média				**1431,94**	**1,35**	**732,9**			**1433,16**	**1,08**	**226,47**

Os valores experimentais alcançados pelas instâncias do grupo TA expressam a grande dificuldade em atingir o GAP zero. Isto ocorre devido à dimensão das instâncias que, sendo maiores, apresentam maior complexidade combinatória. Contudo, o tempo de processamento da Busca Tabu Otimizada, em relação a Busca Tabu Básica, pôde ser reduzido ao tempo em que se reduzia também o GAP obtido para cerca de cinquenta e cinco por cento das instâncias na execução do algoritmo com o limite de 60000 repetições; mas, essa porcentagem sofre um aumento significativo com o aumento de repetições, chegando a setenta por cento.

Houve casos em que o GAP obtido com a otimização foi sensivelmente pior, tanto na primeira como na segunda execução, porém, não foi possível relacionar uma regra para tal ocorrência. Observando a Tabela 4, encontra-se situações de piora na quantidade das soluções geradas pelo GRASP + Busca Tabu Otimizado, como no caso das instâncias TA09 e TA11, onde há um aumento no GAP de 1,10 para 1,33 e de 1,03 para 1,62, respectivamente. No

entanto, observando as médias gerais percebe-se ainda que a obtenção do GAP nulo não ficou descartada, como ocorreu para a instância TA14. Os valores médios encontrados nas duas tabelas são aproximados, mas o GAP médio do GRASP + Busca Tabu Otimizada alcança um ganho de trinta por cento.

Para o grupo LA:

Repetições sem Melhora=60000

Tabela 5: Grupo LA (01 a 40) - execução do algoritmo com até 60000 repetições sem melhora.

		GRASP+BUSCA TABU					GRASP+BUSCA TABU OTIMIZADO				
Nome (JxM)	**BKS**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**
la01(10x5)	666	666	666	666,00	0,00	35	666	666	666,00	0,00	8
la02(10x5)	655	655	655	655,00	0,00	37	655	655	655,00	0,00	9
la03(10x5)	597	597	597	597,00	0,00	41	597	597	597,00	0,00	9
la04(10x5)	590	590	590	590,00	0,00	38	590	590	590,00	0,00	7
la05(10x5)	593	593	593	593,00	0,00	39	593	593	593,00	0,00	7
la06(15x5)	926	926	926	926,00	0,00	87	926	926	926,00	0,00	16
la07(15x5)	890	890	890	890,00	0,00	88	890	890	890,00	0,00	377
la09(15x5)	951	951	951	951,00	**0,00**	**92**	951	951	951,00	**0,00**	**16**
la10(15x5)	958	958	958	958,00	0,00	87	958	958	958,00	0,00	15
la11(20x5)	1222	1222	1222	1222,00	**0,00**	**154**	1222	1222	1222,00	**0,00**	**29**
la12(20x5)	1039	1039	1039	1039,00	0,00	162	1039	1039	1039,00	0,00	29
la13(20x5)	1150	1150	1150	1150,00	0,00	156	1150	1150	1150,00	0,00	27
la14(20x5)	1292	1292	1292	1292,00	0,00	155	1292	1292	1292,00	0,00	29
la15(20x5)	1207	1207	1207	1207,00	0,00	158	1207	1207	1207,00	0,00	388
la16(10x10)	945	946	959	949,80	0,11	87	946	959	950,00	0,11	17
la17(10x10)	784	784	785	784,20	0,00	79	784	785	784,10	0,00	16
la18(10x10)	848	848	853	850,20	0,00	80	848	853	849,00	0,00	15
la19(10x10)	842	842	848	845,20	0,00	74	842	852	848,00	0,00	15
la20(10x10)	902	902	902	902,00	0,00	74	902	909	902,70	0,00	15
la21(15x10)	1046	1049	1053	1051,20	0,29	166	1047	1055	1050,20	0,10	423
la22(15x10)	927	931	935	933,40	0,43	195	927	940	934,80	0,00	235
la23(15x10)	1032	1032	1032	1032,00	0,00	151	1032	1032	1032,00	0,00	442
la24(15x10)	935	944	948	945,40	0,96	151	939	946	942,80	0,43	31
la25(15x10)	977	983	984	983,80	0,61	178	979	995	983,10	0,20	60
la26(20x10)	1218	1218	1218	1218,00	0,00	279	1218	1218	1218,00	0,00	654
la27(20x10)	1235	1241	1255	1250,20	0,49	297	1240	1257	1250,00	0,40	300
la28(20x10)	1216	1216	1219	1216,80	0,00	327	1216	1220	1216,40	0,00	1061
la29(20x10)	1152	1185	1228	1205,00	2,86	347	1177	1298	1198,80	2,17	68
la30(20x10)	1355	1355	1355	1355,00	0,00	283	1355	1355	1355,00	0,00	777
la31(30x10)	1784	1784	1784	1784,00	0,00	562	1784	1784	1784,00	0,00	235
la32(30x10)	1850	1850	1850	1850,00	0,00	572	1850	1850	1850,00	0,00	471
la33(30x10)	1719	1719	1719	1719,00	0,00	627	1719	1719	1719,00	0,00	366
la34(30x10)	1721	1721	1721	1721,00	0,00	604	1721	1721	1721,00	0,00	1190
la35(30x10)	1888	1888	1888	1888,00	0,00	618	1888	1888	1888,00	0,00	1080
la36(15x15)	1268	1278	1288	1282,00	0,79	249	1269	1287	1277,00	0,08	50
la37(15x15)	1397	1415	1428	1418,80	1,29	328	1410	1426	1418,10	0,93	221
la38(15x15)	1196	1207	1223	1213,40	0,92	277	1202	1219	1209,30	0,50	87
la39(15x15)	1233	1245	1250	1247,80	0,97	282	1243	1248	1247,20	0,81	67
la40(15x15)	1222	1228	1235	1232,20	0,49	335	1229	1239	1233,60	0,57	109
Média				**1118,32**	**0,26**	**219,26**			**1117,90**	**0,16**	**230,03**

Repetições sem Melhora=80000

Tabela 6: Grupo LA (01 a 40) - execução do algoritmo com até 80000 repetições sem melhora.

		GRASP+BUSCA TABU					GRASP+BUSCA TABU OTIMIZADO				
Nome (JxM)	**BKS**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**
la01(10x5)	666	666	666	666,00	0,00	47	666	666	666,00	0,00	9
la02(10x5)	655	655	656	655,20	0,00	52	655	655	655,00	0,00	11
la03(10x5)	597	597	597	597,00	0,00	57	597	597	597,00	0,00	11
la04(10x5)	590	590	590	590,00	0,00	49	590	590	590,00	0,00	10
la05(10x5)	593	593	593	593,00	0,00	51	593	593	593,00	0,00	10
la06(15x5)	926	926	926	926,00	0,00	115	926	926	926,00	0,00	21
la07(15x5)	890	890	890	890,00	0,00	114	890	890	890,00	0,00	142
la09(15x5)	951	951	951	951,00	0,00	117	951	951	951,00	0,00	21
la10(15x5)	958	958	958	958,00	0,00	122	958	958	958,00	0,00	22
la11(20x5)	1222	1222	1222	1222,00	0,00	202	1222	1222	1222,00	0,00	38
la12(20x5)	1039	1039	1039	1039,00	0,00	201	1039	1039	1039,00	0,00	28
la13(20x5)	1150	1150	1150	1150,00	0,00	212	1150	1150	1150,00	0,00	35
la14(20x5)	1292	1292	1292	1292,00	0,00	208	1292	1292	1292,00	0,00	36
la15(20x5)	1207	1207	1207	1207,00	0,00	210	1207	1207	1207,00	0,00	278
la16(10x10)	945	945	952	947,00	0,00	132	945	959	950,90	0,00	23
la17(10x10)	784	784	784	784,00	0,00	102	784	785	784,30	0,00	19
la18(10x10)	848	848	861	850,80	0,00	101	848	849	848,10	0,00	21
la19(10x10)	842	846	850	847,80	0,48	100	843	850	847,10	0,12	19
la20(10x10)	902	902	902	902,00	0,00	100	902	909	902,70	0,00	21
la21(15x10)	1046	1049	1055	1052,20	0,29	230	1047	1056	1052,30	0,10	552
la22(15x10)	927	931	937	934,20	**0,43**	219	932	939	935,30	**0,54**	431
la23(15x10)	1032	1032	1032	1032,00	0,00	203	1032	1032	1032,00	0,00	164
la24(15x10)	935	942	944	943,00	0,75	233	939	946	942,30	0,43	45
la25(15x10)	977	978	984	982,60	0,10	223	978	984	982,80	0,10	64
la26(20x10)	1218	1218	1218	1218,00	0,00	369	1218	1218	1218,00	0,00	677
la27(20x10)	1235	1249	1256	1251,60	1,13	483	1241	1254	1248,00	0,49	679
la28(20x10)	1216	1216	1217	1216,20	0,00	400	1216	1219	1216,50	0,00	814
la29(20x10)	1152	1179	1257	1210,40	2,34	487	1179	1228	1198,10	2,34	79
la30(20x10)	1355	1355	1355	1355,00	0,00	377	1355	1355	1355,00	0,00	794
la31(30x10)	1784	1784	1784	1784,00	0,00	743	1784	1784	1784,00	0,00	390
la32(30x10)	1850	1850	1850	1850,00	0,00	741	1850	1850	1850,00	0,00	402
la33(30x10)	1719	1719	1719	1719,00	0,00	834	1719	1719	1719,00	0,00	512
la34(30x10)	1721	1721	1721	1721,00	0,00	758	1721	1721	1721,00	0,00	1352
la35(30x10)	1888	1888	1888	1888,00	0,00	801	1888	1888	1888,00	0,00	1000
la36(15x15)	1268	1276	1291	1281,20	0,63	327	1270	1284	1277,50	0,16	82
la37(15x15)	1397	1420	1428	1422,60	1,65	360	1407	1426	1418,30	0,72	310
la38(15x15)	1196	1213	1219	1215,80	1,42	372	1202	1214	1208,80	0,50	81
la39(15x15)	1233	1243	1251	1247,60	0,81	363	1240	1250	1247,20	0,57	68
la40(15x15)	1222	1229	1234	1232,00	0,57	357	1228	1235	1232,80	0,49	79
Média				**1118,57**	**0,27**	**286,46**			**1117,85**	**0,17**	**239,74**

O comportamento geral das instâncias do grupo LA, em relação aos algoritmos testados, é de atingir rapidamente resultados próximos do ótimo, o que é verificado pela consecução do GAP nulo ou próximo disto. Tal característica deve-se ao perfil das referidas instâncias. Observa-se que nestas instâncias retangulares obtêm-se mais facilmente o GAP nulo do que nas instâncias quadradas, observando-se que quanto maior o número de máquinas, maior a dificuldade de resolução do problema.

Ao examinar as tabelas anteriores (Tabelas 5 e 6), percebe-se a redução do tempo de execução para obtenção do GAP zero, com a utilização da Busca Tabu Otimizada para cerca de cinquenta por cento das instâncias. O GAP nulo foi alcançado já na execução da Busca Tabu simplificada, para a qual a geração da solução inicial é significativamente relevante, observando-se o tempo de obtenção do GAP nulo nesta fase para muitas instâncias. Por outro lado, percebe-se também, a obtenção do GAP nulo ou, ainda, do GAP menor, com um relevante aumento no tempo de execução da Busca Tabu Otimizada em relação à Busca Tabu Básica, como ocorre para as instâncias LA07, LA15, LA34 e LA35. Tal ocorrência se dá, provavelmente, por se tratar de um ótimo local que não pode ser ultrapassado no período de tempo estipulado para a execução dos algoritmos.

Assim, mesmo para instâncias mais complexas, ainda que o GAP nulo não tenha sido atingido, este pode ser reduzido com a otimização, bem como o tempo de execução; o que se verifica nos resultados obtidos pelas instâncias LA36-LA40. Ao se considerar simplesmente o comparativo entre os dois algoritmos, pode-se confirmar uma melhoria em relação ao GAP obtido no algoritmo otimizado, seja pela redução do valor do GAP, seja pela redução do tempo de execução para obtenção de um GAP equivalente. Fato esse que se constitui em exceção apenas para um caso isolado, a instância LA22; que, na primeira execução, alcançou o GAP nulo com a otimização e, na segunda, foi a única instância a não sofrer melhorias, dada a obtenção do GAP e do tempo de execução sensivelmente maiores, com a otimização. Tal comportamento possivelmente possa ser explicado pelo fato de se ter iniciado o processo com uma solução próxima da ótima, a qual representa um ótimo local explorado por dois caminhos distintos: um, próximo da solução ótima e, outro, distante o suficiente para não ser encontrado mesmo num número maior de repetições sem melhora.

Por fim, traçando-se um comparativo entre as médias alcançadas pelos dois algoritmos nas duas execuções, percebe-se uma melhora geral relativa aos GAPs. Para o grupo de instâncias LA, obteve-se uma redução dos GAPs médios de 0,26 para 0,16 considerando o cenário com 60000 iterações sem melhora e uma redução de 0,27 para 0,17 num outro cenário com 80000 iterações sem melhora.

Para o grupo AZB:

Repetições sem Melhora=60000

Tabela 7: Grupo AZB (5 a 9) - execução do algoritmo com até 60000 repetições sem melhora.

		GRASP+BUSCA TABU					GRASP+BUSCA TABU OTIMIZADO				
Nome (JxM)	**BKS**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**
abz5(10x10)	1234	1236	1239	1237,60	0,16	79	1236	1242	1238,00	0,16	17
abz6(10x10)	943	943	948	945,60	0,00	74	943	948	946,00	0,00	14
abz7(20x15)	656	672	705	679,20	**2,44**	**406**	665	678	672,50	**1,37**	**72**
abz8(20x15)	665	675	690	686,20	1,50	461	677	695	683,90	1,80	117
abz9(20x15)	679	690	700	692,40	**1,62**	**473**	689	714	693,20	**1,47**	**123**
Média				**848,20**	**1,14**	**298,60**			**846,72**	**0,96**	**68,60**

Repetições sem Melhora=80000

Tabela 8: Grupo AZB (5 a 9) - execução do algoritmo com até 80000 repetições sem melhora.

		GRASP+BUSCA TABU					GRASP+BUSCA TABU OTIMIZADO				
Nome (JxM)	**BKS**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**
abz5(10x10)	1234	1238	1238	1238,00	0,32	108	1236	1240	1237,80	0,16	24
abz6(10x10)	943	943	948	945,40	**0,00**	**97**	943	948	944,50	**0,00**	**20**
abz7(20x15)	656	665	678	672,00	1,37	526	666	673	669,50	1,52	374
abz8(20x15)	665	677	685	679,80	1,80	675	674	689	680,50	1,35	114
abz9(20 x 15)	679	689	708	695,00	1,47	698	690	699	693,10	1,62	130
Média				**846,04**	**0,99**	**420,80**			**845,08**	**0,93**	**132,40**

Ao se analisar o grupo de instâncias ABZ, percebe-se que as execuções otimizadas trouxeram, de modo geral, uma redução no tempo de execução. O mesmo não pôde ser observado quanto aos GAPs encontrados que variaram entre melhora, piora e igualdade. Para as instâncias maiores, verifica-se que os GAPs estiveram mais instáveis, variando entre melhora e piora e, contudo, contando com a redução do tempo de execução. Já na execução do algoritmo otimizado para as instâncias menores, conseguiu-se manter os valores de GAP obtidos, ou melhorar, sem zerar.

As médias encontradas apresentam proximidade entre os GAPS médios, o que denota o comportamento do algoritmo com a solução inicial. Ao se observar as instâncias ABZ5, ABZ[7-9], percebe-se o término prematuro da execução sem a obtenção de significante melhora do GAP, fato ainda mais ressaltado na execução da primeira instância.

Para o grupo ORB:

Repetições sem Melhora=60000

Tabela 9: Grupo ORB (1 a10) - execução do algoritmo com até 60000 repetições sem melhora.

		GRASP+BUSCA TABU					GRASP+BUSCA TABU OTIMIZADO				
Nome (JxM)	**BKS**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**
orb01(10x10)	1059	1059	1072	1069,20	**0,00**	**160**	1060	1085	1070,60	**0,09**	**21**
orb02(10x10)	888	888	892	889,20	0,00	120	888	890	889,00	0,00	17
orb03(10x10)	1005	1005	1028	1020,40	0,00	123	1005	1042	1019,30	0,00	23
orb04(10x10)	1005	1012	1019	1013,80	0,70	96	1011	1019	1013,30	0,60	19
orb05(10x10)	887	894	903	898,80	**0,79**	**92**	887	898	893,60	**0,00**	**20**
orb06(10x10)	1010	1016	1023	1020,40	**0,59**	**169**	1010	1029	1017,90	**0,00**	**31**
orb07(10x10)	397	397	400	398,40	0,00	83	397	404	399,30	0,00	17
orb08(10x10)	899	908	927	918,40	**1,00**	**120**	899	943	915,00	**0,00**	**21**
orb09(10x10)	934	934	943	938,60	0,00	102	934	943	937,90	0,00	18
orb10(10x10)	944	946	952	950,20	**0,21**	**91**	944	957	947,60	**0,00**	**16**
Média				**911,74**	**0,33**	**115,60**			**910,35**	**0,07**	**20,30**

Repetições sem Melhora=80000

Tabela 10: Grupo ORB (1 a10) - execução do algoritmo com até 80000 repetições sem melhora.

		GRASP+BUSCA TABU					GRASP+BUSCA TABU OTIMIZADO				
Nome (JxM)	**BKS**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**
orb01(10x10)	1059	1065	1077	1071,80	**0,57**	**181**	1059	1089	1074,10	**0,00**	**26**
orb02(10x10)	888	888	889	888,80	0,00	93	888	889	888,90	0,00	24
orb03(10x10)	1005	1018	1027	1022,20	1,29	141	1008	1098	1029,00	0,30	27
orb04(10x10)	1005	1011	1015	1012,60	0,60	134	1011	1015	1013,00	0,60	24
orb05(10x10)	887	894	899	896,40	0,79	126	889	902	895,40	0,23	26
orb06(10x10)	1010	1016	1022	1019,40	**0,59**	**154**	1010	1023	1018,10	**0,00**	**49**
orb07(10x10)	397	397	398	397,20	0,00	118	397	401	398,00	0,00	21
orb08(10x10)	899	912	927	919,20	**1,45**	**135**	899	958	914,90	**0,00**	**24**
orb09(10x10)	934	934	943	936,20	0,00	132	934	943	936,60	0,00	26
orb10(10x10)	944	944	946	944,80	0,00	118	944	951	945,60	0,00	24
Média				**910,86**	**0,53**	**133,20**			**911,36**	**0,11**	**27,10**

Com a execução do grupo de instâncias ORB, foi observado que o GRASP + Busca Tabu Otimizada melhorou substancialmente os resultados obtidos, resolvendo com otimalidade para oito das dez instâncias testadas.

Assim, com base na Tabela 9, o comportamento geral é que o GAP obtido com a otimização consegue repetir o valor obtido na primeira execução e reduzir o tempo. Em alguns casos, porém, o GAP não chega a zerar, ainda que o tempo de execução seja muito curto, como pode ser visto em ORB01. Isto ocorre devido a um término prematuro da Busca Tabu Otimizada, incentivado pelas não-melhorias verificadas pela Busca Tabu, impedidos de melhorar, provavelmente, dada à marcação dos movimentos proibidos registrados pela lista Tabu. Quando se compara os GAPs médios, observa-se uma redução de 0,33 para 0,07 e de 0,53 para 0,11 nas Tabelas 9 e 10, respectivamente.

De modo geral, também para esta instância, percebe-se, através das médias, melhorias tanto do tempo de execução quanto do GAP médio obtido.

Para o grupo YN:

Repetições sem Melhora=60000

Tabela 11: Grupo YN (1 a 4) - execução do algoritmo com até 60000 repetições sem melhora.

		GRASP+BUSCA TABU					GRASP+BUSCA TABU OTIMIZADO				
Nome (JxM)	**BKS**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**
yn1(20x20)	826	899	909	903,80	8,84	657	900	909	905,00	8,96	118
yn2(20x20)	861	916	927	922,40	6,39	643	919	933	925,70	6,74	128
yn3(20x20)	827	908	915	911,80	9,79	818	903	912	908,20	9,19	170
yn4(20x20)	918	982	1036	1006,00	6,97	815	977	1074	1020,00	6,43	139
Média				**936**	**8**	**733,25**			**939,73**	**7,83**	**138,75**

Repetições sem Melhora=80000

Tabela 12: Grupo YN (1 a 4) - execução do algoritmo com até 80000 repetições sem melhora.

		GRASP+BUSCA TABU					GRASP+BUSCA TABU OTIMIZADO				
Nome (JxM)	**BKS**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**	**Melhor**	**Pior**	**Media**	**GAP**	**Tempo**
yn1(20 x 20)	826	900	905	903,40	8,96	858	896	908	902,00	8,47	163
yn2(20 x 20)	861	923	932	925,40	7,20	892	916	929	922,40	6,39	164
yn3(20 x 20)	827	906	911	908,60	9,55	983	903	915	908,90	9,19	213
yn4(20 x 20)	918	980	1085	1009,80	6,75	963	989	1141	1042,30	7,73	213
Média				**936,80**	**8,12**	**924**			**943,90**	**7,95**	**188,25**

O grupo de instâncias YN traz uma característica em comum: a complexidade de resolução definida pela dimensão para todas as instâncias. Também se observou que, apesar de não se haver alcançado o GAP zero, o tempo de execução ficou bastante reduzido, inclusive com a redução do GAP para algumas instâncias. Em relação ao GAP médio, foi obtida uma redução de 8,12 para 7, 95, conforme se verifica na Tabela 12.

Por fim, tendo em vista todos os grupos de instâncias, os resultados demonstraram ser possível alcançar o BKS com mais rapidez, através da otimização, para 31% das instâncias testadas. As instâncias de ORB, por exemplo, apresentaram resultados iguais aos encontrados na literatura para 80% delas, sendo também mais rápido com a otimização. Da mesma forma para as instâncias de LA, que alcançaram a mesma qualidade para 49% das instâncias. Estes são resultados que demonstram a robustez do algoritmo.

Comparando-se as versões de implementação do hibridismo GRASP e Busca Tabu, percebe-se que a versão com modelagem matemática consegue reduzir o tempo gasto no processamento para se alcançar valores de GAP iguais ou melhores que os obtidos com a versão do hibridismo simplificado para 79% das instâncias.

6. CONSIDERAÇÕES FINAIS

Destacam-se como contribuições deste trabalho: o método inovador para a construção da solução inicial, o hibridismo de GRASP e Busca Tabu aplicado ao problema de *job shop scheduling* e a incorporação da modelagem matemática como mecanismo de busca local.

Conforme apresentado até então, o foco deste trabalho foi o de propiciar uma solução inicial que favorecesse as buscas locais por melhores soluções. E como foram encontrados, na literatura observada, indicativos de boas práticas quanto ao uso das metaheuristicas utilizadas, esse estudo ateve-se ao uso do GRASP sob a óptica dos grafos disjuntivos e da Busca Tabu. Todavia, não se recomenda a exclusão de outras possibilidades de pesquisa.

Conforme apresentado, a formulação do GRASP em função da heurística de Grafos Disjuntivos foi essencial para a proposta da Rotulação Dinâmica, sendo sua aplicação também apoiada em tais concepções. Outra contribuição do uso dos Grafos Disjuntivos foi em relação ao sistema de avaliação das soluções, que se baseou nos conceitos que regem o Caminho Crítico. Obtida a solução inicial, fez-se necessário o uso de uma metaheurística que trouxesse benefícios em relação ao processo de busca local. Daí a junção com a metaheurística Busca Tabu.

Observados os conceitos apresentados, foi percebida a necessidade de se analisar o comportamento de cada formulação do algoritmo em relação a uma gama de entradas, não apenas na sua forma original, mas também em suas variações, principalmente, para se analisar os valores agregados com cada versão. As metaheurísticas adotadas foram amplamente testadas para um conjunto de problemas-teste conhecidos na literatura. Um paralelo entre o uso da formulação do hibridismo simplificado e do otimizado foi estabelecido, uma vez que utilizaram o mesmo processo de construção da solução inicial.

Ao se observar todos os conjuntos de instâncias aferidas nos experimentos computacionais, foi verificado que, em muitos casos, o GAP zero é rapidamente alcançado, o que denota a relevância da formulação da solução inicial. Com a aplicação da otimização, o comportamento mais frequente é a redução (ou repetição) do GAP (otimizado) em relação ao obtido (com o algoritmo básico), o que ocorreu para 82% das instâncias testadas no primeiro ciclo (60000 repetições sem melhora) e 87%, para o segundo ciclo (80000 repetições sem melhora).

Embora haja casos em que o hibridismo GRASP + Busca Tabu Otimizada encontra o GAP nulo com tempo superior ao hibridismo básico; foi observado que, quando o hibridismo otimizado é aplicado o tempo de execução tende a diminuir, enquanto que o GAP, normalmente, vem a zerar ou diminuir em relação ao hibridismo sem otimização, isso ocorre para 79% das instâncias testadas no primeiro ciclo (60000 repetições sem melhora) e 77%, para o segundo ciclo (80000 repetições sem melhora).

Não sendo considerado o tempo de execução total da solução otimizada proposta, isto é, com o uso da "Exploração de Vizinhança via Modelo Matemático", pode-se ponderar que haja uma melhora no tempo de execução do híbrido GRASP e Busca Tabu Otimizado em relação ao GAP da solução híbrida GRASP e Busca Tabu Simplificado, o que pode ser quantificada com base nos GAPs médios. Em relação às instâncias do grupo TA, os GAPs médios diminuíram de 1,35 para 1,27 e de 1,35 para 1,08, considerando, respectivamente os cenários de 60000 e de 80000 iterações sem melhora. Para o grupo de instâncias LA, obteve-se uma redução dos GAPs médios de 0,26 para 0,16 considerando o cenário com 60000 iterações sem melhora e uma redução de 0,27 para 0,17 num outro cenário com 80000 iterações sem melhora. Para o grupo de instâncias ABZ também ocorreram reduções quanto aos GAPs médios, tanto no cenário de 60000 iterações sem melhora, de 1,14 para 0,96, quanto para o cenário de 80000 iterações sem melhora, de 0,99 para 0,93. Observando-se as instâncias ORB, percebe-se reduções de GAPs ainda mais relevantes, sendo de 0,33 para 0,07 na execução dos algoritmos com 60000 repetições, e de 0,53 para 0,11 na execução dos algoritmos com 80000 repetições sem melhora. Para o grupo de instâncias YN, as reduções de GAPs foram de 8 para 7,83 na execução dos algoritmos com 60000 repetições sem melhora e de 8,12 para 7,95 na execução dos algoritmos com 80000 repetições sem melhora.

Foi observado o aumento da possibilidade de se alcançar o GAP nulo para instâncias com menor número de máquinas. Onde, quanto maior a proporção n/m (n_job/n_máquinas) melhores são os resultados alcançados. Considerando estudos encontrados na literatura, bem como os experimentos realizados, é provável que tal proporção possa influenciar nas especificações da implementação adotada para cada instância.

Considerando que os resultados obtidos com a segunda proposta de solução estiveram, de modo geral, próximos dos produzidos pela primeira resolução, isto é, o GAP obtido poucas vezes foi superior, ou seja, chegou a zerar em contraprova da resolução simplificada; os testes demonstraram que a formulação da solução inicial possibilitou resultados aproximados.

Assim, pode-se inferir que a formulação utilizada para a geração da solução inicial, proposta neste trabalho, favorece os resultados finais. Assim, supõe-se que, caso se utilize um

algoritmo mais sofisticado na fase de busca local, a exemplo do utilizado na literatura destacada, provavelmente, poderiam se superar alguns valores resultantes.

Outrossim, para trabalhos futuros, seria válido se analisar o comportamento da formulação do hibridismo entre outras metaheurísticas e o GRASP — com a Rotulação Dinâmica —, com fins de aferição. Como foi demonstrado pelos experimentos computacionais, a relação de dependência entre a dimensão das instâncias e o desempenho do algoritmo, seria válido um estudo mais apurado em relação a tal característica, com a aplicação de metaheurísticas diferentes para cada perfil de instância, ou mesmo, a introdução de mais uma metaheurística em fases de busca local, para instâncias maiores.

REFERÊNCIAS BIBLIOGRÁFICAS

[01] ADAMS, J., BALAS, E., e ZAWACK, D. (1988). **The shifting bottleneck procedure for job shop scheduling**. Management Science, 34:57–73.

[02] AIEX, R. M., BINATO, S. e RESENDE, M. G. C. (2003). **Parallel GRASP with path-relinking for job shop scheduling.** Parallel Computing **29**. p 393-430.

[03] ALVAREZ-VALDES, R.; FUERTES, A.; TAMARIT, J. M.; GIMENEZ, G. e RAMOS, R. (2005). **A heuristic to schedule flexible job-shop in a glass factory**. European Journal of Operational Research, **165**: 525-534.

[04] ANDERSON, JAMES H. (2004). **Handbook of Scheduling: Algorithms, Models, and Performance Analysis**. Ed. Chapman and Analysis.

[05] APPLEGATE, D. e COOK, W. (1991). **A computational study of the job-shop scheduling problem**. ORSA Journal on Computing, 3:149-156.

[06] ARMENTANO, V. e ARAÚJO, O. (2006). **GRASP with memory-based mechanisms for minimizing total tardiness in single machine scheduling with setup times**. J Heuristics, **12**: p 427-446.

[07] BINATO, S.; HENRY, W. J.; LOEWENSTERN, D. M. e RESENDE, M. G. C. (2001). **A GRASP for job shop scheduling**. Scientific Literature Digital Library. Acesso em 15/08/2009. Disponível em: <http://citeseer.ist.psu.edu/binato00grasp.html>.

[08] BLAZEWICZ, J.; PESH, E. e STERNA, M. (2000). **The disjuntive graph machine representation of the job shop scheduling problem**. European Journal of Operational Research, **127**: 317-331.

[09] BRASEL, H.; HERMS, A.; MÖRIG, M; TAUTENHANHN, T; TUSCH, J. e WERNER, F. (2007). **Heuristic constructive algorithms for open shop scheduling to minimize mean flow time**. European Journal of Operational Research, **189**: 856-870.

[10] BRUCKER, P., JURISCH, B., e SIEVERS, B. (1994). **A branch and bound algorithm for the job-shop scheduling problem.** Discrete Applied Mathematics, 49:105-127.

[11] CARLIER, J. e PINSON, E.. (1989). **An algorithm for solving the job-shop problem.** Management Science, 35:164-176.

[12] CRAVO, G., RIBEIRO, G., LORENA, L. (2006). **Um GRASP eficiente para o problema da rotulação cartográfica de pontos**. Acesso em: 15/08/2009. Disponível em: <http://www.lac.inpe.br/~lorena/glaydston/sbpo06-gildasio-glaydston-lorena.pdf>.

[13] FEO, T. A. e RESENDE, M. G. C. (1995). **Greedy randomized adaptive search procedures.** Journal of Global Optimization, **6**: p 109-133.

[14] GLOVER, F.(1986) **Future paths for integer programming and links to artificial intelligence**. Computers & Operations Research; 13:533-549.

[15] LENSTRA, J. K. and RINNOOY KAN, A. H. G. (1979). **Computational complexity of**

discrete optimization problems. Annals of Discrete Mathematics, 4:121-140.

[16] MOREIRA, M. C. de O.; ARROYO, J. E. C.; JANUÁRIO, T. de O. e OLIVEIRA JÚNIOR, P. L. (2006). **Um algoritmo genético para o problema job shop scheduling bi-objetivo**. Acesso em: 10/05/2009. Disponível em: <http://www.scribd.com>.

[17] MULLER, G. I. e GÓMEZ, A. T. (2006). **Estratégias de Escalonamento em um Ambiente de Job-shop**. Anais do XV Seminário de Computação, p 201-208.

[18] OLIVEIRA, R. L. (2000). **Escalonamento de um Job Shop:** Um Algoritmo com Regras Heurísticas. Acesso em: 02/03/2006. Disponível em: <http://www.inf.ufrgs.br/pos/SemanaAcademica/Semana2000/RonaldLopes>.

[19] PINSON, E. (1995). **The job shop scheduling problem**: ***A concise survey and some recent developments***. In P. Chrétienne, E. G. Coffman Jr., J.K. Lenstra, and Z. Liu, editors, Scheduling theory and its application, pages 277-293. John Wiley and Sons.

[20] RESENDE, M. G. C. e RIBEIRO, C. C (2002). **Greedy randomized adaptive search procedures.** AT&T Labs Research Technical Report TD-53RSJY, version 2. To appear in State of the Art Handbook in Metaheuristics, F. Glover and G. Kochenberger, eds., Kluwer.

[21] SUBRAMANIAN, A.; OCHI, L. S. e CABRAL, L. A. F. (2009). **An ILS based heuristic for the vehicle routing problem with simultaneous pickup and delivery and time limit**. Lectures Notes on Computer Science. 4972: 135-146.

[22] TAILLARD, E. D. (1989). **Parallel taboo search technique for the job shop scheduling problem**. Technical Report ORWP 89/11, DMA, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland.

[23] VAN LAARHOVEN, P. J. M., AARTS, E. H. L., LENSTRA, J. K.(1992). **Job shop scheduling by simulated annealing**. Operations Research;40(1):113–25.

[24] WATSON, J.; BECK, J. C.; HOWE, A.E. e WHITLEY, L. D. (2002). **Problem Difficulty for Tabu search in Job-Shop Scheduling**. Acesso em 02/01/2009. Disponível em: <http://portal.acm.org/citation.cfm?id=643185>

[25] ZHANG, C. Y., LI, P.; GUAN, Z. e RAO, Y. (2007). **A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem.** Computers & Operations Research **34**: p 3229-3242.

[26] ZHANG, C. Y., LI, P.; GUAN, Z. e RAO, Y. (2008). **A very fast TS/SA algorithm for the job shop scheduling problem.** Computers & Operations Research **35**: p 282 – 294.