UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

ABORDAGENS HEURÍSTICAS APLICADAS AO PROBLEMA DA ALOCAÇÃO DINÂMICA DE ESPAÇOS

WAGNER SILVA COSTA

JOÃO PESSOA - PB

AGOSTO - 2013

ABORDAGENS HEURÍSTICAS APLICADAS AO PROBLEMA DA ALOCAÇÃO DINÂMICA DE ESPAÇOS

Wagner Silva Costa

DISSERTACÃO SUBMETIDA AO CORPO DOCENTE DA COORDENACÃO DO PROGRAMA DE PÓS-GRADUAÇÃO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DA PARAÍBA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS DA COMPUTACÃO.

Orientador:

Prof. Lucídio dos Anjos Formiga Cabral, D.Sc.

JOÃO PESSOA – PB

AGOSTO - 2013

CONTEÚDO

Lista de Figuras

Lista de Tabelas

Lista de Siglas

PADE	Problema de Alocação Dinâmica de Espaços
GRASP	General Responsibility Assignment Software Patterns
HGT	Híbrido GRASP Tabu Search
FA	First Assignment
VND	Variable Neighborhood Descent
ES	Espaço de Trabalho
D	Depósito
PQA	Problema Quadrático de Alocação
PQAG	Problema Quadrático de Alocação Generalizado
PDLF	Problema Dinâmico de Layout de Facilidades
WAP	Workspace Allocation Problem
SSAP	Storage Space Allocation Problem
LIC	Lista Inicial de Candidatos
LRC	Lista Restrita de Candidatos
RC	Randomized Clustering

A minha esposa, Maria Valdenia Sales Ferreira Silva.

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 dos amigos, do incentivo e do companheirismo, onde se faz toda a diferença. E pelo que muito venho a agradecer:

- A Deus por me amparar nos momentos difíceis, me dar força interior para superar as dificuldades, mostrar os caminhos nas horas incertas e suprir em todas as minhas necessidades.
- A minha esposa Maria Valdenia Sales Ferreira Silva que Deus colocou em minha vida e escolhi para viver, amor incondicional e em nenhum momento a distância não nos separa.
- A minha mãe Josefa Bernardo da Silva, amor incondicional de mãe, pela vida e amor que me deu. Muitas vezes, renunciou aos seus sonhos para que eu pudesse realizar o meu, partilho a alegria deste momento.
- Ao meu irmão, Rubens Silva Bernardo, pelo apoio, paciência e incentivo ao longo desta caminhada.
- Meu orientador, Lucídio dos Anjos Formiga Cabral, queria te dizer OBRIGADO por TUDO. Nesse mundo, repleto de pessoas ruins, você me faz acreditar que os bons são a maioria. Só tenho a agradecer aos seus ensinamentos (pessoais e acadêmicos), orientações, palavras de incentivo, paciência e dedicação. Você é uma pessoa ímpar, o meu profundo e sentido agradecimento, em especial a sua família: sua esposa Ana Maria e seus filhos: Ana Luiza, Bruno e Gabriel. Tenho orgulho em dizer que um dia fui seu orientando e também dizer que sou seu AMIGO.
- Ao Professor Gilberto Farias de Sousa Filho, muito obrigado pela ajuda, ensinamentos, orientações e contribuições.
- Ao meu Amigo Lettiery D'Lamare Portela Procópio, o meu muito obrigado pela ajuda, ensinamentos e contribuições para o término deste trabalho.
- Aos meus Amigos Antonio Junio Figueiredo da Mata, Francisco Hélio Pereira de Souza, Francisco Vanier de Andrade e José Galdino da Silva, que, na alegria ou na tristeza, estão comigo e torcem por mim.
- A minha família.
- A todos vocês, meus sinceros agradecimentos.

Resumo

O Problema da Alocação Dinâmica de Espaços (PADE) é recente na literatura e foi inspirado na necessidade de minimizar as distâncias percorridas entre recursos solicitados para a execução de atividades em centrais de energia nuclear. Existem semelhanças do problema com projetos em que a movimentação de recursos gera custos para o planejamento final, ou em casos onde o congestionamento destes recursos não é aconselhável, como em construções de grandes obras ou na mineração. Faz-se necessário, o uso de métodos aproximativos por ser este considerado NP-Difícil. Para isso, uma nova heurística construtiva é proposta utilizando um modelo em programação linear inteira baseado em fluxos de atividades que incorpora a informação do grafo de dependências entre as atividades e, então, uma aplicação híbrida da meta-heurística GRASP é discutida. Os resultados computacionais demonstram que esta abordagem é capaz de obter soluções de alta qualidade.

Palavras-chave — modelagem em programação linear inteira; grafos de dependências; problema de alocação de recursos; GRASP.

Abstract

The Dynamic Allocation Problem Spaces (DSAP) is recent literature, and was inspired by the need to minimize the distances between requested resources for the execution of activities in nuclear power plants. There are similarities of the problem with projects in which the movement of resources generates costs for the final planning, or in cases where congestion of these resources is not advisable, as in large buildings or works in mining. It is necessary, the use of these approximation methods considered to be NP hard. For this, a new constructive heuristic is proposed using integer linear programming model based on activity streams that incorporate information from the graph of dependencies between activities, and then an application of the hybrid GRASP meta-heuristic is discussed. The computational results show that this approach is able to obtain high-quality solutions.

Keywords - modeling in integer linear programming; dependency graphs; issue of resource allocation; GRASP.

Capítulo 1

Introdução

O Problema da Alocação Dinâmica de Espaços (PADE) foi inspirado na necessidade de otimização da distância percorrida por recursos necessários para a realização de atividades em projetos. Inicialmente foi proposto tendo como aplicação a manutenção de redes de energia nuclear. Mais tarde, identificou-se que o mesmo tem aplicações em outros projetos em que a movimentação de um recurso é uma tarefa custosa, cujo congestionamento desses recursos não é aconselhável, como em construção de pontes e atividades que envolvem mineração. Um projeto é dividido por um número de períodos consecutivos e, em cada um deles, uma quantidade de atividades é realizada. Os recursos necessários para as atividades devem ser associados a espaços de trabalho e, os recursos ociosos no período devem ser guardados em depósitos. O objetivo do problema é minimizar a distância total percorrida pelos recursos entre os espaços (de trabalho ou depósitos) [1].

Este problema é considerado NP-Difícil [2,3,18], e portanto, a dificuldade em encontrar uma solução ótima para instâncias de elevadas dimensões justifica o uso de métodos aproximativos.

O presente trabalho apresenta, primeiramente, algoritmos de construção já explorados em outros trabalhos com as metaheurísticas GRASP e VND contrapondo com uma nova proposta baseada em grafo de dependências de atividades e movimentos de vizinhança para o refinamento da solução encontrada em tempos viáveis. O conjunto de instâncias e um modelo em programação matemática disponível na literatura são utilizados junto ao CPLEX para obtenção das respectivas soluções ótimas, porém, limita-se a algumas pela dimensão da instância. Assim foi possível chegar a soluções ótimas para algumas instâncias ainda com valor exato em aberto, comprovar a eficiência das heurísticas propostas e impugnar alguns resultados da literatura.

1.1 Objetivos do Trabalho

Tendo em vista os aspectos apresentados, o objetivo principal do trabalho é o desenvolvimento de métodos híbridos baseados nas metaheurísticas para a resolução de problemas da alocação dinâmica de espaços baseados em uma heurística que explora o grafo de dependências entre atividades. Em particular, o método proposto utilizará uma aplicação híbrida da metaheurística GRASP, explorando sua eficiência computacional, combinados com etapas de refinamentos compostas por métodos exatos para acelerar a convergência dos métodos. Os objetivos específicos desse trabalho são:

- Efetuar o levantamento bibliográfico sobre as abordagens heurísticas e exatas aplicadas na resolução do problema da alocação de espaço;
- Estudar o problema da alocação dinâmica de espaços explorando as relações de dependência entre atividades e recursos a luz da teoria dos grafos;
- Desenvolver novos métodos heurísticos para o teste das novas ideias e conhecimentos adquiridos pelos estudos das referências bibliográficas sobre o PADE;
- Comparar os resultados obtidos pela nova construção com os trabalhos recentes encontrados na literatura utilizando um conjunto de instâncias testes oriundas da literatura.

1.2 Organização da proposta

O restante desta dissertação está estruturada da seguinte forma:

- Capítulo 2: Traz uma revisão bibliográfica dos principais trabalhos sobre o PADE. É feita a caracterização do problema e são mostradas as suposições, além disso, é apresentada uma solução para o problema da alocação dinâmica de espaços.
- Capítulo 3: Apresenta a fundamentação teórica a respeito da alocação dinâmica de espaços, os principais algoritmos descritos na literatura e as estruturas de vizinhança relacionadas ao problema. Na Seção 3.6 referente às metaheurísticas, é apresentada uma descrição sobre as mesmas utilizadas nesse problema, como *Simulated Annealing* (SA) e *Tabu Search* (Busca Tabu).
- Capítulo 4: Aborda as instâncias conhecidas do problema com seus dados de entrada, *layouts* e arquivos, ao término é exemplificado e detalhado as estruturas que representam uma solução para o problema da alocação dinâmica de espaços. Apresenta a proposta deste trabalho descrevendo uma estrutura de dependências criada para ser utilizada pelo modelo em programação linear inteira proposto para o problema, e a

adequação da metaheurística GRASP para o novo contexto, e por fim é também proposta uma nova estrutura de vizinhança.

- Capitulo 5: Este capítulo traz os resultados alcançados com este trabalho. Inicialmente se descreve os valores obtidos por [1], além de descrever o ambiente em que foram feitas as execuções. Posteriormente, se apresenta os resultados obtidos com comparações de diferentes implementações e abordagens.
- Capitulo 6: O último capítulo apresenta as conclusões obtidas com o trabalho e discute possíveis abordagens para trabalhos futuros.

1.3 Motivação

O advento tecnológico surgiu com a necessidade de realizar grandes cálculos para estratégias bélicas na segunda Guerra Mundial, assim, surge também a Investigação Operacional ou Pesquisa Operacional, sendo um ramo interdisciplinar da matemática aplicada que através de modelos em programação linear inteira constrói algoritmos para analisar sistemas complexos do mundo real conseguindo melhorar ou otimizar resultados e performances antes já conhecidos.

Com o fim das guerras e a expansão capitalista surge uma nova competição mundial, a concorrência entre as grandes empresas que para inovar nas suas estratégias de mercado utilizam das novas tecnologias como um auxilio nos seus processos de produção, administração e distribuição. Dentro desta realidade nasce a logística como uma área da gestão responsável por planejar, implementar e controlar a movimentação e o armazenamento de produtos e serviços relacionados, abrangendo desde a origem deste recurso até o consumo do mesmo, como relatado por [11].

Em [12], relata que a logística tem como um dos seus objetivos proporcionar a aprimoração dos serviços oferecidos aos seus clientes. Desta maneira, a logística é mais um artefato estratégico que as empresas podem utilizar para organizar suas atividades e seu contexto vai desde a escolha adequada dos fornecedores, até atingir o cliente final do processo.

Nesta realidade [13] nos mostra que a logística sendo apropriadamente compreendida, serve para a criação de estratégias que permitem a redução dos custos e aperfeiçoamento dos serviços oferecidos. Com estes fatos, surge a capacidade de consolidar as diferenças competitivas no mercado, sendo esta a forma escolhida por diversas empresas para atribuir vantagens a seu nome. Assim, é exposto um dos

problemas da logística: é a localização de facilidades, que pode ser representada fisicamente por depósitos, postos policiais, escolas, usinas nucleares, etc.

Muitas soluções para este problema contudo, são vistas através de modelos que buscam minimizar o custo do projeto como um todo, utilizando métodos exatos, porém estas abordagens se tornam bastantes restritas tendo em vista o tamanho do problema e o método a ser resolvido. Neste cenário podemos utilizar abordagens heurísticas que são capazes de encontrar soluções de boa qualidade que se aproximam da ótima conhecida.

Este trabalho tem como principal motivação a construção de ideias e algoritmos para o avanço de resultados no Problema da Alocação Dinâmica de Espaços semelhantes a projetos que buscam melhorar a localização de facilidades, através de uma revisão bibliográfica dos principais trabalhos publicados. Por outro lado, o problema traz um cenário onde a movimentação de recursos custa muito para o projeto, assim visamos a otimização do custo final, através de metaheurísticas e modelos embutidos na construção e no refinamento da solução.

Tendo em vista este cenário, o PADE traz soluções para diversos problemas reais como as distâncias que percorrem os recursos para a manutenção de redes de energias nucleares ou ainda o deslocamento de tratores de depósitos para a realização de atividades em minas, ou seja, o Problema de Alocação Dinâmica de Espaços pode trazer melhores soluções para projetos que antes eram resolvidos levando em consideração o menor custo da movimentação atual e não o custo da solução global.

Capítulo 2

O Problema da Alocação Dinâmica de Espaços

O Problema da Alocação Dinâmica de Espaços (PADE) que foi inspirado na necessidade de otimizar a distância percorrida pelos recursos necessários para a realização de atividades em projetos. Originalmente foi proposto tendo como aplicação a manutenção de redes de energia nuclear. Mais tarde, identificou-se que o mesmo tem aplicações potenciais em outros projetos em que a movimentação de um recurso é uma tarefa difícil ou cara, cujo congestionamento desses recursos não é aconselhável, como em construção de pontes e atividades que envolvem mineração.

2.1 Um Breve Exemplo

Número dos Recursos (Máquinas)	**Descrição**
1	Perfuradeira leve
2	Máquina que faz o cimento
3	Laboratório de solos
4	Extratora de solos
5	Perfuradeira de força
6	Caminhão de carga
7	Retroescavadeira
8	Máquina que constrói colunas de sustentação
9	Extratora de grandes pedras

Número da Atividade	Descrição	Tempo necessário (Mês)	Meses início - fim
1	Retirar Entulho	1	1 – 1
2	Perfurar	2	1 – 2
3	Analisar Solo	1	2 - 2
4	Preparar Túnel	2	3 – 4
5	Retirar Grandes Pedras	1	4 - 4

2.1.1 Como alocar as atividades e os recursos desta agenda de projeto?

Períodos	Atividades (Recursos)	Recursos Ociosos
1	A1(6,7) A2(1,5)	2,3,4,8,9
2	A2 (1,5) A3 (3,4)	2,6,7,8,9
3	A4 (2,8)	1,3,4,5,6,7,9
4	A4 (2,8) A5(6,9)	1,3,4,5,7

Solução: Parte 1/2

Alocação de Atividades

Períodos	Espaços de Trabalho		
	ES1	ES2	ES3
1	A2 (1,5)		A1 (6,7)
2	A2 (1,5)	A3 (3,4)	
3	A4 (2,8)		
4	A4 (2,8)		A5 (6,9)

Solução: Parte 2/2

Alocação de Recursos Ociosos

Períodos	Depósitos		
	D1	D2	D3
1	2,8	3,4	9
2	2,8		6,7,9
3	1,5	3,4	6,7,9
4	1,5	3,4	7

2.2 Descrição Formal do PADE

Em todos esses casos, o espaço é um recurso escasso importante para a redução dos custos de manipulação de materiais, alocando com eficiência as atividades e os equipamentos e materiais de construção ativos ou ociosos, sendo o seu uso eficiente fundamental para a redução das ocorrências de congestionamento.

Com isto surge a movimentação dos recursos que pode ser classificada em três tipos:

- *Atividade para Atividade*: neste movimento um determinado recurso é requerido por uma atividade e após seu uso (no término da atividade) é solicitado para outra atividade, criando assim um trajeto entre dois espaços de trabalho;
- *Atividade para Depósito*: neste movimento acontece a saída de um recurso de um determinado local de trabalho, no qual estava sendo utilizado por uma atividade e, ao término dessa execução, o recurso não é mais requerido por nenhuma atividade no período, necessitando de alocação em um depósito, traçando um caminho entre um espaço de trabalho e um depósito;
- *Depósito para Atividade*: com este movimento, o recurso faz o caminho inverso do *Movimento de Atividade para Depósito*, uma vez que, em um determinado período, o recurso que está em um depósito é solicitado para uma atividade, realizando o trajeto do depósito para um espaço de trabalho.

Surge também uma nova variável do PADE, o custo, que por sua vez, é utilizada para classificar numericamente o valor gerado pela solução encontrada e seu valor é a soma de todos os deslocamentos realizados pelos recursos, de tal forma que, em cada movimento tem-se um valor do custo agregado, existindo extremos conhecidos, onde o mínimo equivale a 0 (zero) unidades e o máximo equivale ao maior deslocamento possível dentro do contexto do problema.

No Movimento de *Atividade para Atividade* o custo é um valor entre zero, onde as duas atividades estão no mesmo local de trabalho, e seu valor máximo, onde as atividades estão nos locais de trabalho mais distantes do problema.

Já nos Movimentos de *Atividade para Depósito* e *Depósito para Atividade*, temos como seus custos mínimos equivalente a uma unidade, pois em todo caso é necessário o deslocamento do recurso para outra localização e seu custo máximo também equivale à maior distância entre um local de trabalho e um depósito no

problema.

Assim, para a contagem de um custo individual gerado por cada atividade, é necessária a soma dos custos criados pelos movimentos de recursos que foram requeridos para sua execução, sendo possível alcançar como custo mínimo zero unidades, onde neste caso específico duas atividades consecutivas compartilham dos mesmos recursos e iguais locais de trabalho, onde ainda há uma coincidência do término das suas execuções com o término do projeto geral, que por sua vez, a última atividade não terá custo algum.

O PADE apresenta relações muito próximas com outros problemas importantes na área de Otimização Combinatória: o Problema Quadrático de Alocação (PQA), o Problema Dinâmico de *Layout* de Facilidades (PDLF) e o Problema Quadrático de Alocação Generalizado (PQAG), referenciados na literatura, respectivamente, como Quadratic Assignment Problem (QAP), Dynamic Facility *Layout* Problem (DFLP) e Generalized Quadratic Assignment Problem (GQAP).

Em [14] verificamos que o Problema Quadrático de Alocação (PQA) é o problema da designação (com custo mínimo) de objetos a locais, em que cada objeto deve ser atribuído a um único local e reciprocamente, sendo conhecidas as distâncias entre pares de locais e os fluxos de algum tipo de demanda entre pares de objetos, assim o PQA assemelha-se com problemas conhecidos como o caixeiro-viajante, o clique maximal, o problema de particionamento e o de isomorfismo de grafos, que por sua vez ganha variação na literatura como o Problema Quadrático de Alocação Generalizado (PQAG).

No Problema Dinâmico de *Layout* de Facilidades (PDLF) [10] a atribuição de objetos a locais deve ser programada para um número de períodos consecutivos, podendo os fluxos entre objetos variar de um período para outro, levando-se ainda em conta o custo de realocação dos objetos. O PDLF pode ser visto então como uma sucessão de PQA's. Também como uma generalização do PQA, o Problema Quadrático de Alocação Generalizado (PQAG) [15] considera alocar, com custo mínimo, múltiplos objetos a um mesmo local, de acordo com sua capacidade.

O PADE pode ser delimitado em dois subproblemas: o primeiro consiste na associação das atividades aos espaços de trabalho (*Workspace Allocation Problem* - WAP) e o segundo consiste na alocação de recursos ociosos aos depósitos (*Storage Space Allocation Problem* - SSAP). O comportamento do WAP pode ser visto como um PDLF. No PADE o custo final de um projeto é dado pela soma de todos os custos

gerados pelos movimentos dos recursos, do mesmo modo que no PDLF, cujo custo total é dado pela junção do custo dos fluxos de materiais com o custo da realocação das facilidades. No segundo problema, observam-se relações com o PQAG, onde é considerada a alocação de objetos (recursos ociosos) em mesmo local (depósito), respeitando uma capacidade pré-determinada, da mesma forma que é abordado no SSAP.

De uma forma geral, podemos conceituar o PADE, como uma agenda de atividades de um projeto dividida em períodos. A cada período um conjunto de atividades deve ser realizado, sendo que, cada atividade solicita um determinado conjunto de recursos para sua realização. Um recurso considera-se necessário quando for utilizado por alguma atividade do período, ou ocioso, caso contrário, isto é, quando nenhuma das atividades realizadas no período o solicita. Uma solução do problema resulta na associação dos recursos a espaços próprios, a saber: atividades e seus recursos a espaços de trabalho e recursos ociosos a depósitos. São necessários como dados de entrada do PADE:

- O *layout* (Espaços de Trabalho e Depósitos) do projeto juntamente com as distâncias e a capacidade de cada espaço, ou seja, a quantidade máxima de recursos suportada por cada local deste *layout*.
- O fluxo do projeto com a quantidade e o conjunto das atividades a serem processadas por períodos, com os recursos a serem utilizados nas atividades. Podemos considerar que apenas uma atividade pode ser realizada no espaço de trabalho individualmente por um determinado período de tempo. Também é considerado que a capacidade do local de trabalho é suficiente para o armazenamento dos recursos da atividade a ela associada. Deste modo, cada atividade requer apenas um local de trabalho e pelo menos um recurso.

Para evitar a movimentação desnecessária de recursos e assim, minimizar a distância percorrida por esses, ressalta-se também que, no caso de uma atividade ser realizada por mais de um período consecutivo, esta deve permanecer sempre no mesmo espaço de trabalho em cada um destes períodos.

Segundo [3], são feitas as seguintes suposições para o PADE:

1. Os locais do espaço de trabalho e depósitos são conhecidas;
2. As distâncias entre locais são determinadas a priori;
3. Os recursos requeridos para realizar cada atividade são conhecidas;
4. O escalonamento de atividades é dado;

5. Somente uma atividade pode ser realizada em um espaço de trabalho em cada período, e o espaço de trabalho associado para cada atividade é grande o suficiente para realizar a atividade e armazenar os recursos requeridos;

6. Cada atividade requer somente um espaço de trabalho e pelo menos um recurso;

7. Se uma atividade é realizada em múltiplos períodos, a atividade é associada ao mesmo espaço de trabalho a cada período em que é realizada;

8. As capacidades dos depósitos são conhecidas;

9. O objetivo é minimizar a distância total dos recursos, através da duração do projeto;

10. Se um recurso é ocioso em períodos consecutivos, ele é associado ao mesmo espaço de armazenamento durante estes períodos.

As Tabelas 2.1 a 2.3 apresentam um exemplo dos dados de entrada para uma instância do problema.

A Tabela 2.1 apresenta a agenda de um projeto com 6 atividades e com 4 períodos de tempo. Cada atividade traz, entre parênteses, os recursos que vão necessitar para suas funções, e na última coluna podemos ver os recursos ociosos a cada período. Por exemplo, no período 2, temos duas atividades em execução (A_2 e A_3) e os recursos ociosos destes períodos são 2, 6, 7, 8 e 9, sendo requeridos apenas os recursos 1 e 5 para a atividade A_2 e o recurso 3 e 4 para a atividade A_3.

Tabela 2.1. Agenda do Projeto

Períodos	**Atividades (Recursos)**	**Recursos Ociosos**
1	A_1(6,7) A_2 (1,5)	2,3,4,8,9
2	A_2(1,5) A_3 (3,4)	2,6,7,8,9
3	A_4 (2,8)	1,3,4,5,6,7,9
4	A_4 (2,8) A_5 (5,7) A_6 (6,9)	1,3,4

A Tabela 2.2 mostra um exemplo de *layout* com 6 localidades do PADE, sendo na primeira fileira 3 espaços de trabalho ES_1, ES_2 e ES_3, que representam na matriz distância, descrita pela Tabela 2 os índices 1, 2 e 3, respectivamente, e para os depósitos temos D_1, D_2 e D_3 com índices 4, 5 e 6. Levando-se em consideração a agenda do projeto, representada pela Tabela 2.1, a matriz de distâncias pela Tabela 2.3, o *layout* de 6 espaços visto na Tabela 2.3 e ainda uma capacidade máxima igual a 3 recursos por depósito, podemos construir uma solução para o PADE formada por duas estruturas.

Tabela 2.2. *Layout* com 6 espaços

ES_1	ES_2	ES_3
D_1	D_2	D_3

A Tabela 2.3 apresenta as distâncias entre os espaços, considerando-se a distância de Manhattan. As colunas/linhas 1 a 3 representam os espaços de trabalho e as colunas/linhas de 4 a 6, os depósitos.

Tabela 2.3. Matriz de distâncias entre locações

	1	**2**	**3**	**4**	**5**	**6**
1	0	1	2	1	2	3
2	1	0	1	2	1	2
3	1	2	0	3	2	1
4	1	2	3	0	1	2
5	2	1	2	1	0	1
6	3	2	1	2	1	0

A Tabela 2.4 nos mostra parte da solução para o projeto correspondente à alocação das atividades em seus determinados espaços de trabalho.

Tabela 2.4. Solução para Alocação de Atividades

Períodos	**Espaços de Trabalho**		
	ES_1	ES_2	ES_3
1	A_2 (1,5)		A_1 (6,7)
2	A_2 (1,5)	A_3 (3,4)	
3	A_4 (2,8)		
4	A_4 (2,8)	A_5 (5,7)	A_6 (6,9)

Na Tabela 2.5 é possível construir a solução para os recursos ociosos. Para avaliarmos a qualidade da solução descrita nas Tabelas 2.4 e 2.5 devemos somar as distâncias, definidas pela Tabela 2.2, percorridas pela movimentação de cada recurso entre os locais do *layout* durante as transições dos períodos. Por exemplo, o recurso 5 não é movimentado entre os períodos 1 e 2, no entanto, o mesmo entre os períodos 2 e 3 é movimentado do espaço de trabalho ES_1 para o depósito D_1 incorrendo em custo unitário. Em seguida o recurso 5 é movimentado para o espaço de trabalho ES_2 para ser usado pela atividade A_5, adicionando um custo de mais duas unidades ao custo total do projeto.

Tabela 2.5. Solução para Alocação de Recursos Ociosos

Períodos	**Depósitos**		
	D_1	**D_2**	**D_3**
1	2,8	3,4	9
2	2,8		6,7,9
3	1,5	3,4	6,7,9
4	1	3,4	

A alocação de recursos pode ser entendida da seguinte maneira: no período 1 as atividades A_2 e A_1 entram em execução e seus recursos são alocados no espaço de trabalho ES_1 e ES_3. Os recursos ociosos 2 e 8, 3 e 4 e 9 são associados aos depósitos D_1, D_2 e D_3, respectivamente. No período 2 a atividade A_3 entra em execução e seus recursos são transportados do depósito D_2 para o espaço de trabalho ES_2 e os recursos 2 e 8; 6,7 e 9 são associados aos depósitos D_1 e D_3, respectivamente. No período 3 a atividade A_4 entra em execução e, por isso, os recursos 2 e 8 são movimentados de D_2 para ES_3 e os recursos da atividade A_2 é levado do espaço de trabalho ES_2 para o depósito D_1, os recursos 3 e 4 são levados do espaço de trabalho ES_2 para D_2. No período 4 as atividades A_5 e A_6 entram em execução e seus recursos são alocados no espaço de trabalho ES_2 e ES_3. O recurso 1 é transferido do espaço de trabalho ES_2 para o depósito D_1 e os recursos 3 e 4 são transferidos de ES_2 para D_2.

A movimentação dos recursos detalhada no parágrafo anterior pode ser sumarizada na Tabela 2.6. A distância total percorrida pelos recursos é de 14 unidades.

Tabela 2.6. Distância total percorrida pelos recursos

	Períodos			
Recursos	1	2	3	4
1			$ES_1 \rightarrow D_1$	
2			$D_1 \rightarrow ES_1$	
3		$D_2 \rightarrow ES_2$	$ES_2 \rightarrow D_2$	
4		$D_2 \rightarrow ES_2$	$ES_2 \rightarrow D_2$	
5			$ES_1 \rightarrow D_1$	$D_1 \rightarrow ES_2$
6		$ES_3 \rightarrow D_3$		$D_3 \rightarrow ES_3$
7		$ES_3 \rightarrow D_3$		$D_3 \rightarrow ES_2$
8			$D_1 \rightarrow ES_1$	
9				$D_3 \rightarrow ES_3$
Distância	0	4	6	4

Capitulo 3

Fundamentação Teórica

Neste capítulo será realizada uma revisão bibliográfica do Problema de Alocação Dinâmica de Espaços, no qual veremos os mais importantes trabalhos e projetos publicados sobre o assunto, também detalhes sobre os principais algoritmos que ajudaram a construir uma boa solução para este problema.

Em [2] o problema é definido de acordo com a necessidade de minimizar a distância percorrida por recursos como equipamentos, peças, ferramentas, entre outros, na operação de planejamento do desligamento em centrais de energia nuclear. É definido um modelo em programação matemática, executado no CPLEX, o qual consegue resolver apenas as instâncias menores.

Por fim, é proposta uma aplicação da meta-heurística S*imulated* A*nnealing*, que é capaz de encontrar boas soluções.

Em [3] são propostos cinco novos algoritmos de construções e uma aplicação da meta-heuristica Busca Tabu. Neste trabalho foi acrescentado ao problema uma nova restrição que obriga a associar ao mesmo depósito, recursos que permaneçam ociosos por períodos consecutivos de tempo. Esta nova restrição traz, como principal razão, a diminuição de movimentos de recursos ociosos, já que movimentos desnecessários geram custos no projeto final.

Em [18] é proposto um novo modelo em programação matemática mais geral que considera a minimização da realocação dos recursos. Como custos de realocação são considerados custos de transporte e de preparação dos recursos. Neste trabalho, a imposição de não haver movimentos de recursos ociosos que ocorrem em períodos consecutivos, feita em [3], não é mantida. Além disso, o trabalho apresenta três diferentes heurísticas baseadas em Busca Tabu. Os resultados reportados apontam a heurística que implementa intensificação e diversificação como o melhor método.

Em [20] um método de construção e um algoritmo híbrido baseado em GRASP e Busca Tabu (HGT) são propostos. São feitas comparações entre os resultados da literatura e o HGT, mostrando que o HGT é capaz de obter soluções de melhor qualidade.

O presente trabalho enfoca uma nova proposta baseada na construção de um grafo de dependências entre as atividades, que será utilizado na fase de construção do GRASP e explora estruturas de vizinhanças já conhecidas da literatura para a fase de busca local, principalmente as utilizadas em [20].

Por esta razão, não podemos deixar de fora deste trabalho, uma explicação do modelo em programação matemática do Problema da Alocação Dinâmica de Espaços.

3.1 Formulação em Programação Matemática

Na definição formal do PADE, os seguintes índices e parâmetros necessários devem ser considerados:

Índices:

J = conjunto de atividades (j = 1, 2, ..., |J|);

R = conjunto de recursos (r = 1, 2, ..., |R|);

P = conjunto de períodos (p = 1, 2, ..., |P|);

N = número total de locais do *layout* (espaços de trabalho e depósitos);

L = conjunto de locais do *layout*, |L| = N (k, l = 1, 2,..., N);

W = conjunto de espaços de trabalho, $W \subset L$ ($w \in W$);

S = conjunto de depósitos, $S \subset L$ ($s \in S$) e $W \cup S = L$;

Rjp = conjunto de recursos necessários para realizar a atividade j no período p;

Ip = conjunto de recursos ociosos no período p;

Ap = conjunto de atividades no período p.

Parâmetros:

d_{kl} = distância entre os locais k e l;

C_s = capacidade do depósito s.

Variáveis de Decisão:

$x_{rkp} = 0 \text{ ou } 1, \forall r \in R, \forall k \in L, \forall p \in P,$

$y_{jw} = 0 \text{ ou } 1, \forall j \in J, \forall w \in W.$

O modelo a seguir foi definido por [2], como já mencionado anteriormente.

Minimizar:

$$\sum_{r=1}^{|R|}\sum_{k=1}^{N}\sum_{l=1}^{N}\sum_{p=1}^{|P|-1} d_{kl}\, x_{rkp}\, x_{rl(p+1)} \tag{1}$$

Sujeito a:

$$\sum_{s\in S} x_{rsp} = 1, \forall r \in Ip, \forall p \in P, \tag{2}$$

$$\sum_{r\in I_p} x_{rsp} \leq \mathrm{C_s}, \forall s \in \mathrm{S}, \forall \mathrm{p} \in \mathrm{P}, \tag{3}$$

$$\sum_{w\in W} y_{jw} = 1, \forall j \in J, \tag{4}$$

$$\sum_{j\in A_p} y_{jw} \leq 1, \forall w \in W, \forall p \in P, \tag{5}$$

$$\sum_{r\in R_{jjp}} x_{rwp} = \left|Rjp\right| \mathrm{y_{jw}}, \forall j \in \mathrm{A_p}, \forall w \in \mathrm{W}, \forall \mathrm{p} \in \mathrm{P}, \tag{6}$$

$$x_{rkp} = 0 \text{ or } 1, \forall \mathrm{r} \in \mathrm{R}, \forall \mathrm{k} \in \mathrm{L}, \forall \mathrm{p} \in \mathrm{P}, \tag{7}$$

$$y_{jw} = 0 \text{ or } 1, \forall j \in \mathrm{J}, \forall w \in \mathrm{W}. \tag{8}$$

A função objetivo (1) minimiza a distância percorrida pelos recursos ao longo dos períodos em que o projeto é realizado. As restrições (2) garantem que cada um dos recursos ociosos, em cada período, seja associado a somente um depósito e as restrições (3) asseguram que a capacidade dos depósitos seja respeitada em cada período de tempo. As restrições (4) e (5) garantem, respectivamente, que toda atividade seja alocada a um único espaço de trabalho e que cada espaço de trabalho tenha no máximo uma atividade associada. As restrições (6) garantem que todos os recursos necessários à realização de uma atividade sejam associados ao mesmo espaço de trabalho ao qual a atividade tenha sido associada. Finalmente, as restrições (7) e (8) determinam que as variáveis de decisão são binárias.

Esta formulação quadrática tem sua forma linearizada substituindo-se a função objetivo (1a) pelo termo (1), onde $z_{rklp,p+1}$ é uma variável de decisão binária. Além disso, às restrições 9 – 11 devem ser acrescentadas [3].

$$\sum_{r=1}^{|R|}\sum_{k=1}^{N}\sum_{l=1}^{N}\sum_{p=1}^{|P|-1} d_{kl} z_{rklp,p+1} \tag{1a}$$

$$x_{rkp} + x_{rlp} - 1 \leq z_{rklp,p+1}\ ,\ \forall r \in R, \forall k, l \in L, l \# k, \forall p \in P, \tag{9}$$

$$x_{rkp} + x_{rlp+1} \geq 2 z_{rklp,p+1}\ ,\ \forall r \in R, \forall k, l \in L, l \# k, \forall p \in P, \tag{10}$$

$$x_{rklp+1} = 0 \text{ or } 1, \forall r \in R, \forall k, l \in L, l \# k, \forall p \in P. \tag{11}$$

3.2 Algoritmos da Literatura

Os algoritmos propostos nos trabalhos são apresentados na seção 3.2.1. Primeiramente, são descritos os algoritmos de construção e em seguida, definidos os movimentos e algoritmos baseados em meta-heurísticas que, aplicados a uma solução, tem por objetivo atingir um ótimo local.

3.2.1 Algoritmos de Construção

3.2.1.1 Algoritmo de Construção *First Assignment* 1 (FA1) [2]

Este algoritmo constrói uma solução para o PADE em duas fases: a primeira alocando atividades e a segunda, os recursos ociosos. Na primeira fase, a primeira atividade do período é associada ao primeiro espaço de trabalho disponível, ou seja, que não é ocupado por nenhuma atividade, a segunda atividade ao segundo espaço disponível e assim por diante, concluindo-se esta fase quando cada atividade tiver sido alocada a um espaço de trabalho. Se uma atividade é realizada em períodos consecutivos, ela deve ser associada ao mesmo espaço de trabalho em cada um destes períodos. Na segunda fase do algoritmo, para cada período, um subconjunto do conjunto de recursos ociosos é associado ao primeiro depósito. O número de recursos deste subconjunto é determinado pela capacidade do depósito. O segundo depósito recebe os recursos do segundo subconjunto de recursos ociosos e assim por diante até que todos eles tenham sido alocados.

3.2.1.2 Algoritmo de Construçao *First Assignment* 2 (FA2) [3]

Trata-se de uma adaptação de FA1, com modificações na fase de associação de recursos ociosos. Desta forma, a primeira fase permanece inalterada. Na segunda fase, a associação dos recursos ociosos é modificada somente para aqueles que permanecem ociosos por períodos consecutivos. Neste caso, o recurso é associado ao mesmo depósito em todos os períodos que permanecer ocioso.

3.2.1.3 Algoritmo de Construção *Randomized Clustering* (RC) [3]

Este algoritmo constrói uma solução parcial para o PADE em duas fases: A primeira segue a ideia do algoritmo de clusterização de [4], utilizado no problema de células de manufatura, para alocação de atividades. Em seguida é utilizada *Randomized Storage Policy* (RSP) para associar os recursos ociosos a depósitos, similar à proposta de [5] para associação de produtos em almoxarifados.

RC aloca atividades a espaços de trabalho da seguinte maneira: são criadas duas matrizes, uma chamada matriz Atividades-Recursos que associa cada recurso às atividades nas quais ele é utilizado e, a outra, denominada matriz de Escalonamento que associa atividades a períodos. O procedimento agrupa em *clusters* atividades que ocorrem em períodos diferentes, respeitando o limite de capacidade do espaço de trabalho. Após o processo de clusterização, em cada grupo, têm-se as atividades associadas a um espaço de trabalho.

A solução determinando a alocação dos recursos ociosos a depósitos, obtida por RSP, é feita da seguinte maneira: após se obter a completa alocação das atividades para alocar cada recurso ocioso de um período, é verificado o depósito (observada sua capacidade) mais próximo do espaço de trabalho ao qual está associada à atividade que necessita o recurso ocioso, a este depósito o recurso é alocado. Os recursos ociosos em um período que são requeridos mais cedo por atividades são associados primeiro. Além disso, é necessário alocar os recursos que não serão mais utilizados por atividades no restante dos períodos. Neste caso, o recurso é alocado ao depósito mais próximo do espaço de trabalho ao qual está associada à última atividade que o utilizou.

3.3 Estruturas de Vizinhanças

Uma vez que uma solução viável *s* esta disponível, estruturas vizinhas propostas em [2] são usadas:

- NA(s) está relacionada a associação de atividades em s, e
- NR(s) está relacionada a associação de recursos ociosos em s.
- NRA(s) está relacionada a associação de atividades e recursos ociosos em s.

3.3.1 Movimentos de Vizinhanças

Um movimento pode ser descrito por uma modificação aplicada a uma solução original *s* construindo uma solução vizinha *s'*. Para o PADE são descritos em [2,3] estruturas diferenciadas em movimentos de atividades ou de recursos ociosos, a seguir são descritas as estruturas conhecidas e adicionada mais uma vizinhança ao projeto.

3.3.2 Movimentos de Atividades

Para o movimento de atividades foram encontradas apenas três tipos de vizinhança, porém, sendo implementada neste trabalho unicamente as duas primeiras estruturas.

3.3.2.1 Movimento M1: Transfere dos espaços de trabalho duas ou mais atividades dos períodos consecutivos que sejam alocadas (Tabela 3.1).

Tabela 3.1 Exemplo do movimento M1

Período	Espaço de Trabalho		
	ES_1	ES_2	ES_3
1		**A_1 (6,7)**	
2	**A_2 (2,3)**	**A_1 (6,7)**	
3	**A_2 (2,3)**		
4	**A_2 (2,3)**		
5	**A_2 (2,3)**	**A_3 (4)**	A_4 (1,8)
6		**A_3 (4)**	A_4 (1,8)
7			A_4 (1,8)
8	A_5 (1,5)		
9			A_6 (5,8)
10			A_6 (5,8)

Com este exemplo pode perceber que é possível encontrar uma nova solução permutando os blocos de atividades, ou seja, as atividades A_1 e A_3 serão movidas para o espaço de trabalho ES_1 e a atividade A_2 será movida para o espaço de trabalho ES_2.

3.3.2.2 Movimento M2: É realocado uma atividade em um espaço de trabalho que esteja disponível por todos os períodos necessários para aquela atividade (Tabela 3.2).

Tabela 3.2 Exemplo do movimento M2

Período	Espaço de Trabalho		
	ES_1	ES_2	ES_3
1			**A_1 (6,7)**
2		A_2 (2,3)	**A_1 (6,7)**
3		A_2 (2,3)	
4		A_2 (2,3)	
5	A_3 (4)	A_2 (2,3)	A_4 (1,8)
6	A_3 (4)		A_4 (1,8)
7			A_4 (1,8)
8	A_5 (1,5)		
9			A_6 (5,8)
10			A_6 (5,8)

Será realizada uma troca no espaço de trabalho da atividade A_1 para o novo espaço de trabalho ES_3.

3.3.2.3 Movimento M3: É a combinação do movimento M1 e M2, o qual troca duas atividades por outras duas atividades em locais de trabalho diferentes e períodos consecutivos (Tabela 3.3).

Tabela 3.3 Exemplo do movimento M3

Período	Espaço de Trabalho		
	ES_1	ES_2	ES_3
1	A_1 (6,7)		
2	A_1 (6,7)	A_2 (2,3)	
3		A_2 (2,3)	
4		A_2 (2,3)	
5	**A_4 (1,8)**	A_2 (2,3)	**A_3 (4)**
6	**A_4 (1,8)**		**A_3 (4)**
7	**A_4 (1,8)**		
8			**A_5 (1,5)**
9	**A_6 (5,8)**		
10	**A_6 (5,8)**		

Este movimento transfere as atividades A_3 e A_5 para o espaço de trabalho ES_3 e as atividades A_4 e A_6 para o local ES_1. Vale ressaltar que o movimento M3 não foi implementado neste trabalho e está sendo exposto apenas como demonstração das estruturas de vizinhança descritas em outros trabalhos.

Como solução inicial para gerar todas estas vizinhanças temos a Tabela 3.4 que descreve a agenda das atividades.

Tabela 3.4 Solução original

Período	Espaço de Trabalho		
	ES_1	ES_2	ES_3
1	A_1 (6,7)		
2	A_1 (6,7)	A_2 (2,3)	
3		A_2 (2,3)	
4		A_2 (2,3)	
5	A_3 (4)	A_2 (2,3)	A_4 (1,8)
6	A_3 (4)		A_4 (1,8)
7			A_4 (1,8)
8	A_5 (1,5)		
9			A_6 (5,8)
10			A_6 (5,8)

3.3.3. Movimento de Recursos Ociosos

Para os movimentos de recursos ociosos foram eleitas, para o problema, cinco estruturas de vizinhanças, sendo implementadas apenas no algoritmo proposto para este trabalho, os motivos serão descritos a seguir.

3.3.3.1 Movimento M4: Uma troca dos depósitos dos recursos ociosos por apenas um período (Tabela 3.5).

Tabela 3.5 Exemplo do movimento M4

Período	Depósito		
	D_1	D_2	D_3
1	***1*** 5	2 3	***4*** 8
2	4 5		1 8
3	4 5 7	6	1 8
4	4 5 7	6	1 8
5	5 7	6	
6	5 7	6 2 3	
7	4 5 7	6 2 3	
8	4 7	6 2 3	1 8
9	4 7	6 2 3	1
10	4 7	6 2 3	1

Na Tabela 3.5 é visto um exemplo do movimento M4 onde dois recursos apenas por um período de tempo, são realocados para novos depósitos. O recurso ***4*** no primeiro período vai para o local D_3 e para o ***1*** é destinado o movimento inverso, saindo do depósito D_3 para ser associado ao D_1.

O movimento M4 não foi incluído no algoritmo deste trabalho, pois ele desconsidera a última suposição sobre o PADE incluída por [3], a qual diz que um recurso ocioso deve ser associado ao mesmo depósito por todos os períodos consecutivos que permanecer ocioso para evitar uma movimentação desnecessária e gerando mais custo para o projeto, contrariando o objetivo do problema que é de minimizar este valor.

3.3.3.2 Movimento M5: É deslocado um recurso para novo depósito que não esteja com sua capacidade máxima por apenas por um único período (Tabela 3.6).

Tabela 3.6 Exemplo do movimento M5

Período	Depósito		
	D_1	D_2	D_3
1	4 5	***1*** 2 3	8
2	4 5		1 8
3	4 5 7	6	1 8
4	4 5 7	6	1 8
5	5 7	6	
6	5 7	6 2 3	
7	4 5 7	6 2 3	
8	4 7	6 2 3	1 8
9	4 7	6 2 3	1
10	4 7	6 2 3	1

É exposto na Tabela 3.6 o deslocamento do recurso ***1*** do depósito D_3 para o D_2, contudo, pelos mesmos motivos do movimento M4, também não foi implementada esta estrutura de vizinhança.

3.3.3.3 Movimento M6: Troca depósitos de dois ou mais recursos por períodos consecutivos (Tabela 3.7).

Tabela 3.7 Exemplo do movimento M6

Período	Depósito		
	D_1	D_2	D_3
1	5	***4*** 2 3	1 8
2	5	***4***	1 8
3	***6*** 5 7	***4***	1 8
4	***6*** 5 7	***4***	1 8
5	***6*** 5 7		
6	***6*** 5 7	2 3	
7	***6*** 5 7	***4*** 2 3	
8	***6*** 7	***4*** 2 3	1 8
9	***6*** 7	***4*** 2 3	1
10	***6*** 7	***4*** 2 3	1

Observa-se na Tabela 3.7 uma demonstração do movimento M6, onde o recurso ***6*** é realocado no depósito D_1 e obrigando o recurso 4 por dois conjuntos de períodos consecutivos a ser associado ao D_2.

3.3.3.4 Movimento M7: Realocação de um recurso ocioso por uma sequência de períodos consecutivos para um depósito que não tenha chegado a sua capacidade máxima.

Tabela 3.8 Exemplo do movimento M7

Período	Depósito		
	D_1	D_2	D_3
1	4	2 3	1 8 ***5***
2	4		1 8 ***5***
3	4 7	6	1 8 ***5***
4	4 7	6	1 8 ***5***
5	7	6	***5***
6	7	6 2 3	***5***
7	4 7	6 2 3	***5***
8	4 7	6 2 3	1 8
9	4 7	6 2 3	1
10	4 7	6 2 3	1

Na Tabela 3.8 é evidenciado o movimento M7 onde o recurso ***5*** é transferido do depósito D_1 para o D_3 o qual estava vazio por todo o período consecutivo requerido necessário para ocorrer esta movimentação.

3.3.3.5 Movimento M8: É a união dos movimentos M6 e M7 (Tabela 3.9)

Tabela 3.9 Exemplo do movimento M8

Período	Depósito		
	D1	D2	D3
1	4 ***2***	***5*** 3	1 8
2	4	***5***	1 8
3	4 7	6 ***5***	1 8
4	4 7	6 ***5***	1 8
5	7	6 ***5***	
6	***2*** 7	6 ***5*** 3	
7	4 ***2*** 7	6 ***5*** 3	
8	4 ***2*** 7	6 3	1 8
9	4 ***2*** 7	6 3	1
10	4 ***2*** 7	6 3	1

A Tabela 3.9 é uma demonstração do movimento M8 onde os recursos ***2*** e ***5*** são trocados de depósitos, assim teremos uma movimentação com o mesmo principio que a anteriormente citada M6, por este argumento, a vizinhança M8 é transformada na estrutura M6 para fins de implementação do algoritmo deste trabalho.

A solução original que serviu como base para ser manipulada por todas as estruturas de vizinhanças para recursos ociosos descritos anteriormente, é apresentada na Tabela 3.10.

Tabela 3.10 Solução original

Período	Depósito								
	D_1			D_2			D_3		
1	4	5			2	3	1	8	
2	4	5					1	8	
3	4	5	7	6			1	8	
4	4	5	7	6			1	8	
5		5	7	6					
6		5	7	6	2	3			
7	4	5	7	6	2	3			
8	4		7	6	2	3	1	8	
9	4		7	6	2	3	1		
10	4		7	6	2	3	1		

3.4 Instâncias do Problema da Alocação Dinâmica de Espaços

Para realizar testes uniformes sobre os algoritmos do PADE, foram criadas noventa e seis instâncias diferentes por [2,3] e disponibilizadas por [1].

Este conjunto de instâncias possuem características que podem ser descritas pelas seguintes variáveis.

N: número de locais possíveis em um *layout*, N= {6, 12, 20, 32}, sendo a metade deste número reservado para espaços de trabalho e a outra parte destinados a depósitos.

P: número de períodos representado por P = {10, 15, 20}.

A: número de atividades, A = {6, 9, 10, 12, 13, 14, 15, 16, 18, 20, 21, 23, 24, 25, 31, 33, 34, 37, 38, 40, 41, 51, 52, 55, 57, 60, 78, 87}.

3.4.1 *Layout*

Para melhor detalhar o *layout* das instâncias do PADE, abaixo foi demarcado o conjunto N de quatro números de locais diferentes anteriormente descritos na seção 3.4. Instâncias com 6 espaços têm *layout* 2x3, divididos por 3 locais reservados para espaços de trabalho e 3 locais reservados para depósitos. Instâncias com 12 locais têm *layout* 2x6, destinados 6 locais para espaços de trabalho e 6 locais destinados para

depósitos. Instâncias com 20 locais têm *layout* 4x5 divididos em 4 linhas sendo a primeira e a quarta destinadas para depósitos e as linhas 2 e 3 para locais de trabalho. Instâncias de 32 locais têm *layout* 4x8 onde a primeira e a quarta linha de espaços são destinadas a depósitos e as duas linhas intermediárias (3 e 4) são reservadas para os locais de trabalho.

Com a Tabela 3.11 é possível ver a representação do *layout* 2x3 que é dividido por duas linhas, a primeira linha determinada pelos locais 0, 1 e 2 sendo um conjunto de espaços de trabalho e na segunda linha teremos os locais 3, 4 e 5 como depósitos disponíveis.

Tabela 3.11 Representação do *Layout* 2x3

ES_1	ES_2	ES_3
D_1	D_2	D_3

Para este *layout* coincidem os locais "lógicos" citados acima com os locais "reais" da matriz de distâncias. Esta diferença se faz visível a partir das instâncias com 20 e 32 locais. Na Tabela 3.12 pode ser visto a representação dos números de cada local real para o *layout* 2x3.

Tabela 3.11 Representação dos Locais no *Layout* 2x3 para a Matriz Distâncias

0	1	2
3	4	5

A matriz distância como já dito é um dos dados de entrada do problema e seu *layout* se adapta de acordo com o tamanho da instância, representado pelo número de locais.

Para o *layout* 2x6 de 12 localidades, é representado pela Tabela 3.13 na qual a primeira linha são locais de trabalho e a segunda foi destinada para depósitos.

Tabela 3.13 Representação dos Locais no *Layout* 2x6

ES_1	ES_2	ES_3	ES_4	ES_5	ES_6
D_1	D_2	D_3	D_4	D_5	D_6

Também a exemplo do *layout* 2x3, os índices locais do *layout* 2x6 correspondem aos índices da matriz de distâncias (Tabela 3.14).

Tabela 3.14 Representação dos Locais no *Layout* 2x6 para a Matriz Distâncias

0	1	2	3	4	5
6	7	8	9	10	11

A partir de instâncias com o número de locais maior ou igual a 20, é notada uma diferença entre os índices da matriz de distâncias e os índices dos locais lógicos do *layout*.

Com o *layout* 4x5 representado na Tabela 3.15 pode-se ver um acréscimo de duas linhas.

Tabela 3.15 Representação dos Locais no *Layout* 4x5

D_1	D_2	D_3	D_4	D_5
ES_1	ES_2	ES_3	ES_4	ES_5
ES_6	ES_7	ES_8	ES_9	ES_{10}
D_6	D_7	D_8	D_9	D_{10}

Os índices da matriz distância continuam com a mesma lógica das instâncias de 6 e 12 localidades, onde os primeiros locais da matriz são os espaços de trabalho e ao fim dos índices destes locais, são considerados os depósitos. Isto pode ser justificado para a utilização do modelo em programação matemática de uma forma genérica, desconsiderando o *layout* ou a instância que esta sendo executada sendo que esta diferença pode ser vista na Tabela 3.16.

Tabela 3.16 Representação dos Índices dos Locais na Matriz de Distâncias no *Layout* 4x5

10	11	12	13	14
0	1	2	3	4
5	6	7	8	9
15	16	17	18	19

Já para o *layout* 4x8, que possui o maior número de locais de todas as instâncias, são adicionado mais três colunas totalizando 32 locais com 16 locais de trabalho e 16 depósitos (Tabela 3.17).

Tabela 3.17 Representação dos Locais no *Layout* 4x8

D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8
ES_1	ES_2	ES_3	ES_4	ES_5	ES_6	ES_7	ES_8
ES_9	ES_{10}	ES_{11}	ES_{12}	ES_{13}	ES_{14}	ES_{15}	ES_{16}
D_9	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}	D_{16}

Que seguindo a mesma regra do *layout* 4x5 é diferente os índices lógicos dos índices reais da matriz de distâncias que pode ser observado na Tabela 3.18.

Tabela 3.18 Representação dos Índices dos Locais na Matriz Distâncias no *Layout* 4x8

16	17	18	19	20	21	22	23
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
21	22	23	24	25	29	30	31

3.4.2 Arquivos de entrada para o PADE

As diferenças entre os locais lógicos e reais das instâncias são percebidas graças aos arquivos de entrada destas instâncias os quais são detalhados a seguir.

3.4.2.1 Arquivo DIST.cpp

Este arquivo contém um conjunto de números que representa uma matriz, onde cada célula representa a distância entre os locais *i* e *j* acessados através da coluna *i* e da linha *j* ou inversamente, porém vale ressaltar que a primeira linha desta matriz, representa todas as distâncias do local 0 que é o primeiro local de trabalho para todos os outros no *layout*. Outra característica desta matriz é que sua diagonal principal é nula e o número de linhas é idêntico ao número de colunas.

Tabela 3.19 Matriz Distância com 6 Locais

0	1	2	1	2	3
1	0	1	2	1	2
1	2	0	3	2	1
1	2	3	0	1	2
2	1	2	1	0	1
3	2	1	3	1	0

Na Tabela 3.19 pode ser visto um exemplo da matriz de distâncias de uma instância de 6 localidades, na qual podem ser vistas todas as características citadas acima onde pode ser encontrada no arquivo DIST.cpp dentro da pasta correspondente ao número da instância.

3.4.2.2 Arquivo PERI.cpp

Para este arquivo foi definida a função de guardar as agendas das atividades, assim, é determinado o tempo de execução de todas as atividades do projeto, como uma matriz de apenas duas colunas que na primeira célula de cada linha possui o período de inicio correspondente a atividade com o número da linha e o segundo valor refere-se ao período final, com isso, esta matriz tem a quantidade de linha igual ao número total de atividades.

Tabela 3.20 Representação do arquivo PERI.cpp

1	2
2	5
4	6
5	7
7	9
9	10

A Tabela 3.20 é a expressão do arquivo PERI.cpp da primeira instância, na qual é possível enumerar 6 atividades onde a primeira A_1 é representado pela primeira linha com os períodos de início e fim 1 e 2 respectivamente.

3.4.2.3 Arquivo RES.cpp

Com este arquivo é possível montar o conjunto de recursos que cada atividade irá requerer para sua execução. Este número pode ser no máximo 3 e mínimo 1, da mesma forma que é tratado o arquivo PERI.cpp onde cada índice da linha é associado a uma atividade, ou seja, a primeira linha corresponde a atividade A_1 a segunda assemelha-se a atividade A_2 e assim sucessivamente.

Tabela 3.21 Conjunto de Recursos das Atividades

6	7	0
1	5	0
4	0	0
2	8	0
3	4	0
9	0	0

Com a Tabela 3.21 podem ser extraídos todos os conjuntos de recursos das atividades, para a primeira linha correspondente a atividade A_1 são reservados os recursos 6 e 7 e o recurso inexistente 0 para completar a capacidade máxima. Juntamente com o arquivo PERI.cpp, o arquivo RES.cpp constroem toda a agenda de atividades necessária para a construção da solução completa do PADE.

3.4.2.4 Arquivo Other.txt

O arquivo Other.txt representa apenas um repositório de variáveis daquela instância, nele são descriminados os números correspondentes ao total de recursos disponíveis, a capacidade máxima dos depósitos, o número de locais disponíveis para o *layout*, o total de atividades daquele projeto e a quantidade de períodos necessários para todas as execuções, e são apresentados nesta ordem.

Tabela 3.22 Conjunto de Variáveis Apresentado pelo Arquivo Other.txt

9	3	6	6	10

Por fim é feito um balanço das variáveis de toda a instância como é possível ver na Tabela 3.22 correspondente a instância 1 do conjunto conhecido.

3.5 Representação de uma Solução para o PADE

Para representar uma solução do PADE foram descritas 3 matrizes por [1], e adotadas neste trabalho.

1. A matriz $A_{|P|x|W|}$, sendo a_{pw} a variável responsável por armazenar o número da atividade alocada ao espaço de trabalho w no período p, para [1] os índices das atividades iniciam com 1 e 0 para a demarcar que o local de trabalho estará livre, porém, neste trabalho foi redefinida a primeira atividade como A0 e um valor negativo para a ausência da mesma no local de trabalho para facilitar na programação.
2. A segunda matriz é representada por $R_{|P|x(E)}$, sabendo que r_{pc_s} é o recurso ocioso que está associado em uma posição entre 1 e c_s do depósito *s* e no período *p*. Caso não haja recurso alocado em c_s de *s* no período *p* é representado por 0.
3. Por fim temos uma matriz $L_{|P|x|R|}$, onde l_{pr} guarda o número do local em que está associado o recurso *r* no período *p*.

Uma melhor compreensão destas estruturas são vistas nas tabelas 2.4, 2.5 e 3.23 representando a matriz A, R e L respectivamente.

Tabela 3.23 Representação da Matriz L

Período	**Recursos**								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	0	3	4	4	0	2	2	3	5
2	0	3	1	1	0	5	5	3	5
3	3	0	4	4	3	5	5	0	5
4	3	0	4	4	3	2	5	0	2

Vale ressaltar que os locais representados nas células da matriz L devem ser os mesmos da matriz de distâncias, pois é com essa estrutura que será realizado o cálculo do custo da solução.

Como esta matriz possui toda informação de uma solução da instância, também pode ser identificados erros como: recursos que em um determinado período deveriam estar em um depósito, por serem ociosos naquela ocasião, e estão em um local de trabalho e vice-versa; ou ainda, associar na matriz L índices de locais lógicos do *layout* onde deveriam aparecer índices reais da matriz distância, acarretando erros ao calcular o custo total da solução em instâncias acima de 20 localidades em que haja diferença entre esses valores.

3.6 Meta-heurísticas

Uma meta-heurística pode ser definida como um processo iterativo que guia uma heurística subordinada pela combinação de conceitos capazes de explorar e valorizar o espaço de busca, utilizando o aprendizado com a estrutura do problema para que se possa chegar, de forma eficiente, a soluções próximas do ótimo [6]. As heurísticas apresentadas mais à frente utilizam os movimentos apresentados na seção anterior e os conceitos das meta-heurísticas Simulated Annealing e Busca Tabu para o melhoramento de soluções iniciais construídas para o PADE. A seguir é feita uma breve descrição das metaheurísticas Simulated Annealing e Busca Tabu.

3.6.1 A meta-heurística Têmpora Simulada

Em [2] foram apresentados duas variantes da meta-heurística SA (Simulated Annealing) para o PADE.

A meta-heurística SA inicia com uma solução inicial y^0 chamada solução corrente e seu custo de transporte TC (y^C) é obtido. A cada iteração uma solução vizinha da corrente é obtida pela realização de um movimento. Esta solução é denominada y'. Se o custo da solução vizinha é menor do que o custo da solução corrente, então a solução vizinha é selecionada como a solução corrente na próxima iteração, caso contrário, a solução vizinha é selecionada como uma solução corrente para próxima iteração com relação à probabilidade de aceitação.

A cada iteração, o custo e a melhor solução são salvos e atualizados se necessário. A heurística é repetida para um determinado numero de iterações ou até que o critério de parada seja encontrado.

Uma solução inicial para o WAP (*Workspace Allocation Problem*) é gerada associando a primeira atividade ao primeiro espaço de trabalho, a segunda atividade ao segundo espaço de trabalho e assim sucessivamente, no primeiro período. No segundo período, se uma ou mais atividades são realizadas no primeiro período, associa-se estas atividades ao mesmo espaço de trabalho do primeiro período. As outras atividades são associadas ao primeiro espaço de trabalho disponível. Este processo é repetido até que todas as atividades em cada período estejam associadas aos espaços de trabalho.

Para o WAP, um movimento é definido como segue:

1. Trocam-se os locais (espaços de trabalho) para 2 atividades em um ou mais períodos.

2. Em um ou mais períodos, remove-se uma atividade de um local (espaço de trabalho) e associa-se ela a um local disponível (espaço de trabalho vazio).

3. Uma combinação dos passos 1 e 2.

Para o SAP um movimento é definido como segue:

1. Troca-se as locais (espaços de armazenamento) para 2 recursos associados a diferentes espaços de armazenamento em um dos períodos.

2. Em um dos períodos, remove-se um recurso de um local (espaço de armazenamento) e associa-se ele a um local diferente (espaço de armazenamento) na qual a capacidade não foi atingida).

A probabilidade de aceitação é definida como a probabilidade de aceitar uma solução vizinha não melhorada como a solução corrente para a próxima iteração e é definida como:

$$P(\Delta TC) = e^{-\Delta TC/T_c}$$

$$T_c = T_0 \alpha^{r-1} \text{ para r=1, 2,... R}$$

Onde:

$$\Delta TC = TC(\gamma^c) - TC(\gamma')$$

T_c representa a temperatura corrente, T_0 é a temperatura inicial, r-1 é o número de reduções e α é chamada a taxa de resfriamento e é usualmente fixada com o valor 0.90 como em [16,17].

A Figura 3.1 apresenta o pseudocódigo de SA1. Inicialmente, o algoritmo recebe os seguintes parâmetros: T_0, a temperatura inicial; α, a taxa de resfriamento; SAmax, o número de iterações para cada temperatura; Max_Iter, o número máximo de iterações consecutivas sem melhora; p, a probabilidade de realizar um movimento de recursos ociosos e a solução inicial y0, obtida através do algoritmo de construção FA1.

Heurística SA1(T_0, α, SAmax,Max_Iter, p, y0)

1: r ← 1; i ← 0;

2: yc ← y0; TC(yc) ← TC(y0);

3: Melhor_Sol ← yc; Melhor_Custo ← TC(yc);

4: enquanto i < Max_Iter faça

5: j ← 0; /*Contador do número de movimentos de cada temperatura*/

6: Tc ← T0αr−1;

7: enquanto j < SAmax faça

8: a) Escolher aleatoriamente t e movimento, probabilidade p para recursos ociosos;

b) Gerar um vizinho y0 de yc;

9: j ← j + 1;

10: Calcular f(y0);

11: Calcular Δ ← f(y0) − TC(yc);

12: se Δ < 0 então

13: yc ← y0;

14: TC(yc) ← f(y0);

15: se Melhor_Custo > TC(yc) então

16: Melhor_Sol ← yc;

17: Melhor_Custo ← TC(yc);

18: m se

19: se não

20: x ← random(0,1);

21: se x < e−ΔTC/Tc então

22: TC(yc) ← f(y0);

23: i ← i + 1;

24: m se

25: m se

26: m enquanto

27: r ← r + 1;

28: m enquanto

29: Retornar Melhor_Sol e M Melhor_Custo;

end

Figura 3.1. Algoritmo SA I.

A segunda heurística SA desenvolvida (SA II) realiza todos os passos da heurística SA I. Entretanto a geração da solução inicial (para o SAP) e o movimento SAP são diferentes dos da heurística SA I. A geração da solução inicial para o WAP na heurística SA II é a mesma para a heurística SA I. Quando gerando uma solução inicial para o SAP no SA II, o primeiro conjunto de recursos livres no período 1 é associado ao primeiro espaço de armazenamento. O número de recursos ociosos no conjunto é determinado pela capacidade do primeiro espaço de armazenamento. Então o próximo conjunto de recursos ociosos é associado ao segundo espaço de armazenamento e assim por diante, no período 1. No período 2, se um ou mais dos recursos ociosos no período 2 estão também ociosos no período 1, associa-se estes recursos ociosos ao mesmo espaço de armazenamento como no primeiro período. Os outros recursos ociosos são associados ao primeiro espaço de armazenamento disponível. Este processo é repetido até que os recursos ociosos em cada período estejam associados aos espaços de armazenamento.

3.6.2 A meta-heurística Busca Tabu (BT)

A busca tabu foi introduzida por [7,19]. Em [8] podemos encontrar o primeiro trabalho a apresentar a meta-heurística Busca Tabu para o Quadratic Assignment Problem (QAP). Em [9] os autores apresentaram a única metaheurística Busca Tabu para o DFLP. Uma vez que esta abordagem funcionou bem para o DFLP, que é similar ao PADE, ela foi aplicada neste.

Em uma solução inicial a heurística BT explora todos os movimentos candidatos possíveis na vizinhança da solução corrente usando trocas de pares e a melhor solução vizinha é considerada com relação ao melhor valor da função objetivo, mesmo se os resultados são piores que a solução corrente. Para prevenir ciclagem, os movimentos recentes são proibidos para certo número de iterações. O número de movimentos de iterações tabu é chamado duração tabu. A lista de movimentos recentes e sua duração tabu são mantidas em uma lista tabu. Entretanto, se um movimento que dá a melhor solução já esta na lista tabu, então a restrição tabu é substituída e o movimento é realizado. A condição que permite que a restrição tabu colocada em um movimento seja substituída é chamada de critério de aspiração. Entretanto o melhor movimento permitido selecionado ou é um movimento tabu que dá a melhor solução ou é o melhor movimento que não é classificado como tabu (não tabu). A solução obtida depois de

realizar o melhor movimento permitido é definida como solução corrente na próxima iteração e o processo é repetido até que o critério de parada seja atingido. Para cada iteração a melhor solução encontrada e seu custo são salvas ou atualizadas se necessário.

Há 2 tipos de movimentos: atividade e recurso ocioso. Em um movimento de atividade é importante que a suposição 7 (capítulo 2), seja imposta para manter a viabilidade, portanto, uma atividade de movimento é definida como segue:

1. Trocam-se os locais (espaços de trabalho) de 2 ou mais atividades nos períodos que as atividades são realizadas;

2. Remove-se uma atividade de um espaço de trabalho e esta é associada a um espaço de trabalho vazio durante os períodos em que a atividade é realizada;

3. Combinação dos passos 1 e 2.

Quando realizando um movimento de recurso livre, é importante que a suposição 10 (capítulo 2) seja imposta, o que reduz o espaço de soluções e o número de movimentos de recursos ociosos. O movimento de recursos ociosos é definido como abaixo:

1. Troca de locais (espaços de armazenamento) de 2 ou mais recursos ociosos associados a diferentes espaços de armazenamento em períodos consecutivos que eles estejam ociosos;

2. Remove-se um recurso ocioso de um local de armazenagem e associa-se ele a um espaço disponível durante períodos consecutivos em que ele esta ocioso;

3. Combinação dos passos 1 e 2.

3.6.3 Meta-heurística Híbrida GRASP e Busca Tabu (HGT)

Este trabalho apresenta um algoritmo de construção e uma heurística híbrida baseada em metaheurísticas GRASP e Busca Tabu, em conjunto e também implementadas de forma isolada. Estes algoritmos têm alcançado bons resultados para um número de problemas de otimização combinatória [3,13] e a combinação destas ideias em heurísticas híbridas têm sido proveitosa para problemas difíceis, tais como o Problema Quadrático de Alocação (PQA).

O algoritmo de construção proposto tem duas fases. A primeira destina-se a alocar as atividades para os espaços de trabalho e, na segunda, recursos ociosos são alocados para os depósitos. Na primeira fase, é considerado o conjunto J de atividades e

o conjunto W de espaços de trabalho. Em cada iteração, uma Lista de Candidatos Inicial (LCI) é elaborada com os melhores candidatos a serem considerados para inclusão na solução.

Na primeira fase, cada solução viável é iterativamente construída como se segue. Primeiro, uma atividade a ϵ J é aleatoriamente associada com um espaço de trabalho. Em cada iteração, uma Lista Restrita de Candidatos (LRC) é obtida a partir da LCI restante, e um par (espaço de trabalho, atividade) é escolhido aleatoriamente a partir de LRC e incluídos na solução. LCI contém as atividades ainda não associadas, pareadas com os espaços de trabalho disponíveis. Para cada par, o custo da inserção na solução inicial é calculada. LCI é então classificada em ordem decrescente de custo e uma parte da lista contendo os melhores elementos irá compor a LRC. O valor deste percentual é dado por α x min(n_av, |W|), onde n_av é o número de espaços de trabalho disponíveis e $\alpha \epsilon$ [0,1] é o parâmetro do algoritmo GRASP.

A segunda fase corresponde à política de armazenamento randomizada (PAR) proposta em [3], mas adaptada para levar em consideração a associação de recursos feita no último período.

Tal política associa recursos ociosos a depósitos da seguinte forma: após obter uma completa alocação de atividades, cada recurso ocioso é alocado para o depósito mais próximo da atividade que o exigirá. Os primeiros recursos ociosos a serem alocados são aqueles que primeiro serão solicitados. Além disso, um recurso que irá permanecer ocioso durante os períodos restantes será alocado para o depósito mais próximo do espaço de trabalho onde a última atividade que o utilizou foi alocada.

No primeiro período, esta política inicializa a alocação de recursos ociosos para depósitos como em [3], mas, em seguida, a alocação para o período posterior é feita observando a alocação que foi efetuada para o período anterior. Então, se um determinado recurso permanece ocioso, sua anterior alocação é mantida para o novo período. Os recursos ociosos restantes recursos são alocados como em [3].

As estruturas de vizinhança que são utilizadas neste trabalho são as mesmas que foram descritas na seção 3.3.

Foi proposto um algoritmo híbrido que combina as meta-heurísticas GRASP e Busca Tabu (HGT). A fase de construção do algoritmo atribui uma solução conforme o algoritmo de construção supracitado. Na fase de busca local, todos os movimentos possíveis de atividades e recursos ociosos são executados, conforme as estruturas de

vizinhanças já citadas. O melhor movimento é executado e a solução corrente é atualizada. A busca termina quando a solução corrente não pode mais ser melhorada.

Este algoritmo híbrido usa a Busca Tabu como um passo de intensificação. Iniciando com a solução corrente, o algoritmo tabu executa os seguintes passos até um número fixo de iterações consecutivas sem melhoras ser alcançado:

- Avaliar todos os possíveis movimentos relacionados às atividades e para cada uma, executar o algoritmo heurístico em [3], a fim de realocar os recursos ociosos de acordo com a nova matriz de atividades obtida.
- Escolher o melhor movimento.
- Atualizar a solução corrente e Lista Tabu.

Uma lista tabu dinâmica FIFO armazena os movimentos recentes realizados no conjunto de atividades. Um movimento é classificado como Tabu se ele pertence à Lista Tabu ou se trata de um movimento inverso de outro movimento pertencente à Lista Tabu. O tamanho da lista varia entre um limite inferior l_i e um limite superior l_s. O tempo durante o qual um dado movimento é considerado Tabu é determinado pelo tamanho da lista. No início e a cada η iterações sem melhora, um novo tamanho da lista é escolhido aleatoriamente dentro do intervalo $[l_{i\eta}, l_s]$.

Um critério de aspiração permite um movimento Tabu ser realizado se o custo da solução gerado pelo movimento, seguido da aplicação da heurística em [3], é melhor que a solução corrente.

No final deste processo, a solução é atualizada de acordo com o melhor valor. O resultado final é a melhor solução obtida após a execução de um número fixo de iterações.

O algoritmo HGT pode ser visualizado na Figura 3.2

Heurística HGT (ITER_GRASP, α, ITER_TABU, $l_{i\eta}$ $l_{s\eta}$ η)

1: **for** k ← 1 **until** ITER_GRASP **do**

2: Sol ← 1 constructionAlgorithm (α) ;

3: Sol ← localSearch (Sol);

4: Sol ← tabuSearch (Sol,ITER_Tabu, $l_{i\eta}$ $l_{s\eta}$ η);

5: updateSolution (Sol, bestSol);

6: **endfor**

7: return bestSol;

end

Figura 3.2. Algoritmo Híbrido GRASP/Tabu.

A Tabela 3.24 apresenta apenas as instâncias envolvendo 32 locações que correspondem aos problemas testes numerado entre 73 e 96. As colunas mostram os custos obtidos pelos respectivos algoritmos citados no cabeçalho. O algoritmo HGT possui os melhores resultados.

Tabela 3.24. Resultados Heurísticos para problemas com 32 locações

	Instância	✓ Tempo	SA (2005)	Old TS (2006)	TS II (2008)	HGT (2010)
	73	5.01	85	75	74	**71**
	74	4.08	108	102	97	**90**
	75	4.47	117	113	110	**101**
	76	3.09	160	161	156	**144**
	77	3.74	79	73	73	**70**
	78	6.27	111	118	101	**92**
	79	4.52	116	110	110	**99**
	80	3.37	182	183	175	**163**
	81	10.25	140	130	121	**109**
*	**82**	-	-	-	-	**-**
	83	13.73	200	204	192	**183**
	84	9.52	287	294	282	**266**
	85	13.21	144	135	125	**117**
	86	20.76	209	207	192	**177**
	87	17.74	205	195	193	**181**
	88	13.15	302	318	302	**276**
	89	17.19	195	175	171	**159**
	90	26.76	278	269	262	**243**
	91	15.39	293	297	284	**269**
	92	17.84	395	407	396	**371**
	93	29.59	211	195	189	**176**
	94	34.07	298	298	281	**271**
	95	27.55	332	327	318	**301**
	96	25.49	485	487	464	**436**

* Atividades executando em paralelo em alguns períodos, compartilhando o mesmo recurso, o que configura uma instância inviável.

✓ Tempo de execução em segundos.

3.6.4 Meta-heurística GRASP

Grasp (Greedy Randomized Adaptive Search Procedure) é uma metaheurística com diversos métodos construtivos, consistindo basicamente em criar uma solução inicial aleatória e se concentra em melhorá-la efetuando uma busca local, o seu principal objetivo e diferencial é explicado por suas três primeiras letras, Greedy (Gulosa), Randomized (Aleatória) e Adaptive (Adaptativa).

Historicamente foi proposta por [21] se mostrando eficiente para problemas de otimização combinatória, por ser um processo iterativo, porém cada iteração é independente da outra, considerada uma heurística do tipo multistart, sendo distinguíveis a construção de uma solução e sua busca local como diferentes fases.

Na construção são visíveis as seguintes características: iterativa, pois é construída uma solução por um elemento de cada vez; adaptativa onde a escolha do próximo elemento para esta construção tem influência das decisões anteriormente tomadas; randômica uma vez que no mesmo projeto, podem ser criadas diferentes soluções com o mesmo algoritmo; e por fim, ainda pode ser definida como gulosa em casos que se buscam sempre a melhor decisão local sem relacionar ao projeto como um todo.

Para a construção da solução, todos os elementos iniciais são considerados como possuindo o mesmo custo de alocação, por isto, às vezes se tem muitas possibilidades, justificado por estes casos, o GRASP utiliza uma lista restringindo os candidatos (LRC) que tem tamanho $p = 1 + \alpha\ (a - 1)$, sendo α um parâmetro de entrada que define o quanto guloso se torna o algoritmo e a o número total de elementos criados.

Com LRC é selecionado um elemento de forma aleatória, porém não garantindo ser a melhor escolha, assim, possibilitando o GRASP através destas escolhas aleatórias para a construção de várias soluções. Contudo, muitas vezes não são encontradas os melhores locais, com isto, se faz necessário o refinamento desta solução com uma busca local onde são melhoradas e este esquema do GRASP é repetido por um certo número de vezes que se nomeia por iterações.

A seguir teremos um pseudocódigo da meta-heurística Grasp

```
1  enquanto (condição de parada não for satisfeita), faça
2      solução = crie aleatoriamente uma solução();
3      solução = busca local(solução);
4      se solução é a melhor solução até então conhecida então
5          grave(solução);
6      fim se
7  fim Enquanto
```

Figura 3.3. Pseudocódigo da meta-heurística GRASP.

3.6.4.1 Construção de uma Solução GRASP para o PADE

Em [1] a autora mostra que a construção de uma solução para o PADE é realizado por duas partes, a associação de atividades em seus espaços de trabalho, e os recursos ociosos em seus depósitos.

Para a primeira fase é criada uma lista inicial de candidatos (LIC),sendo um candidato um par de atividades e local de trabalho disponível para o seu alojamento contendo ainda um custo que é gerado se for alocado a sua atividade no seu local de trabalho. Esta lista é restrita a uma quantia de elementos calculada por *gp + (alfa * (gg - gp)*, onde *gp* é o custo do menor candidato e *gg* é o maior custo encontrado nesta lista, para *alfa* foi destinado um valor $\in [0,1]$ que quantifica o quanto guloso se torna este algoritmo.

```
1   para p = 0 até P-1 faça
2       para a = 1 até J faça
3           para w = 1 até W faça
4               se a não foi alocada então
5                   candidato←ConstruirCandidatos(a,w);
6                   AdicionaremLIC(candidato);
7               fim se
8           fim para
9       fim para
10      valorRestricao←gp + (alfa * (gg - gp);
11      RestringirLICemLRC(valorRestricao);
12      Escolher Aleatoriamente um Candidato em LRC ();
13      Associar Atividade a no Local de Trabalho w do Candidato Eleito ();
14      Limpar Listas LIC e LRC();
15  fim para
```

Figura 3.4. Meta-heurística GRASP para a aloc. de atividades nos locais de trabalho.

Com isto é criada uma lista restrita de candidatos como LRC para a eleição aleatória de um candidato que será alocado. Este processo é refeito excluindo os candidatos que possuem as atividades que já foram escolhidas até todas as atividades serem atendidas neste período, repetindo todo o processo por todos os períodos do projeto.

Na segunda parte do algoritmo de construção, são associados os recursos ociosos do primeiro período aos depósitos mais próximos dos locais de trabalho em que serão requeridos. Ao término da alocação de todos os recursos ociosos do primeiro período, são associados com prioridade os recursos que permanecerem ociosos ao mesmo depósito do período anterior. Os novos ociosos (recursos que no período anterior estava em uma atividade e agora não é mais requerido), são alocados a depósitos que estejam próximos dos locais de trabalho que estavam as últimas atividades que lhe requereram, este processo é repetido por todos os demais períodos e ao fim, todos os recursos ociosos serão associados a um depósito.

```
1  para p = 0 faça
2      para ro = PróximoRecursoOciosos(p) faça
3          w← Local deTrabalho que irá requerir o Recursos (ro);
4          depósito← DepositoProximo(w);
5          Associar Recurso (ro,depósito);
6      fim para
7  fim para
8  para p = 1 até P-1 faça
9      para ro = Próximo Recurso Ociosos que Estava em um Depósito (p) faça
10         depósito← Deposito em que Estava (ro, p-1);
11         Associar Recurso (ro, depósito);
12     fim para
13     para ro = Próximo Recurso Ociosos que Estava em um Depósito (p) faça
14         w← Local de Trabalho que Estava (ro, p-1);
15         depósito← Deposito Proximo e Livre (w);
16         Associar Recurso (ro, depósito);
17     fim para
18 fim para
```

Figura 3.5. Algoritmo para a alocação de recursos ociosos nos depósitos.

Com a conclusão da segunda parte está criada uma solução para o projeto, abaixo é possível ver um pseudocódigo deste algoritmo de construção.

```
1  Alocar Atividades com Primeira Parte (alfa);
2  Alocar Recusos Ociosos com Segunda Parte ();
3  Calcular Custo ();
```

Figura 3.6. Pseudocódigo de uma solução inicial do PADE.

3.6.5. A Meta-heurística *Variable Neighborhood Descent* (VND)

Mais conhecida como VND esta meta-heurística se enquadra como uma busca local que tem como principal objetivo melhorar a solução inicial por uma troca sistemática de vizinhanças, distintas de outras metaheurísticas de busca local, não é determinado uma trajetória mas experimenta distintas vizinhanças da solução corrente, indo para uma solução melhor somente quando houver uma melhora, caso contrário, é experimentado outra vizinhança. Deste modo, são preservadas características da solução atual sendo criadas soluções próximas e promissoras, isto não é comumente visto em metaheurísticas que utilizam apenas uma vizinhança que podem ocorrer casos onde é restrito o desempenho da heurística se a estrutura não for adequadamente escolhida para o problema.

```
1 Selecionar um conjunto de estruturas de vizinhanças Nk, com k = 1,..., kmax
2 s = GerarSolucaoInicial();
3   enquanto critério de parada não satisfeito faça
4       k = 1
5       enquanto k <= kmax faça
6           s' = Vizinho qualquer gerado por Nk (s)
7           s'' = BuscaLocal (s')
8           se f(s'') < f(s) então
9               s = s''
10              k = 1
11          se não
12              k = k + 1
13          fim se
14      fim enquanto
15  fim enquanto
```

Figura 3.7. Algoritmo VND aplicado para o refinamento da solução inicial.

A estrutura de algoritmo do VND é basicamente criar soluções partindo de uma vizinhança eleita, esse processo de escolha é visto no inicio onde Nk representa a estrutura, ao variar k por todas as possibilidades o VND garante chegar na melhor solução local para as estruturas de vizinhanças disponíveis no projeto.

É gerado uma solução inicial *s* que doa as características para que se encontre um melhor local. Ao criar um vizinho *s'* é realizado uma busca local para melhorar a solução *s'* no qual é gerado *s''* que torna-se a nova base para o VND se seu custo for menor do que a solução original *s* reiniciando o loop da vizinhança k, caso contrário, é utilizado uma nova estrutura para realizar o processo ($k = k + 1$), sendo necessário para a finalização do processo, a busca local para todas as vizinhanças k conhecidas. A partir deste contexto, o VND garante encontrar a mínima solução local em um conjunto conhecido de estruturas de vizinhanças.

Capítulo 4

Abordagens Propostas para o PADE

Neste capítulo é proposta uma nova construção para o PADE através de um modelo em programação matemática baseado em um grafo de dependências entre atividades, combinado com a meta-heurística GRASP para a construção de uma solução de boa qualidade.

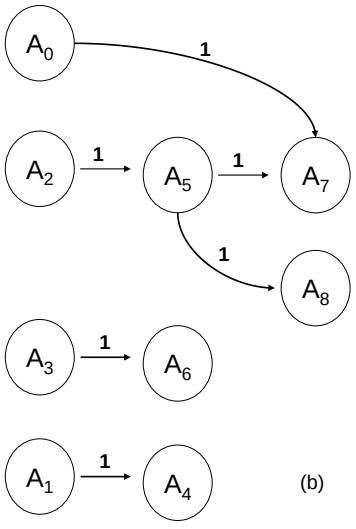
4.1 Grafo de Dependências

O grafo de dependências é uma representação das conexões entre as atividades de uma instância do PADE, sendo cada atividade representada por um vértice e a existência de um arco entre um par de vértices indica a dependência de um ou mais recursos necessários que estão em uma atividade e serão requeridos por outra.

Este grafo dirigido G (V, A) é formado por dois conjuntos, sendo um de vértices V e outro de arcos A. Cada arco descreve a ordem de precedência das atividades e seu peso w_{ij} é a quantidade de recursos que partem da atividade i e são necessários para a execução da atividade j.

Período	Atividades (recursos necessários)
1	A_0 (5 6) A_1(1)
2	A_0 (5 6) A_1(1) A_2(0 4)
3	A_1(1) A_2(0 4)
4	A_1(1) A_2(0 4) A_3 (3)
5	A_2(0 4) A_3 (3) A_4(1 7)
6	A_3 (3) A_4(1 7) A_5 (0 8)
7	A_4(1 7) A_5 (0 8) A_6(2 3)
8	A_6(2 3) A_7(0 5)
9	A_6(2 3) A_7(0 5) A_8(8)
10	A_7(0 5) A_8(8)

(a)



(b)

Figura 4.1. (a) Tabela de agendamento de atividades por período. (b) Grafo de dependências de recursos da instância.

Como exemplo, seja a agenda de um projeto com oito atividades, apresentada na Figura 4.1a. No grafo (Figura 4.1b) as atividades correspondem aos vértices. Os arcos denotam a relação de dependência entre as atividades, por exemplo, a atividade A_7 depende de dois recursos, o recurso 0, utilizado anteriormente pela atividade A_5 e o recurso 5 (vindo da atividade A_0). Porém, a atividade A_5 também possui uma dependência da atividade A_2 que coexiste com a atividade A_0 (ambas são realizadas em um mesmo período).

Observando o grafo de dependências, podemos interpretar para cada atividade *v* (vértice) o grau de entrada como a quantidade de recursos necessários para a execução da atividade *v*, que estavam anteriormente alocados a uma ou mais atividades e o grau de saída de uma atividade *v* como o conjunto de recursos que deverão ser liberados para uso futuro em outras atividades. Estes são denotadas, respectivamente, por ir_v e r_v . Por exemplo, para na atividade A_5 temos $ir_v = 1$ e $r_v = 2$.

4.1.1 Grafo de Dependências Estendido

Para delimitar as atividades que coexistem entre si, fez-se necessário a extensão do grafo de dependências G (V, A) para transformá-lo em G'= (V', A') e adicionar as seguintes modificações:

- Inclusão de dois vértices virtuais *s* e *t* para representarem os nós fonte e sumidouro da rede de fluxo, respectivamente.
- Inclusão do conjunto A^T de arcos (i, j) temporais, necessários para indicar que a atividade i não tem conflito de período com a atividade j. Assim pode-se concluir que essas atividades i e j podem coexistir no mesmo espaço de trabalho por serem realizadas durante períodos diferentes, ou seja, o período final da primeira atividade é menor do que o período de início da segunda atividade.
- Inclusão do conjunto A^V de arcos (i, j) virtuais que unem o vértice origem s a todos os vértices (atividades) $v \in V$ e estes ao vértice sumidouro t.

Devem ser consideradas ainda as seguintes definições:

$$V' = V \cup \{s,t\};$$
$$A' = A \cup A^T \cup A^V;$$
$$w_{ij} = 0 \quad \forall (i,j) \in A^V;$$
$$w_{ij} = -(ir_i + ir_j) \quad \forall (i,j) \in A^T;$$

Neste contexto, grafo de dependência estendido *G'* tem conjunto de vértices *V'* que representa a união do antigo conjunto de vértices *V* com os dois vértices virtuais {*s*, *t*}; o conjunto de arcos *A'* é a união dos conjuntos *A*, A^T e A^V; o peso de todas os arcos do conjunto A^V é 0. Por fim, temos que o peso dos arcos do conjunto A^T é igual -(ir_i + ir_j).

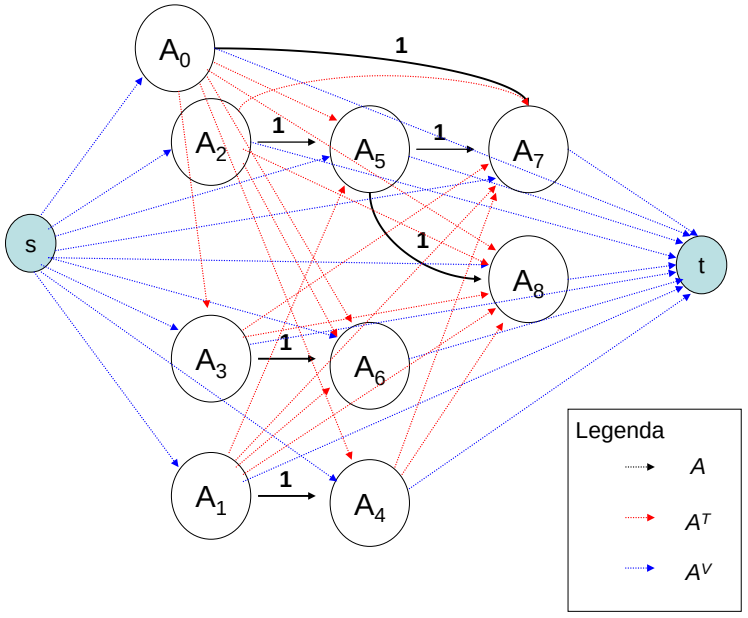


Figura 4.2. Grafo de Dependências Estendido.

O método de alocação das atividades aos espaços de trabalho corresponde à escolha dos arcos com maior soma dos valores do grafo de dependências estendido. Assim, é justificada a suposição que coloca o valor negativo dos arcos do conjunto A^T para que tenham pouca influência sobre os arcos reais do conjunto *A*, também evitando

a escolha dos arcos mais leves, ou seja, alocar atividades que possuam poucos recursos dependentes ao mesmo espaço de trabalho.

4.2 Modelo de fluxos baseado no Grafo de Dependências

Para obter uma solução de melhor qualidade durante a fase de construção, é necessária uma boa escolha dos fluxos de atividades que serão associados aos espaços de trabalho, a qual é obtida através da resolução de um modelo em programação linear inteira proposto para eleger os melhores fluxos com o auxílio do grafo de dependências estendido. Para a compreensão deste modelo considera-se a variável de decisão x_{ij} que se torna 1 para indicar que a atividade i está no mesmo local de trabalho da atividade j e 0, caso contrário.

Maximizar $$\sum_{(i,j)\in A'} w_{ij} x_{ij} \qquad (9)$$

Sujeito a:

$$\sum_{j\in N-\{s\}} x_{ij} = 1, \quad \forall i \in V-\{s,t\}, \qquad (10)$$

$$\sum_{i\in N-\{t\}} x_{ij} = 1, \quad \forall j \in V-\{s,t\}, \qquad (11)$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A'. \qquad (12)$$

Para este modelo, é interessante que se maximize a soma dos pesos dos arcos escolhidos, uma vez que este peso representa o número de recursos compartilhado entre as atividades em um mesmo local de trabalho, minimizando a movimentação de recursos, que é a ideia base do PADE.

Assim, a função objetivo (9) maximiza a soma dos pesos dos arcos formando fluxos de atividades com maior quantidade de recursos compartilhados, preservando as maiores dependências entre as atividades.

As restrições (10) garantem que cada vértice *v* possua somente um arco de saída e as restrições (11) garantem que, para cada vértice, haja somente um arco de entrada, de modo que uma atividade esteja em apenas um fluxo. As restrições (12) determinam que as variáveis de decisão são binárias.

Solução do Problema de Programação Linear Inteira (PPLI)

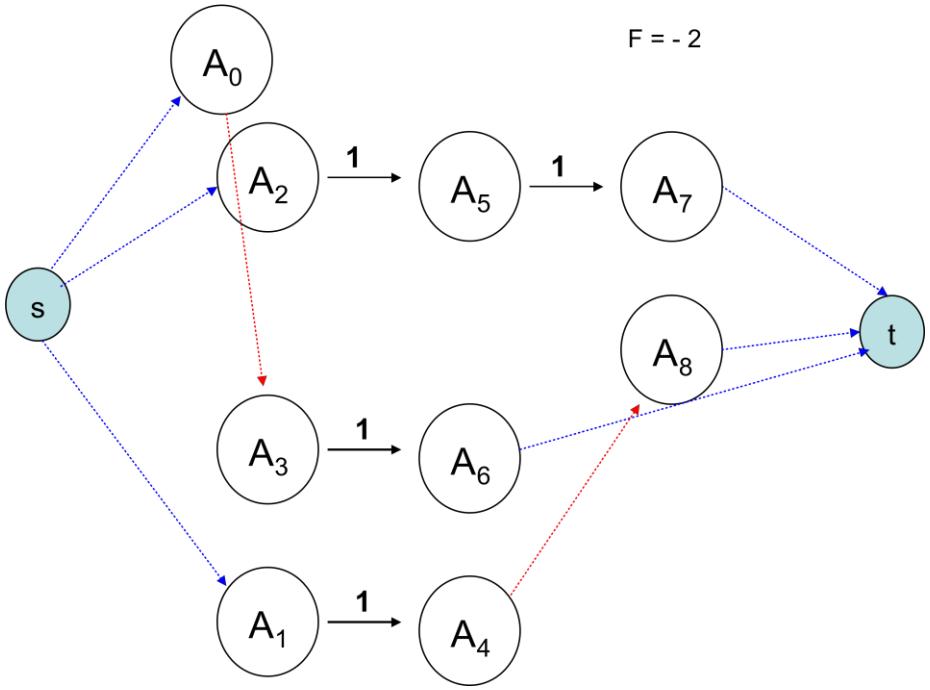


Figura 4.3 Solução do Problema de Programação Linear Inteira Sobre o Grafo de Dependências Estendido.

4.3 GRASP para Alocação dos Fluxos

Com os fluxos de atividades criados pelo modelo em programação linear inteira, é necessário agora escolher os locais de trabalho para a alocação com o auxilio do GRASP. Nesta parte do algoritmo são construídas regras para esta decisão, e ainda foi criada uma nova estrutura de vizinhança com a intenção de encontrar uma solução vizinha que melhor adeque a associação do fluxo no espaço de trabalho.

Para avaliar a dependência entre as atividades dos fluxos *f1* e *f2*, fez-se necessário a criação de um valor *gd* que considera estas ligações, esta variável aproxima os fluxos que possuem maior número de arestas interligando *f1* e *f2*. Pode ser considerado zero o *gd* de um fluxo autossuficiente, pois logicamente não existe nenhuma dependência para nenhum vértice fora os que estão em seu conjunto.

Outra métrica utilizada para a classificação do fluxo de atividades é o custo percorrido de um recurso entre os locais de trabalho do fluxo já associado e outro que ainda é um candidato para alocação. Esse valor é considerado apenas na parte de classificação dos candidatos na LIC e LRC do GRASP nomeado de *gl* que é a distância entre as dependências e o espaço de trabalho atual.

Ainda com duas variáveis a gdc e gc, sendo a primeira apenas para instâncias de 20 e 32 localidades onde existe um local de trabalho que nomeamos de *correspondente* por estar a um passo de distância do outro e estar na outra linha de locais de trabalho do *layout*, o gdc é a distância que todos os recursos terão que percorrer dos fluxos $f_{2,3,4...}$ já atendidos para o local de trabalho *correspondente* do fluxo atual f_1, esse valor tende a diminuir de acordo com a proximidade do local de trabalho correspondente dos fluxos $f_{2,3,4...}$, assim é priorizada a alocação de um fluxo no local de trabalho que mais lhe centralizar dos *correspondentes* dos fluxos que possuírem dependência de f_1. A segunda variável *gc* é a média das distâncias de todos os locais de trabalho $w_{2,3,4...}$ para o local de trabalho w_1 do fluxo que está sendo analisado, este valor tende a diminuir e a priorizar os locais de trabalho que sejam centrais nos *layouts* do PADE.

A partir de todos os valores demonstrados anteriormente, é criada uma função para calcular o peso *g* dos candidatos, onde agora um candidato é o índice de identificação do fluxo e o local de trabalho possível que possa ser associado, esse valor tem como fórmula $g = (gc + gl + gdc) - gd$ e como principal funcionalidade servir de avaliador de cada candidato para a construção de uma LRC.

O procedimento GRASP para alocação dos fluxos de atividades é semelhante ao explicado no capitulo 2 do sub-tópico 2.2.1.1 Construção de uma Solução GRASP para o PADE, baseada na construção de [1]. A diferença desta construção é a exclusão dos loops para a utilização de períodos e atividades, sendo necessário criar apenas os candidatos para os fluxos em relação ao local de trabalho, como pode ser visto no pseudocódigo a seguir.

```
1   para i = 1 até Alocar Todos os Fluxos faça
2       para f = 0 até Quantidade de Fluxos -1 faça
3           para w = 1 até W faça
4           se f não foi alocado então
5               g = (gc + gl + gdc) - gd;
6               candidato ← Construir Candidatos (f, w, g);
7               Adicionarem LIC (candidato);
8           fim se
9           fim para
10      fim para
11      valorRestricao←gp + (alfa * (gg - gp);
12      Restringir LIC em LRC (valorRestricao);
13      Escolher Aleatoriamente um Candidato em LRC ();
14      Associar as Atividades do fluxo f no Local de Trabalho w do Candidato
        Eleito();
15      Limpar Listas LIC e LRC();
16  fim para
```

Figura 4.4. Meta-heurística GRASP para a alocação de Fluxos nos Locais de Trabalho.

4.4 Alocação dos Recursos Ociosos

Após o término da fase anterior de construção temos todas as atividades alocadas a algum local de trabalho, assim é dado inicio a alocação dos recursos ociosos de forma parecida com a segunda parte do algoritmo no sub-tópico 3.3.1.1 *Construção de uma Solução GRASP para o PADE*.

Porém, foi adicionado um algoritmo que atua na escolha do depósito dos recursos que acabam de sair de atividades (novos ociosos) e que serão requeridos para outras atividades em períodos futuros, o qual analisa o depósito mais vazio no percurso entre os dois locais de trabalho.

Este novo algoritmo traz melhorias para a construção, pois, o movimento entre os dois locais de trabalho intercalado por um ou mais períodos no depósito, deverá ser realizado com o mínimo de consequências possíveis para a capacidade do depósito, assim, deixando um depósito mais livre para outros recursos futuros.

Para melhor compreender o caminho entre os espaços de trabalho abordado nesse algoritmo, a figura 4.4 mostra o deslocamento entre os locais ES_1 e ES_3 intercalado por períodos por um depósito que esteja com menor capacidade de recurso associado.

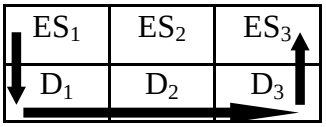


Figura 4.5 Caminho entre Espaços de Trabalho com Depósito intercalando.

4.5 Fase de Busca Local Proposta

Ao fim da construção da solução é aplicado o algoritmo VND com as estruturas de vizinhança anteriormente apresentadas, na seção 3.3.3. Porém, para uma melhoria na qualidade da solução final, foi criada uma nova estrutura de vizinhança pensada em ajustar a alocação dos fluxos.

A Vizinhança de Colunas tem essa nomenclatura por movimentar todas as atividades e os recursos ociosos de uma coluna do *layout* do PADE, assim, o de atividades e os recursos do depósito serão preservados, porém em uma nova coluna.

O exemplo desta nova estrutura pode ser visto na Tabela 4.1 onde todas as atividades e os recursos ociosos do local de trabalho ES_1 e D_1 são transferidos para o ES_3 e D_3 e simultaneamente o contrário.

Tabela 4.1 Exemplo de Vizinhança de Colunas

Período	Espaço de Trabalho		
	ES_1	ES_2	ES_3
1			**A_1 (6,7)**
2		A_2 (2,3)	**A_1 (6,7)**
3		A_2 (2,3)	
4		A_2 (2,3)	
5	**A_4 (1,8)**	A_2 (2,3)	**A_3 (4)**
6	**A_4 (1,8)**		**A_3 (4)**
7	**A_4 (1,8)**		
8			**A_5 (1,5)**
9	**A_6 (5,8)**		
10	**A_6 (5,8)**		

Período	Depósito		
	D_1	D_2	D_3
1	**1 8**	2 3	**4 5**
2	**1 8**		**4 5**
3	**1 8**	6	**4 5 7**
4	**1 8**	6	**4 5 7**
5		6	**5 7**
6		6 2 3	**5 7**
7		6 2 3	**4 5 7**
8	**1 8**	6 2 3	**4 7**
9	**1**	6 2 3	**4 7**
10	**1**	6 2 3	**4 7**

Para projetos com *layout* de 20 e 32 localidades são deslocados apenas um depósito e um local de trabalho como mostra a Tabela 5.2, onde é deslocado as atividade e recursos ociosos do local de trabalho ES_{10} e depósito D_{10} respectivamente para o ES_6 e D_6.

Tabela 4.2 Representação da Vizinhança de Colunas para o *Layout* 4x8

D_1	D_2	D_3	D_4	D_5	***D_6***	D_7	D_8
ES_1	ES_2	ES_3	ES_4	ES_5	***ES_6***	ES_7	ES_8
ES_9	***ES_{10}***	ES_{11}	ES_{12}	ES_{13}	ES_{14}	ES_{15}	ES_{16}
D_9	***D_{10}***	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}	D_{16}

Com a solução construída, é aplicada a Busca Tabu descrita e implementada por [1] e descrito no capítulo 3 Fundamentação Teórica deste trabalho.

Capitulo 5

Resultados Computacionais

O modelo em programação linear inteira descrito neste trabalho foi executado no software ILOG CPLEX disponibilizado pela IBM através da iniciativa acadêmica da mesma empresa. O código foi implementado na linguagem de programação C++, e os testes foram executados em um computador Intel Dual Core, com 4 GB de memória RAM e sistema operacional Linux Ubuntu. Os resultados do CPLEX aqui reportados, assim como o algoritmo HGT da literatura utilizado para comparação foram obtidos por [20], a partir de um computador Core 2 Quad, com 4 GB de memória RAM e sistema operacional Linux.

Para cada execução do algoritmo híbrido GFL foram realizadas 100 iterações do GRASP com o valor de alfa (parâmetro de seleção do tamanho da lista restrita de candidatos) igual a 10%.

Na Tabela 5.1, as colunas #Inst, HGT, GFL e CPLEX denotam respectivamente, o número da instância e os resultados das três estratégias para a resolução do PADE. A coluna GFL também é dividida em *Z*, o melhor valor obtido pelo algoritmo GFL em 10 execuções e Tempo, tempo médio de execução em segundos de GFL. A coluna *CPLEX* é dividida em *Z**, valor da solução ótima para o PADE alcançado pelo *solver* e finalmente o seu tempo de execução em segundos.

Para as instâncias com 6 locais, numeradas de 1 a 24 (conforme [20]), o GFL foi capaz de encontrar um novo melhor valor para a instância 21, com custo igual a 47, que corresponde a solução ótima encontrada pelo CPLEX.

Podemos perceber também que nessas instâncias com 6 locais, o CPLEX encontra as respectivas soluções ótimas em menos de 1 minuto, exceto para a instância 18. Não conseguimos chegar no CPLEX nas instâncias 18 e 21.

TABELA 5.1 COMPARATIVO ENTRE AS ESTRATÉGIAS HGT, GFL E CPLEX PARA INSTÂNCIAS COM 6 LOCAIS.

#Inst	HGT		GFL		CPLEX	
	Z	**Tempo**	**Z**	**Tempo**	**Z***	**Tempo**
1	16	0,2	16	0,2	16	0,3
2	25	0,2	25	0,2	25	0,3
3	18	0,3	18	0	18	0,3
4	25	0,2	25	0	25	3,5
5	16	0,2	16	0,1	16	1,3
6	27	0,2	27	0,1	27	5,5
7	16	0,3	16	0	16	3,5
8	31	0,2	31	0	31	0,9
9	25	0,4	25	0,2	25	6,8
10	46	0,5	46	0,3	46	19
11	32	0,5	32	0	32	7,7
12	41	0,4	41	0	41	15
13	28	0,5	28	0,3	28	11
14	45	0,5	45	0,2	45	18,9
15	35	0,7	35	0,1	35	17,8
16	49	0,5	49	0	49	4,5
17	35	0,8	35	0,5	35	16,2
18	62	0,7	62	0,4	60	62,3
19	46	0,8	46	0,1	46	26,6
20	60	0,6	60	0	60	57,8
21	48	0,9	47	1,1	46	46,5
22	67	0,8	67	0,6	67	103,2
23	55	0,9	55	0	55	24,6
24	74	0,7	74	0	74	32,6

Na tabela 5.2 podemos observar que nas instâncias 31 e 45 as soluções obtidas pelo GFL têm custo maior, enquanto na instância 39 apresenta menor custo. Não conseguimos chegar no CPLEX na instância 31. É possível perceber que o algoritmo proposto neste trabalho (GFL) alcança uma significativa redução no tempo de processamento das instâncias-teste sem comprometer a qualidade das soluções obtidas.

TABELA 5.2 COMPARATIVO ENTRE AS ESTRATÉGIAS HGT, GFL E CPLEX PARA INSTÂNCIAS COM 12 LOCAIS.

#Inst	**HGT**		**GFL**		**CPLEX**	
	Z	**Tempo**	**Z**	**Tempo**	**Z***	**Tempo**
25	31	2,1	31	1,6	31	59.551
26	43	2,8	43	1,6	43	20.663,1
27	43	2,4	43	0,2	43	1.008,4
28	55	1,7	55	0,1	55	582,3
29	29	2	29	1,2	29	94.398
30	49	2,3	49	1,3	49	27.609,2
31	**42**	3,4	**43**	0,3	42	1.950,3
32	69	2,5	69	0,1	69	3.716,4
33	52	9,1	52	4	*	*
34	72	7,2	72	2,8	*	*
35	73	7,4	73	0,7	73	790.671
36	95	4,9	95	0,4	95	57.012
37	48	6,2	48	3	*	*
38	83	6	83	3,2	*	*
39	**69**	6,1	**68**	1,2	68	177.400,5
40	108	3,9	108	0,4	108	211.216,3
41	78	10,6	78	7	*	*
42	102	11,3	102	5,8	*	*
43	110	10,9	110	1,1	*	*
44	140	7	140	0,8	*	*
45	**66**	13,1	**67**	6,2	*	*
46	116	10,8	116	7	*	*
47	115	13,2	115	1,5	*	*
48	171	8,3	171	0,7	*	*

Para as instâncias com 20 locais, numeradas de 49 a 72, e com 32 locais, numeradas de 73 a 96, conforme tabelas 5.3 e 5.4 tivemos comportamento similar entre os algoritmos HGT e GFL. Podemos destacar que para as maiores instâncias foram obtidas melhorias mais significativas no valor total do custo do projeto, além de enorme redução do esforço computacional. Podemos afirmar que a construção desenvolvida neste trabalho reduziu bastante o tempo computacional, pois não são necessárias muitas iterações nas buscas locais para alcançar as melhores soluções conhecidas.

Nas instâncias 68 e 82, foram encontrados recursos requeridos por duas atividades no mesmo período, o que obrigaria um recurso a estar em dois lugares ao mesmo tempo realizando atividades distintas. Desse modo, torna-se inviável uma solução real do PADE que não irá prever este tipo de ação, então, estas foram desconsideradas em todos os algoritmos testados neste trabalho.

TABELA 5.3 COMPARATIVO ENTRE AS ESTRATÉGIAS HGT, GFL E CPLEX PARA INSTÂNCIAS COM 20 LOCAIS.

#Inst	HGT		GFL		CPLEX	
	Z	Tempo	Z	Tempo	Z^*	Tempo
49	44	16	44	11,5	*	*
50	65	17,2	65	10.5	*	*
51	57	23,8	57	3,3	*	*
52	98	12,4	99	1,6	*	*
53	50	13,7	50	8,6	*	*
54	66	19,2	66	10,9	*	*
55	63	18,4	63	3,6	*	*
56	100	15,5	101	2,6	*	*
57	68	50,4	68	25,4	*	*
58	106	46,9	106	24,2	*	*
59	100	49,3	100	7,7	*	*
60	161	31,3	161	4	*	*
61	79	68,4	79	29,4	*	*
62	125	59,8	124	34,5	*	*
63	124	73,4	125	12,3	*	*
64	193	62,2	192	7,5	*	*
65	103	110,1	103	54,9	*	*
66	159	89,9	159	45,6	*	*
67	159	113,7	159	15,5	*	*
68	---------	---------	---------	---------	*	*
69	120	172,7	120	64,2	*	*
70	176	134,1	176	84,6	*	*
71	173	173,3	173	32,9	*	*
72	271	79,9	271	52,3	*	*

TABELA 5.4 COMPARATIVO ENTRE AS ESTRATÉGIAS HGT, GFL E CPLEX PARA INSTÂNCIAS COM 32 LOCAIS.

#Inst	HGT		GFL		CPLEX	
	Z	Tempo	Z	Tempo	Z^*	Tempo
73	72	186,1	72	55,9	*	*
74	97	213,7	97	55,9	*	*
75	109	213,5	109	53,2	*	*
76	156	195,3	156	27,7	*	*
77	72	195,7	72	42,3	*	*
78	100	116,9	100	63,2	*	*
79	109	242,2	109	24,7	*	*
80	177	209,1	177	14,2	*	*
81	117	399,4	117	175,1	*	*
82	---------	---------	---------	---------	*	*
83	190	648,9	190	154,5	*	*
84	280	468,5	280	56,6	*	*
85	126	669,8	127	200,3	*	*
86	193	381,3	193	227,4	*	*
87	196	679,9	196	104	*	*
88	299	379,5	299	36,6	*	*
89	170	996,1	168	362	*	*
90	254	834,5	254	335,9	*	*
91	278	1.264,50	278	310,6	*	*
92	393	852,8	393	70,8	*	*
93	188	1.532,70	185	437,6	*	*
94	284	1.123,60	284	466,6	*	*
95	326	1.164,30	321	218.,4	*	*
96	473	724,2	470	82,2	*	*

TABELA 5.5 INSTÂNCIAS EM QUE O HGT E GFL OBTIVERAM SOLUÇÕES DISTINTAS

#Inst	HGT		GFL	
	Z	**Tempo**	**Z**	**Tempo**
21	48	0,9	**47**	1,1
52	**98**	12,4	99	1,6
56	**100**	15,5	101	2,6
62	125	59,8	**124**	34,5
63	**124**	73,4	125	12,3
64	193	62,2	**192**	7,5
85	**126**	669,8	127	200,3
89	170	996,1	**168**	362
93	188	1.532,7	**185**	437,6
95	326	1.164,3	**321**	218,4
96	473	724,2	**470**	82,2

A Tabela 5.5 apresenta as instâncias cujas soluções tiveram valores diferentes alcançados pelas estratégias HGT e GFL, exceto pelas 3 instâncias já apresentadas na Tabela 5.2. Das 11 instâncias apresentadas o GFL obtém melhores valores em 7 instâncias, perdendo nas outras 4. Podemos destacar que para as maiores instâncias foram obtidas melhorias mais significativas no valor do custo do projeto pelo GFL, além de enorme redução do esforço computacional. Para as instâncias 93, 95 e 96, foram obtidas, respectivamente, reduções de 3, 5 e 3 unidades no valor do custo final do projeto. Em relação aos tempos computacionais (em segundos) estes variaram de 1.532,70; 1.164,30 e 724,2 em HGT para 437,6; 218,4 e 82,2 em GFL, nas supracitadas instâncias, respectivamente. Com base nesses resultados, podemos afirmar que a construção desenvolvida neste trabalho reduziu bastante o tempo computacional, pois não são necessárias muitas iterações nas buscas locais para alcançar as melhores soluções conhecidas.

A partir destes resultados é mostrado que a construção desenvolvida neste trabalho otimizou o tempo, pois não são necessárias muitas interações nas buscas locais para alcançar as melhores soluções conhecidas.

Uma melhora de aproximadamente 75% do tempo do algoritmo de [1], como exemplo a instância 96 que é considerada a mais complexa, foi melhorado o custo por 3 movimentos na solução final e o tempo em aproximadamente 89%.

Capitulo 6

Conclusões

Neste trabalho foi desenvolvido uma meta-heurística híbrida baseada na conjunção da meta-heurística GRASP com uma nova formulação matemática usada na fase de construção para a resolução do Problema da Alocação Dinâmica de Espaços, recém-criado na literatura por McKendall Jr.

A maior contribuição deste trabalho reside na construção de uma nova heurística que traz a utilização de um grafo que representa as dependências de recursos entre as atividades. Isso possibilitou a geração de uma nova formulação matemática baseada em um modelo de fluxo em redes. Este modelo permitiu que a fase de construção da meta-heurística GRASP gerasse boas soluções iniciais para a fase seguinte de busca local. Este algoritmo híbrido provou ser eficaz, obtendo novos melhores valores para instâncias não resolvidas pelo CPLEX dentro de limite aceitável de tempo e mostrou na média, uma significativa redução no tempo de processamento das instâncias sem comprometer a qualidade das soluções quando comparado com melhores valores conhecidos na literatura.

Acredita-se que uma melhoria para o problema, seria uma modelagem dos movimentos dos recursos entre depósitos e locais de trabalho para um grafo, onde, cada vértice seria um estado (ocioso ou em execução) e as arestas seriam a transição entre eles, com valor equivalente ao custo da movimentação dos dois locais. Por isso é interessante minimizar, os fluxos de cada recurso e também uma possível união entre os dois mapeamentos do problema.

Como proposta para trabalhos futuros consideramos um caminho promissor a abordagem sobre a complexidade dos diversos algoritmos apresentados.

Referências Bibliográficas

[1] Silva, G. C., *Algoritmos heurísticos e híbridos para o Problema da Alocação Dinâmica de Espaços*, Tese de Doutorado, Rio de Janeiro: UFRJ/COPPE, 2010.

[2] McKendall JR., A. R., Noble, J. S., Klein, C. M. *Simulated Annealing heuristics for managing resources during planned outages at electric power plants*, Computers & Operations Research, v. 32, pp. 107-125, 2005.

[3] McKendall JR., A. R., Jaramillo, J. *A tabu search heuristic for the dynamic space allocation problem*, Computers & Operations Research, v. 33, pp. 768-789, 2006.

[4] Chan, H. M., Milner, D. A. *Direct clustering algorithm for group formation in cellular manufacture*. Journal of Manufacturing Systems, v. 1, pp. 165–175, 1982.

[5] Tompkins, J. A., White, J. A., Bozer, Y. A., Tanchoco, J. M. A. *Facilities Planning*. John Wiley, New Jersey, 1996.

[6] Blum, C., Roli, A. *Metaheuristics in combinatorial optimization: Overview and conceptual comparison*. ACM Comput. Surv., v. 35, n. 3, pp. 268–308, 2003.

[7] Glover, F. *Tabu search: A tutorial*. Interfaces, v. 20, pp. 74–94, 1990.

[8] Shorin-Kapov, J. *Tabu search applied to the quadratic assignment problem formulation*. ORSA Journal on Computing (1990), pp. 443–455, 1990.

[9] Kaku BK, Mazzola JB. *A tabu-search heuristic for the dynamic plant layout problem*. INFORMS Journal on Computing 1997;9(4):374–84.

[10] Rosenblatt M. J. *The dynamics of plant layout*. Management Science, v. 32, pp. 76-86, 1986.

[11] Novaes, A. C. *Logística e gerenciamento da cadeia de distribuição: estratégia, operação e avaliação*, Rio de Janeiro: Ed. Campus, 2001.

[12] Ballou, R. H. *Logística empresarial: transportes, administração de materiais e distribuição física*. São Paulo: Ed. Atlas, São Paulo, 1993.

[13] Neto, F. F. *A Logística como estratégia para a obtenção de vantagem competitiva*. Revista FAE Business, n. 1, pp. 1-4, dez. 2001.

[14] Loiola, E. M.; Abreu N. M.; Netto P. O. *Uma revisão comentada das abordagens do problema quadrático de alocação*. Scielo Brasil, Rio de Janeiro 2004.

[15] Lee, C.-G., Ma, Z., *The generalized quadratic assignment problem*, Working paper, Dept. of Mechanical and Industrial Engineering, University of Toronto, Canada, 2005.

[16] Wilhelm MR, Ward TL. *Solving quadratic assignment problems by simulated annealing*. IIE Transactions 1987; 19: 107-19.

[17] Heragu SS, Alfa AS. *Experimental analysis of Simulated Annealing based algorithms for the layout problem*. European Journal of Operational Research 1992; 57: 190-202.

[18] McKendall JR., A. R. *Improved tabu search heuristic for the dynamic space allocation problem*, Computers & Operations Research, v. 35, pp. 3347-3359, 2008.

[19] Mladenovic, N., Hansen, P. *Variable neighborhood search*, Comput. Oper. Res., v. 24, n. 11, pp. 1097-1100, 1997.

[20] Silva G. C., Bahiense L., Ochi L. S., Netto P. O. B. *The dynamic space allocation problem: Applying hybrid GRASP and Tabu search metaheuristics*, Computers & Operations Research, v. 39, pp. 671-677, 2012.

[21] Feo, T., Resende, M. *Greedy randomized adaptive search procedures*, J. of Global Optimization, v. 6, pp. 109-133, 1995.