Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada
Mestrado em Sistemas e Computação

O Problema Bin Packing Tridimensional em Contêineres: Usando Interação com o Usuário

Carlos Heitor Pereira Liberalino

Professor orientador
Dario José Aloise

Natal, RN
Agosto de 2005

“E eu que já não sou assim
muito de ganhar,
junto as mãos ao meu redor,
faço o melhor que sou capaz
só pra viver em paz.”
(Marcelo Camelo)

Dedico este trabalho, com muito amor, aos meus pais, a minha irmã,
e ao meu filho, pela motivação, carinho e dedicação.

Agradecimentos

Agradeço a Deus, meu Pai Eterno, por ter me iluminado e me guiado, acendendo todas as luzes e colocando todas as placas para que eu pudesse seguir o meu caminho.

A Nossa Senhora, por ter sempre me guardado e me protegido nessas caminhadas, sempre intercedendo por mim e me ajudando nas horas mais difíceis.

Aos meus pais, Dedé e Fátima, por terem me dado todo apoio e força para enfrentar a vida, e por terem me ensinado a ter paciência, humildade e, principalmente, fé em Deus.

A minha irmã, Camila, por ser sempre o meu exemplo de garra, força e determinação, além de ser minha companheira fiel, amiga e alma-gêmea, compartilhando sempre as minhas venturas e desventuras.

Ao meu filho, Lucca, por toda a alegria e paz que ele me trás com o seu singelo sorriso e sua forma pura de encarar o mundo, me mostrando que todo esse mar de tecnologia e complicação existente não passam de uma simples palavra em seu limitado vocabulário: “Dadô!” (Computador).

A minha querida Andreza MenCa, pela paciência e pelos momentos de paz e música que acrescentou em minha vida.

Ao meu orientador, Professor D.Sc. Dario José Aloise, por sempre ter me acompanhado e aconselhado, com a seriedade de um mestre na hora de falar, e a paciência de um pai na hora de ouvir.

Aos Professores, Doutores, Hugo Alexandre Dantas do Nascimento e Luiz Marcos Garcia Gonçalves, por me apresentar o *framework User Hints* e pela excelente dica de utilizar a ferramenta gráfica de design do *Qt*, respectivamente.

Ao meu amigo, Josivan da Silva Xavier, por toda a ajuda e companheirismo, compartilhando idéias e ajudando a aprumar esse horizonte distante em comum.

As minhas amigas e companheiras de Otimização, Jéssica Neiva e Marilyn Christine, por terem sido as minhas aliadas em busca da solução ótima para esta etapa em comum da vida, com e sem auxílio de heurísticas.

As minhas amigas, Cynthia Sâmara e Valnaide Bittencourt, por todo auxílio e amizade, e por teres sido as "mães" deste projeto.

A Seu Gaspar, Demerson, Vamberto, Graça e Celma, por manterem vivo o espírito de cordialidade e amizade neste departamento, e por toda a ajuda que me prestaram até mesmo com um simples sorriso e o desejo de um bom dia.

Índice

Índice de Figuras e Tabelas

Resumo

Problemas de Corte e Empacotamento (*Cut & Packing*) tentam achar uma disposição ótima de unidades menores (itens ou *bins*) dentro de unidades maiores (objetos), satisfazendo um conjunto de restrições e otimizando uma determinada função. A abordagem proposta neste trabalho tem como objetivo apresentar uma heurística híbrida para o *Problema Bin Packing 3D* em Contêineres, que faz uso das bibliotecas gráficas *Open-GL* e *Qt*, e de uma interação homem-máquina chamada *User Hints* (Dicas de Usuário), onde o usuário, com sua percepção e raciocínio lógico, pode manualmente alterar o resultado fornecido pelo computador, em qualquer instante, para obter uma melhoria para a solução. Os resultados obtidos em 14 instâncias constataram uma melhoria de 2,34% comparada à heurística utilizada, conhecida como "Heurística de Suavização de Superfícies Irregulares" (HSSI).

Palavras-chave: Corte e Empacotamento, Otimização Combinatória, User Hints, OpenGL, Qt.

Abstract

Cut and Packing Problems try to find an optimum disposition of small units (itens or bins) inside bigger units (objects), satisfying a set of constraints and optimizing a certain function. The approach proposed in this work shows an hybrid heuristic to the *3D Bin Packing Problem* in containers that make use of the *OpenGL* and *Qt* graphics libraries, and a human-machine interaction named *User Hints*, where the user, with your own perception and logic reasoning, can manually change the result given by the computer, at any moment, to obtain an improvement to the solution. The results obtained show an improvement of 2.34% compared with the used heuristic, known as *Heurística de Suavização de Superfícies Irregulares (Irregular Surfaces Extenuation's Heuristic)*, in 14 instances of the problem.

Keywords: Cut and Packing, Combinatorial Optmization, User Hints, OpenGL, Qt.

1 – Introdução

Métodos exatos para a solução de problemas de Otimização combinatória exigem um tempo computacional considerado inviável na maioria dos casos. Por outro lado, grande parte das aplicações reais necessita apenas de um bom resultado aproximado em vez de resultados ótimos. Por causa disso, o uso de métodos heurísticos para a resolução destes problemas vem se tornando cada vez mais freqüente.

O Problema do Empacotamento Tri-dimensional (*Bin Packing 3D*) é uma subclassificação do clássico problema de Otimização combinatória conhecido como "Problema de Corte e Empacotamento". Ele pertence à classe dos problemas NP-Árduos [24], onde soluções não são encontradas em tempo polinomial, fazendo-se, portanto, necessária à utilização de técnicas heurísticas para tornar possível a sua resolução, aproximada, mas aceitável e em um tempo computacionalmente viável.

Problemas de empacotamento têm sido tratados por muitos pesquisadores, mas as maiorias dos trabalhos encontrados na literatura têm se restringido a casos uni ou bi-dimensionais. É amplamente aceito que, devido a sua complexidade, qualquer solução analítica para este problema é improvável num futuro previsível. Por isso, a maioria das abordagens bem sucedidas do problema tem a forma de heurísticas. Contudo, apesar do recente surgimento de boas heurísticas para as soluções tri-dimensionais, ainda não se conseguiu um empacotamento que possa ser considerado excelente, provavelmente, em parte, ao nível da complexidade computacional envolvida.

O algoritmo apresentado neste trabalho mostra uma estratégia heurística baseada na suavização de superfícies irregulares, o qual encontra boas soluções em um tempo computacional aceitável para o problema *Bin Packing 3D* (BP3D). Comparado com algoritmos existentes, esta heurística, denominado de "Heurística de Suavização de Superfícies Irregulares" (HSSI), tem um desempenho competitivo [3].

Objetivo

O objetivo deste trabalho consiste em aplicar técnicas de uma abordagem colaborativa Homem-Computador chamada *User Hints* [44] (Dicas de Usuário) para obter uma melhoria na solução para o BP3D. Usando a HSSI, juntamente com *User*

Hints, foi criado um sistema interativo chamado *BP3D Interativo*, no qual o usuário será capaz de observar os resultados gerados pela heurística e interferir nestes de acordo com os seus critérios e experiência. A grande vantagem desta abordagem é que o homem possui um senso crítico (o qual computador nenhum possui) que pode ser usado, fazendo com que o processo atenda as necessidades com mais fidelidade e realidade.

O restante deste trabalho está organizado como descrito a seguir. O Capítulo 2 apresenta uma revisão bibliográfica contendo conceitos relativos à Otimização combinatória, com algumas técnicas e aplicações; uma abordagem sobre problemas de empacotamento; conceito de Otimização interativa e alguns tipos, dando enfoque ao *framework User Hints*; e concluindo com alguns trabalhos similares. O Capítulo 3 apresenta a abordagem interativa usada no empacotamento 3D, seu modelo, visualização e tipos de dicas; mostra como o *BP3D Interativo* foi implementado, descrevendo as bibliotecas gráficas usadas e alguns métodos implementados; e, ao final, descreve os testes realizados e os resultados obtidos. O Capítulo 4 apresenta a conclusão e trabalhos futuros.

2 – Revisão Bibliográfica

2.1 – Otimização Combinatória

Problemas de Otimização Combinatória ocorrem em áreas tão diversas como projetos de sistemas de distribuição de energia elétrica, posicionamento de satélites, projetos de computadores e de chips VLSI, roteamento e/ou escalonamento de veículos, alocação de trabalhadores ou máquinas a tarefas, empacotamento de caixas em contêineres, corte de barras e placas, seqüenciamento de genes e DNA, classificação de plantas e animais, *etc*. Muitos desses problemas podem ser modelados como problemas de maximizar (ou minimizar) uma função cujas variáveis devem obedecer a certas restrições. Encontrar soluções ótimas, ou mesmo aproximadas, para esses tipos de problemas é um desafio nem sempre fácil de ser vencido. Para alguns desses problemas são conhecidos métodos eficientes (rápidos) para se resolvê-los; para outros, métodos de enumeração implícita, relaxação, métodos de planos-de-corte (nem sempre tão rápidos) são alguns dos aplicados com maior sucesso na solução de problemas reais. O problema de Otimização Combinatória, em geral, se resume a encontrar, dentre todas as possíveis soluções, aquela cujo custo seja o menor possível [52].

Uma forma de resolver tais problemas seria simplesmente enumerar todas as soluções possíveis (viáveis) e guardar aquela de menor custo. Entretanto, essa é uma idéia que deve ser desconsiderada, pois para qualquer problema de um tamanho considerável, esse método torna-se impraticável, já que o número de soluções possíveis é muito grande. Mesmo que seja utilizado um supercomputador para resolver o problema, o tempo de processamento pode ser intratável, do ponto de vista da aplicação prática. Embora não pareça, existem milhares de problemas práticos dessa natureza. Os problemas aqui tratados são conhecidos tecnicamente por NP-Árduo (conceito comentado mais adiante). Portanto, técnicas computacionais mais apuradas são necessárias para resolver esses problemas.

2.1.1 – Problemas NP-Completo e NP-Árduo

A classe P é definida como aquela composta por problemas de decisão que podem ser resolvidos deterministicamente por algoritmos de complexidade polinomial.

Um algoritmo de complexidade não polinomial é dito exponencial. A classe NP é um conjunto dos problemas que podem ser resolvidos não deterministicamente por algoritmo de complexidade polinomial. E a classe NP-Completo, é conjunto de problemas NP com a propriedade adicional de que a existência de algoritmo polinomial e determinístico que o resolva implica em P ser igual a NP. Já um problema de decisão π é dito NP-Árduo se todo problema NP é redutível polinomialmente a π.

A maioria dos problemas de otimização NP-árduos exige a avaliação de um imenso número de alternativas para determinar a sua solução exata. Por isso, é de grande importância a existência de heurísticas, que forneçam soluções de boa qualidade (próximas da solução ótima) em um tempo computacional razoável [43].

A solução de problemas de elevado nível de complexidade computacional (NP-Completo, NP-Árduo) tem sido um desafio constante para os pesquisadores de diversas áreas. Particularmente na Otimização, Pesquisa Operacional, Ciências da Computação, Matemática e Engenharias, nos defrontamos freqüentemente com problemas altamente combinatórios, cuja solução ótima em muitos casos ainda está limitada somente a pequenas instâncias.

Resolver um problema de otimização combinatória de forma exata consiste em obter um resultado que corresponde à solução viável de maior (ou menor) valor dentre todas as possíveis encontradas na tentativa de maximizar (ou minimizar) uma determinada função objetivo.

2.1.2 – Técnicas Utilizadas (Modelos e Algoritmos)

A resolução de um problema de Otimização normalmente precisa passar por duas fases: transformação do problema em um modelo e, posteriormente, a implementação de um algoritmo para resolver tal modelo. A seguir é mostrada uma relação de técnicas comumente utilizadas nesta resolução:

Algoritmos Aproximados

Nos Métodos Heurísticos não há garantia alguma a respeito da solução encontrada. Isto é, não há como saber se a solução obtida está "perto" ou "longe" da melhor solução possível em termos de qualidade. Contudo, há ocasiões em que essa noção de proximidade faz-se necessária. Por exemplo, pode-se estar interessado em uma

solução que não precisa ser a melhor, mas deve ser, no máximo, 10% pior que a melhor solução possível. Nesses casos, entram em ação os Algoritmos Aproximados.

Teoria dos Grafos

Modelos baseados em grafos são imensamente utilizados em muitos problemas de otimização combinatória. Grafo é uma forma de representar graficamente, através de pontos e linhas, um conjunto de elementos e suas relações [13]. Esse recurso é muito utilizado para modelar problemas reais, por ser uma forma bastante intuitiva de representá-los. Além disso, na literatura podem ser encontrados algoritmos para resolver diversos problemas em grafos.

Programação por Restrições

Esta técnica teve suas origens no campo da Inteligência Artificial, mais especificamente no ramo da Programação Lógica. Simplificando, consiste em um mecanismo de inferência lógica auxiliado por um resolvedor de restrições que são impostas sobre as variáveis do problema em questão. A modelagem dos problemas é facilitada por se tratar de uma linguagem declarativa baseada em implicações lógicas. Restrições complexas podem ser escritas de forma clara e concisa. Essa técnica tem sido aplicada com bastante sucesso em problemas de porte industrial.

Programação Linear e Programação Inteira

A Programação Linear utiliza modelos matemáticos para expressar um problema em termos de variáveis contínuas e um conjunto de restrições lineares sobre essas variáveis. Dada uma função objetivo que descreve basicamente como é calculado o "custo" a ser minimizado, aplica-se um algoritmo que resolve o problema de forma eficiente. Entretanto, na vida real é muito comum que as variáveis precisem assumir valores inteiros e não contínuos. No entanto, quando se impõe a restrição de que as variáveis assumam valores inteiros o problema pode ficar muito mais difícil, e a idéia natural de simplesmente arredondar os valores não traz bons resultados, sendo aí que entra a Programação Inteira [63].

Algoritmos Paralelos

Nesta área também são investigadas ferramentas para programação paralela (linguagens de programação paralelas e bibliotecas de troca de mensagens) e

desenvolvimento de algoritmos paralelos exatos ou heurísticos para a solução de problemas combinatórios. Alguns dos tópicos explorados são: a utilização e a avaliação de ferramentas de programação paralela, tais como PVM, Linda e MPI; balanceamento de carga; tolerância a falhas; concepção, implementação e avaliação de estratégias paralelas síncronas (decomposição de domínio) e assíncronas (cooperação entre múltiplas buscas) para métodos de busca heurística.

Redes Neurais

São também imensamente utilizadas em muitos problemas de Otimização combinatória. As redes neurais artificiais consistem em um método de solucionar problemas de inteligência artificial, construindo um sistema que tenha circuitos que simulem o cérebro humano, inclusive seu comportamento, ou seja, aprendendo, errando e fazendo descobertas. Mais que isso, são técnicas computacionais que apresentam um modelo inspirado na estrutura neural de organismos inteligentes e que adquirem conhecimento através da experiência [64].

A maioria dos modelos de redes neurais, que podem ter centenas ou milhares de unidades de processamento, possuem alguma regra de treinamento, onde os pesos ou entradas de suas conexões são ajustados de acordo com os padrões apresentados. Arquiteturas neurais são organizadas em camadas, com unidades que são conectadas às unidades da camada posterior.

A rede neural passa por um processo de treinamento a partir dos casos reais conhecidos, adquirindo, a partir daí, a sistemática necessária para executar adequadamente o processo desejado dos dados fornecidos. Sendo assim, a rede neural é capaz de extrair regras básicas a partir de dados reais, diferindo da computação programada, onde se é necessário um conjunto de regras rígidas pré-fixadas e algoritmos.

Algoritmos Heurísticos e Metaheurísticas

Os Métodos Heurísticos são algoritmos que não garantem encontrar a solução ótima de um problema, mas são capazes de retornar uma solução de qualidade em um tempo adequado para as necessidades da aplicação. Foi esta a técnica utilizada para solucionar o problema de empacotamento tri-dimensional (*Bin Packing 3D* – BP3D), alvo deste trabalho, como relatado no Capítulo 3.

Como já mencionado, os métodos tradicionais de Otimização exata se caracterizam pela rigidez de seus modelos matemáticos representados através de seus teoremas, dificultando a representação de situações reais cada vez mais complexas e dinâmicas. O problema desta falta de flexibilidade foi um pouco reduzido a partir do momento em que se passou a associar técnicas de Otimização com ferramentas de Inteligência Artificial, mais especificamente, com as ferramentas de busca heurística. De fato, os algoritmos heurísticos, ou simplesmente heurísticas, se caracterizam pela sua flexibilidade e têm como objetivo encontrar soluções de boa qualidade num tempo computacional viável [2].

Observa-se um conflito entre a qualidade da solução gerada e o esforço computacional despendido nos muitos métodos de resolução de problemas de Otimização. Entretanto, a possibilidade de minimizar esse conflito através do uso das heurísticas causou um grande aumento no interesse por estudos nesse campo.

A utilização de técnicas heurísticas vem sendo fundamentada no desenvolvimento de conceitos de Complexidade Computacional e na classificação de problemas difíceis, para os quais parece não existir algoritmo que os resolva em tempo polinomial. Apesar de existirem algumas críticas aos métodos heurísticos (metaheurísticas), as boas soluções e, em alguns casos soluções ótimas, encontradas por eles em intervalos de tempo reduzidos e compatíveis com as exigências de situações reais, têm contribuído para o grande interesse pelo assunto.

Realizando-se uma comparação entre os métodos exatos de solução e as metaheurísticas, estas costumam ser procedimentos mais simples e flexíveis, o que lhes permite abordar modelos mais complexos [48]. Elas dependem de certas características para garantirem o seu sucesso, tais como:

- Pré-processamento;
- Boas técnicas para construir soluções iniciais;
- Reinicializar procedimentos;
- Adaptação a instâncias especiais;
- Escapar de ótimos locais;
- Fazer uso da estrutura do problema;
- Randomização controlada;
- Diversificação de busca quando nenhuma melhoria adicional parece possível;

- Intensificação da busca em regiões promissoras;
- Melhoria de solução através de busca local;

Entende-se por busca local uma busca realizada em um determinado espaço de solução com o objetivo de melhorar uma solução encontrada. Desta forma, algoritmos de busca local são construídos como uma forma de exploração do espaço de busca, possuindo normalmente uma heurística subordinada, utilizada para obter uma solução aprimorada na vizinhança. Diferentes aspectos do espaço de busca influenciam o desempenho de algoritmos de busca local. Estes algoritmos têm como principais aspectos a serem considerados:

- Partida: solução inicial obtida através de um método construtivo;
- Iteração: melhoria sucessiva da solução corrente através de uma busca na sua vizinhança;
- Parada: primeiro ótimo local encontrado (não existe solução vizinha aprimorada).

Processos de diversificação e intensificação no espaço de soluções viáveis também são fatores importantes a serem considerados na elaboração de uma nova heurística. No processo de diversificação, o objetivo é direcionar a busca para novas regiões, de forma a atingir todo o espaço de soluções possíveis. Enquanto que no processo de intensificação há um reforço à busca na região (vizinhança) de uma solução historicamente considerada boa.

As heurísticas isoladamente também possuem suas limitações, e a principal delas é a deficiência histórica de, em muitos casos, não conseguirem superar as armadilhas dos ótimos locais em Problemas de Otimização. Além disso, a falta de uma base teórica dos métodos heurísticos produz algoritmos muito especializados, ou seja, apesar de sua flexibilidade em incorporar novas situações, o seu desempenho pode oscilar muito com modificações, mesmo pequenas, no problema analisado.

Portanto, modelos rígidos da Otimização, reunidos com os métodos flexíveis da busca heurística, proporcionou o surgimento dos chamados Métodos Inteligentemente Flexíveis, ou Metaheurísticas, que procuram o desenvolvimento de técnicas dotadas de uma certa rigidez matemática e com facilidades em incorporar novas situações, sem, no

entanto, emergir numa inflexibilidade às vezes caótica encontrada em métodos heurísticos.

Os anos 80 marcam o surgimento de artigos sobre novos métodos heurísticos com ferramentas adicionais para tentar superar as limitações das heurísticas convencionais. Surgiram, então, as metaheurísticas como paradigmas de desenvolvimento de algoritmos heurísticos. Dentre as várias técnicas produzidas para tentar reduzir o risco de paradas prematuras, destacamos: Simulated Annealing [33], Busca Tabu [28], GRASP [7] e a Otimização Evolutiva incluindo: os Algoritmos Genéticos (AGs), inicialmente propostos por Holland [30], o Scatter Search (SS), proposto por Fred Glover (1977), a Programação Genética (PG), proposta por Koza [36], e a Programação Evolutiva (PE) proposta por Fogel [23].

Embora com filosofias distintas, estas metaheurísticas possuem, em comum, características que as distinguem das heurísticas convencionais, como, por exemplo, incluir ferramentas para tentar escapar das armadilhas dos ótimos locais e a facilidade para trabalhar em ambientes paralelos.

2.1.3 – Aplicações Práticas

Existe um número enorme de problemas da vida real que podem ser encarados como problemas de Otimização Combinatória. Para se ter uma idéia da diversidade de aplicações possíveis, uma lista de alguns problemas é mostrada a seguir:

- Roteirização de Veículos
- Projeto GENOMA
- Escalonamento de Horários em Escolas e Universidades (Timetabling)
- Corte de Materiais
- Escalonamento de Trabalho Humano
- Escalonamento de Tarefas em Máquinas
- Planejamento de Produção
- Localização de Facilidades
- Projeto de Circuitos Integrados
- Projeto de Redes com Restrições de Conectividade
- Empacotamento

Os Problemas de Empacotamento têm como idéia principal arrumar a melhor maneira de agrupar um conjunto de itens de modo que o espaço total necessário para guardá-los seja minimizado. Em certos casos, o espaço disponível para o armazenamento é predeterminado e o objetivo é guardar o maior número de itens possível. Por exemplo, uma companhia de mudanças deseja encontrar a melhor maneira de arrumar os móveis dentro dos seus caminhões, de modo a realizar a mudança com um número mínimo de viagens.

Como esta é a aplicação prática objeto deste trabalho, um maior aprofundamento sobre este tipo de problema será realizado mais adiante, na seção à seguir.

2.2 – Problemas de Empacotamento

2.2.1 – Introdução

Problemas de corte, de modo geral, consistem em cortar unidades maiores (objetos) em unidades menores (itens), otimizando uma determinada função (por exemplo, minimização da perda). Estes aparecem em ambientes de produção de algumas indústrias, tais como de papel, móveis, vidro, metalúrgica, plástica, têxtil, etc. Nestas indústrias, deparam-se freqüentemente com o problema de se cortar peças grandes (objetos) em pedaços menores (itens) requeridos internamente por outros setores da empresa ou por clientes externos. É necessário planejar os cortes para minimizar os efeitos negativos, como o desperdício que onera os custos de produção [32].

Análogo aos problemas de corte, problemas de empacotamento consistem em empacotar unidades menores (itens) em unidades maiores (objetos), otimizando uma determinada função (por exemplo, minimização do espaço ocioso) e satisfazendo um conjunto de restrições adicionais (por exemplo, estabilidade de carregamento) [68]. Estes são igualmente relevantes no planejamento de operações logísticas como armazenagem, movimentação e transporte, visando à minimização de espaços ociosos, etc. [51].

Estas duas classes de problemas de Otimização estão intimamente relacionadas e são tratadas na literatura como “Problemas de Corte e Empacotamento” [18]. Por

questão de simplicidade, e devido à observação de exemplos práticos, ao nos referirmos aos problemas de corte, estaremos abordando os de natureza uni e bi-dimensional, e, ao nos referirmos aos problemas de empacotamento, os de natureza tri-dimensional.

Para cada versão dimensional dos problemas, existem diversas variantes de acordo com o que se procura otimizar. As versões do caso uni-dimensional, apesar de serem aparentemente mais simples, são computacionalmente difíceis e têm sido investigadas desde o início da década de 60 [26]. Vários algoritmos de aproximação têm sido desenvolvidos para este caso e também para o caso bi-dimensional. Já o estudo de problemas de empacotamento tri-dimensional é mais recente, tendo sua pesquisa sido intensificada a partir dos anos 90 [62].

Para auxiliar no esclarecimento do assunto, Dyckhoff [18] apresentou uma taxonomia ou diagrama relacional (Figura 2.1) de problemas de corte e empacotamento (C&E), que são bastante usados na literatura sob diferentes nomes, embora possuam a mesma estrutura lógica (ver Tabela 2.1).

Classificação dos Problemas de C&E pela Taxonomia de Dyckhoff

Para mostrar a forte relação existente entre os problemas de C&E, Dyckhoff [18] classificou-os segundo as seguintes quatro principais características, com seus respectivos tipos mais comumente citados, denotados com os símbolos indicados:

1. Dimensão do problema

- (1) Uni-dimensional;
- (2) Bi-dimensional;
- (3) Tri-dimensional;
- (N) N-dimensional, com N>3.

2. Forma de alocação das unidades

- (T) Todos os objetos grandes (bins) e uma *seleção* de pedaços menores (itens);
- (S) Uma *seleção* de objetos maiores (bins) e todos os pedaços menores.

3. Sortimento de objetos grandes

- (U) Um objeto;

- (I) Unidades de formatos idênticos;
- (D) Unidades de formatos diferentes.

4. Sortimento de pequenos itens

- (P) Poucas unidades de formato diferentes;
- (M) Muitas unidades de muitos formatos diferentes;
- (R) Muitas unidades de poucos formatos diferentes (não-congruentes);
- (C) Formatos congruentes.

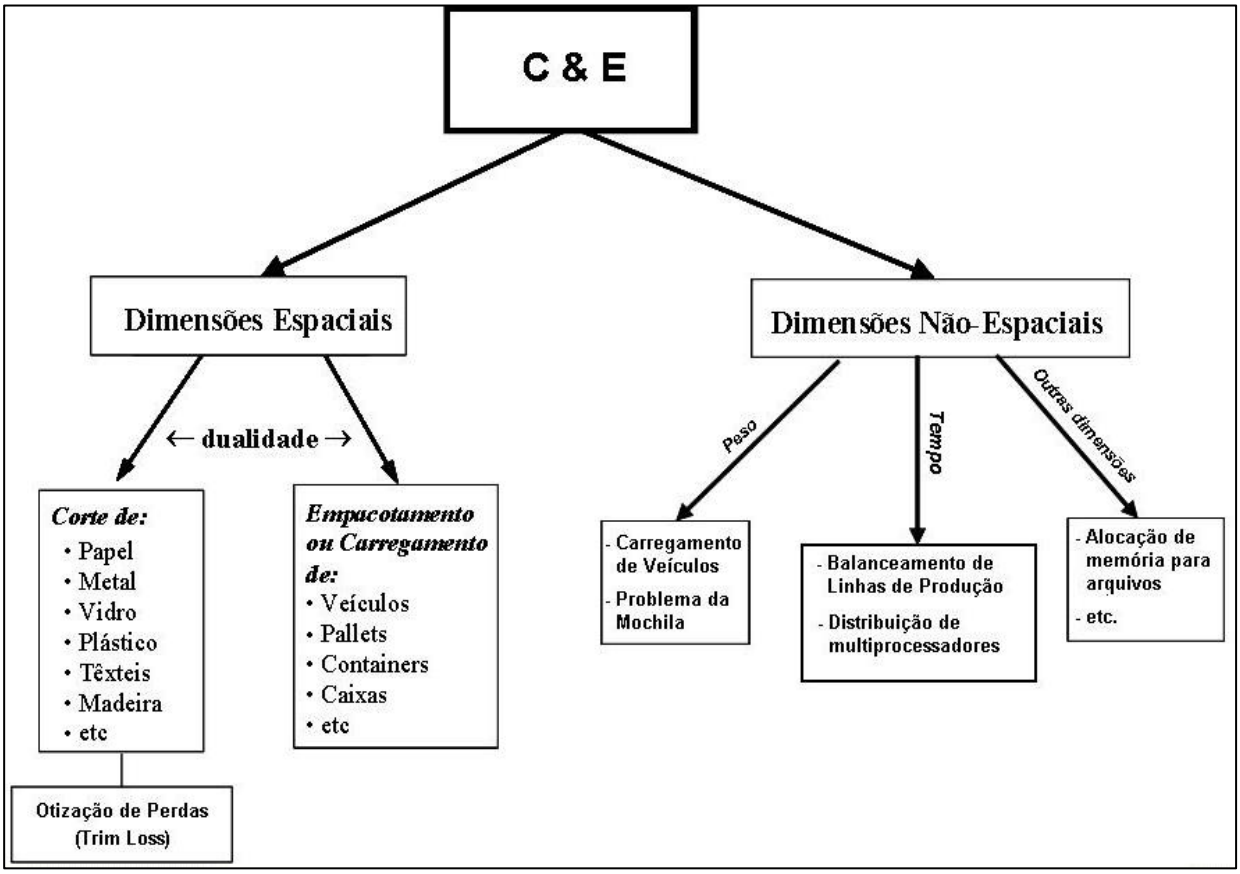

Figura 2.1 – Taxonomia dimensional dos problemas de C&E

Neste caso, os "formatos congruentes" são figuras geométricas de mesma forma e tamanho, diferindo, no máximo, com respeito à orientação e posição. Isto acontece somente nos casos multidimensionais.

Combinando estes tipos de características obtêm-se 96 diferentes tipos de problemas de C&E, os quais Dyckhoff [18], abreviadamente, denotou por α / β / γ / δ.

A tabela abaixo mostra alguns destes problemas e sua classificação. A ausência de símbolo, assinalada com "*", significa que qualquer uma das possibilidades é possível.

Nome	Classificação
Problema da Mochila Clássico	1 / T / U / *
Problema de Empacotamento de Bins Clássico	1 / S / I / M
Problema de Corte Clássico	1 / S / I / R
Problema de Carregamento de Pallets	2 / T / U / (C ou *)
Problema de Carregamento de Veículos	1 / T / I / (M ou R ou C)
Problema de Balanceamento de Linha de Montagem	1 / S / I / M
Problema de Escalonamento de Multiprocessadores	1 / S / I / M
Problema de Alocação de Memória	1 / S / I / M
Problemas de Corte Geral	1, 2, 3 / * / * / *
Problemas de Empacotamento Geral	1, 2, 3 / * / * / *
Problema de Carregamento de Containeres	3 / S / U / * ou 3 / T / I / *
Problema de Alocação de Tarefas	N / T / I / *

Tabela 2.1 – Alguns problemas de C&E e tipos combinados correspondentes.

Como se torna aparente pela notação empregada, os problemas clássicos de empacotamento de bins, corte e carregamento de veículos pertencem ao mesmo tipo combinado considerando as 3 primeiras características. O que muda essencialmente entre eles é a forma de alocação das unidades. Considera-se, por exemplo, em Chvatal [14], que no problema de corte geral a lista de demanda possui muitos itens, porém relativamente poucos deles têm tamanhos diferentes. Para o problema de empacotamento de bins, ao contrário, considera-se que existam muitos itens a serem manuseados, onde muitos deles possuem tamanhos diferentes. No carregamento de veículos somente poucos itens são considerados. Mas são conceitos relativos sem um limite definido quando usar um ou outro termo para o mesmo problema.

Por outro lado, balanceamento de linha, escalonamento de multiprocessadores e alocação de memória pertencem ao mesmo tipo combinado como o clássico empacotamento de bins. A diferença entre eles refere-se a outras características além

destas básicas. Por exemplo, no balanceamento de linha, além destas características existe uma ordem de precedência para os itens.

Diversos tipos de análises têm sido feitos para avaliar o desempenho dos algoritmos de corte e empacotamento. A maioria é feita com a comparação de um valor ótimo com o valor obtido pelo processo. As principais linhas de análise seguidas são as combinatórias, ou análise de pior caso, as probabilísticas e as experimentais.

As análises combinatórias se baseiam na garantia de um desvio máximo (em relação à solução ótima) da solução aproximada obtida por uma determinada heurística quando aplicada para uma dada classe de instâncias.

Nas análises probabilísticas, assume-se uma função de densidade para as instâncias do problema e estabelecem-se propriedades probabilísticas da heurística. Estuda-se o desempenho esperado da heurística ou um limite para a probabilidade de a heurística encontrar uma solução dentro de uma percentagem de optimalidade pré-determinada.

Por último, nas análises experimentais tem-se a comparação experimental do desempenho de vários algoritmos quando aplicados a determinadas instâncias do problema [40].

2.2.2 – Empacotamento Uni-dimensional (1D)

Esta é a versão mais simples, porém computacionalmente complexa, dos problemas de empacotamento. Em sua versão clássica, temos como objetivo minimizar o número de bins (caixas, vagões, trilhas de um disco rígido, contêineres, etc.), de igual capacidade, para acomodarmos uma lista de itens. Como o problema é uni-dimensional, estaremos levando em consideração apenas uma dimensão, a qual não deve ser ultrapassada pelo somatório dos itens. Esta afirmativa está ilustrada na Figura 2.2, onde:

- C é a capacidade do bin (no exemplo, $C = 1$);
- L é uma lista de n objetos;
- N é o número de bins.

No problema representado pela Figura 2.2, deseja-se alocar um conjunto de tarefas (sem restrição de precedência) em um conjunto de máquinas, ou processadores

(computação paralela), de modo que não se ultrapasse um tempo limite estabelecido (deadline), utilizando o menor número de máquinas possível [11].

Desta forma, o problema de Otimização é estabelecer o modo de se empacotar os itens da lista *L*, em *N* ou menos *bins*, tendo estes capacidades máximas *C* [4].

Abaixo, estão relacionadas algumas aplicações para o problema do empacotamento uni-dimensional:

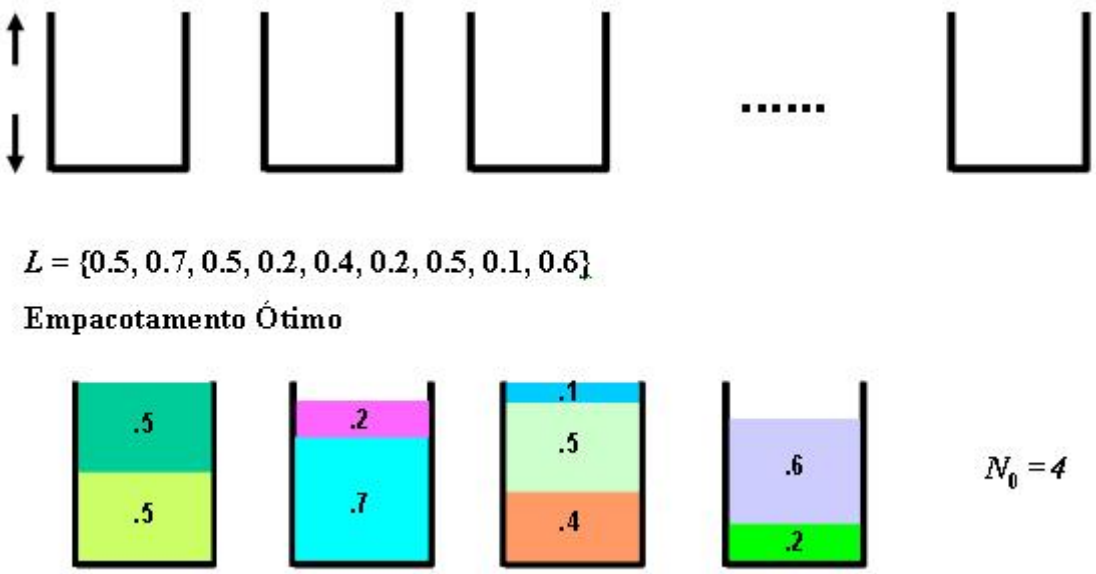

Figura 2.2 – Empacotamento Uni-dimensional

- **Alocação de Comerciais em TV** - têm-se *n* comerciais (p_1, p_2,..., p_n) para televisão, tendo cada comercial p_i uma duração de s_j segundos. Esses comerciais devem ser apresentados em intervalos de um programa de televisão, onde cada intervalo possui *T* segundos. Deseja-se alocar esses comerciais de modo a diminuir o número de intervalos necessários para apresentar esses comerciais.

- **Alocação de Programas em Discos e Fitas Magnéticas** - programas de computador de tamanhos p_1, p_2,..., p_n, cada programa p_i com tamanho s_j *bytes*, devem ser colocados em trilhas ou setores de discos magnéticos de forma a usar o menor número de trilhas para gravar os *n* programas.

- **Carregamento de Veículos -** carregar n cargas com peso s_i ($i=1, 2, ...,n$), em veículos com limite de peso P, de maneira a não violar a restrição de lotação de cada veículo, com o objetivo de minimizar o número de veículos necessários.

- **Escalonamento de Tarefas -** minimizar o número de máquinas necessárias para completar n tarefas ($p_1, p_2,..., p_n$), em um dado limite de tempo T. Sabe-se que cada tarefa p_i requer um tempo s_j para ser executada.

- **Problema da Programação de Veículos -** cargas (por exemplo, jornais) devem ser transportadas por veículos, a partir de um depósito até os pontos de entrega em, no máximo, T horas. Cada veículo pode fazer várias viagens. Programar n viagens com tempo de duração s_i horas ($i=1, 2, ..., n$), de maneira a não atrasar a entrega das cargas e com o objetivo de minimizar o número de veículos necessários.

- **Corte de Bobinas (papel, aço,...) -** bobinas (por exemplo, de papel), de comprimento C são produzidas por uma fábrica e devem ser cortadas em diversos rolos de comprimentos $s_1,..., s_n$, de forma a minimizar o número de bobinas necessárias.

- **Corte de Vigas (em madeireiras, construção civil,...) -** barras (por exemplo, de ferro), de comprimento C, devem ser cortados em diversas barras menores de comprimentos $s_1,..., s_n$. O objetivo é cortar o menor número possível de barras de comprimento C.

- **Pré-paginação -** frações de páginas de memória (de computador) de tamanhos $s_1,..., s_n$, devem ser alocadas em páginas de tamanho C *bytes* (frações de páginas são requeridas por segmentos de programas e estes devem aparecer em um menor número possível de páginas, por exemplo, *loops* internos, *arrays*). Encontrar uma alocação que minimize o número de páginas de tamanho C.

2.2.3 – Empacotamento Bi-dimensional (2D)

Este problema pode ser descrito como o de dispor objetos bi-dimensionais de diferentes áreas ($p_1, p_2, ..., p_n$) dentro de um outro objeto também bi-dimensional de área fixa (Figura 2.3), de tal forma que não haja sobreposição dos mesmos e que a área total desta disposição seja mínima [11], ou mesmo que o aproveitamento desta determinada área seja máximo, caso o número de objetos seja ilimitado.

Comumente se encontra na literatura o empacotamento bi-dimensional em placas e faixas.

O empacotamento 2D em placas consiste em empacotar uma lista de retângulos $L = (r_1, ..., r_n)$, onde $r_i = (x_i, y_i)$, em retângulos ou placas $R = (a, b)$, de forma a minimizar o número de placas usadas.

Já o problema do empacotamento 2D em faixas consiste em, dados uma lista de retângulos $L = (r_1, ..., r_n)$, onde $r_i = (x_i, y_i)$, e um retângulo $R = (a, \infty)$, empacotar os retângulos de L em R, minimizando o tamanho do empacotamento na direção ilimitada do retângulo R [40].

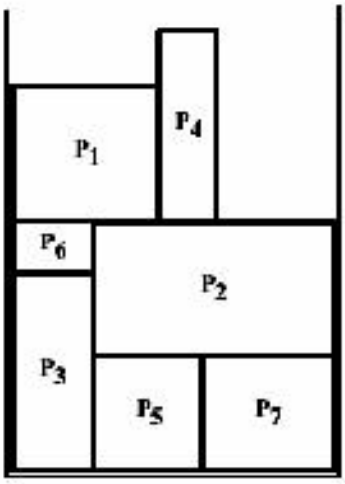

Figura 2.3 – Empacotamento Bi-dimensional.

À seguir, estão listados algumas aplicações do problema de empacotamento 2D:

- **Corte de placas (vidro, chapas, madeira,...)** - placas P de dimensões (a, b) devem ser cortadas em placas menores de vários tamanhos $(r_1,..., r_n)$. O objetivo é minimizar o número de placas P necessárias para cortar as placas menores.

- **Corte de retalhos em fábrica de tecidos, ou em confecção de roupas -** retalhos de tecidos retangulares (r_1,..., r_n), devem ser cortados em um rolo de tecido de largura l, objetivando-se minimizar o comprimento do rolo de tecido a ser cortado.

- **Corte de películas de filme (foto) -** uma película de filme de largura l deve ser cortada em n fotos de tamanhos retangulares, objetivando-se minimizar o comprimento da película de filme usada.

- **Escalonamento de tarefas em sistemas paralelos -** um conjunto de n programas r_1,..., r_n devem ser executados dispondo-se de uma quantidade de recurso R. Cada programa $r_i = (w_i, h_i)$ necessita de w_i unidades do recurso e leva h_i unidades de tempo para ser executado. O objetivo é minimizar o tempo necessário para executar os n programas.

2.2.4 – Empacotamento Tri-dimensional (3D)

No Problema do Empacotamento Tri-dimensional, foco desta pesquisa, deseja-se dispor objetos tri-dimensionais de diferentes volumes (v_1, v_2, ..., v_n) dentro de um outro objeto também tri-dimensional de volume fixo (v_f), de tal forma que o volume total desta disposição seja mínimo [11].

Podemos verificar dois casos particulares desta versão de empacotamento. Uma deles é relacionada a contêiner, e a outra, a *pallet*. De modo geral, considera-se contêiner uma caixa, e *pallet* uma espécie de "bandeja", ou suporte. No caso dos *pallets*, iremos considerar que sua altura é ilimitada, de modo que a preocupação não será minimizar a sua quantidade e sim minimizar a sua altura de empacotamento.

O problema do empacotamento 3D em contêineres (ver Figura 2.4) consiste em: dado uma lista de caixas $L = (b_1, ..., b_n)$, onde $b_i = (x_i, y_i, z_i)$, e contêineres $C = (d_1, d_2, d_3)$, empacotar as caixas de L dentro de contêineres C, maximizando o aproveitamento dos contêineres (ou minimizando o seu número).

Já o problema do empacotamento 3D em pallets consiste em: dado uma lista de caixas $L = (b_1, ..., b_n)$, onde $b_i = (x_i, y_i, z_i)$, e pallets $P = (d_1, d_2, \infty)$, empacotar as caixas

de *L* nos pallets *P*, minimizando o tamanho do empacotamento na direção ilimitada do recipiente.

Além da subdivisão do problema em contêineres e *pallets*, podemos dividi-lo em mais duas subclasses: o Problema do Produtor (PP) e do Problema do Distribuidor (PD).

No PP, dispomos de um contêiner que deve ser carregado com vários itens com o mínimo de variação com relação à sua altura, largura e comprimento. Como, por exemplo, um contêiner de volume 100x120x104m^3 que precise ser carregado com 100 caixas, 50 com 4x5x4m^3, e 50 com 3x10x2m^3. Este problema se refere ao produtor porquê subentende-se que este fabrica um determinado número de itens iguais, e estes devem ser empacotados para sua futura distribuição.

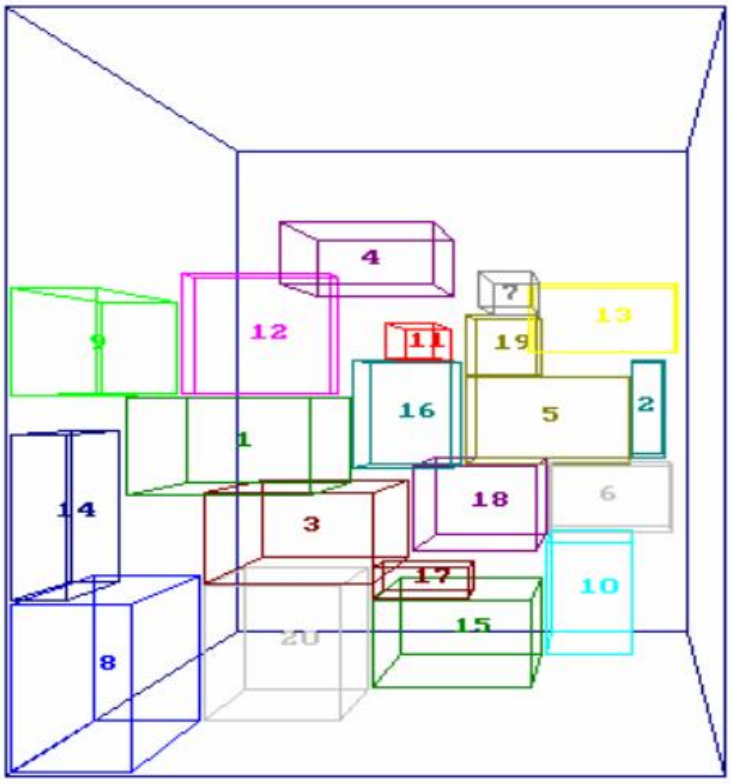

Figura 2.4 – Empacotamento Tri-dimensional.

No PD, dispomos de um contêiner que deve ser carregado com vários itens de tamanhos variados. Este problema se refere ao distribuidor porquê subentende-se que, quem distribui (geralmente supermercados, atacadistas, etc.), o faz com diversos tipos de produtos diferentes.

Alguns exemplos de onde pode ser empregado o empacotamento 3D:

- **Carregamento de cargas em furgões -** carregar cargas $c_1,..., c_n$ em veículos de transporte com dimensões de (a, b, c). O objetivo aqui é minimizar o número de veículos necessários para transportar as n cargas.

- **Empacotamento de caixas em galpões -** caixas $c_1, ..., c_n$ devem ser armazenadas em um galpão com fundo a x b, de forma a minimizar a altura da disposição final das caixas [40].

2.3 – A Heurística de Suavização de Superfícies Irregulares (HSSI)

2.3.1 – Descrição da Heurística

A heurística aqui aplicada, denominada de "Heurística de Suavização de Superfícies Irregulares" (HSSI), foi desenvolvida por Gonzaga & Bittencourt [3] para o problema do BP3D com o intuito de simular um empacotamento seguindo o comportamento da maioria das pessoas ao fazer este processo na vida real.

O cotidiano revela que as pessoas, ao realizarem um empacotamento qualquer, em que vários itens diferentes são postos a sua escolha, têm a tendência de escolher sempre as caixas cujas dimensões se aproximem o máximo possível do espaço disponível para o empacotamento. No caso de ainda existir espaço desocupado (buraco), a próxima caixa a ser escolhida é aquela que preencha tal espaço de modo que o topo de ambas as caixas formem uma superfície sem descontinuidade.

Baseando-se na idéia exposta acima, a HSSI reflete esse mesmo comportamento. Diante de um conjunto de caixas a serem empacotadas, verifica-se as dimensões dos espaços disponíveis, as caixas que podem preencher esse espaço, considerando todas as suas orientações e suas respectivas restrições (poder ser tombada ou sobreposta), e elege-se a melhor caixa de modo a garantir uma suavização da superfície.

Esta heurística basicamente constrói um novo empacotamento em cada iteração. Ela não limita o número de caixas diferentes em cada camada, apenas escolhe as caixas de modo a reduzir os buracos não ocupados em cada camada.

2.3.2 – Funcionamento da Heurística

A HSSI é baseada na construção de camadas, de modo que, praticamente, subdivide o problema do BP3D, diminuindo a dimensão do trabalho a ser realizado. Ao invés de o algoritmo trabalhar com o volume total do contêiner, ele se restringe a uma região (camada) determinada. Após a resolução de uma camada, o algoritmo procura uma nova camada. Isso é feito até que, com a junção dessas camadas, contemple-se todo o contêiner.

Como dito anteriormente, a heurística proposta tem o objetivo de fazer o empacotamento de modo a deixar a superfície suavizada. Desse modo, deve-se definir, de tempos em tempos, qual será a camada trabalhada (Figura 2.5). No contêiner, as dimensões x e z (largura e comprimento) da camada se mantêm constantes, e a dimensão y (altura) vai ser aquela escolhida (para formar a camada) com a análise de todas as dimensões das caixas disponíveis. Essa escolha é feita de modo a priorizar uma dimensão que mais se aproxime de pelo menos uma das dimensões das outras caixas, com o objetivo de diminuir o máximo possível as irregularidades na superfície.

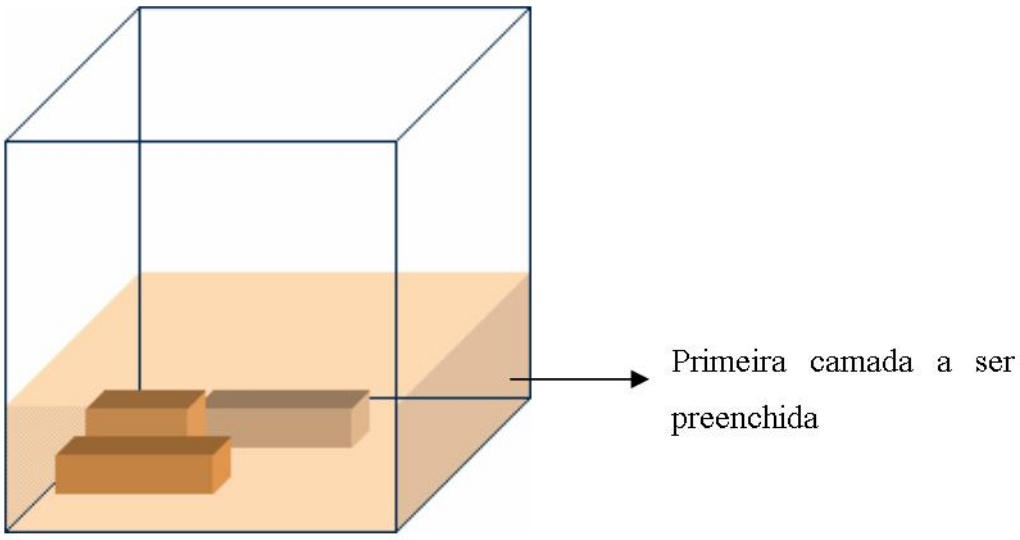

Figura 2.5 – Representação da primeira camada do contêiner.

A escolha do valor da primeira camada (sua altura) representa a relação de quão perto esta (que obrigatoriamente coincidirá com a dimensão de uma das caixas a ser empacotadas) está para altura de todas as outras caixas.

Para isso, cria-se uma lista candidata de todos os possíveis valores (altura) das camadas. A elaboração de tal lista é feita da seguinte forma [9]:

1. Para cada orientação do contêiner observa-se a sua dimensão *y*;
2. Para todas as caixas, e todas as suas dimensões menores que a dimensão *y* do contêiner, faz-se um somatório dos valores absolutos da subtração da dimensão (da caixa) analisada com a dimensão das demais caixas;
3. Guardamos esta relação encontrada (a dimensão atualmente analisada e o somatório descrito anteriormente) como um novo elemento na lista;
4. Fazemos, então, uma ordenação crescente de todos os valores dos somatórios encontrados e elegemos como camada inicial a dimensão correspondente ao primeiro elemento da lista.

Com a primeira camada determinada, o próximo passo é escolher a caixa mais apropriada para ser empacotada. Isso é feito da seguinte forma: seja *Hy* o tamanho da camada corrente. Deve-se escolher uma caixa de dimensão que priorize a seguinte ordem: seja o mais próximo possível de *Hy,* porém menor que *MaxY*; o mais próximo possível de *MaxX,* sem ultrapassar tal valor; e, por último, o mais próximo possível de *Hz,* porém não maior que *MaxZ*. A Figura 2.6 dá uma idéia do processo.

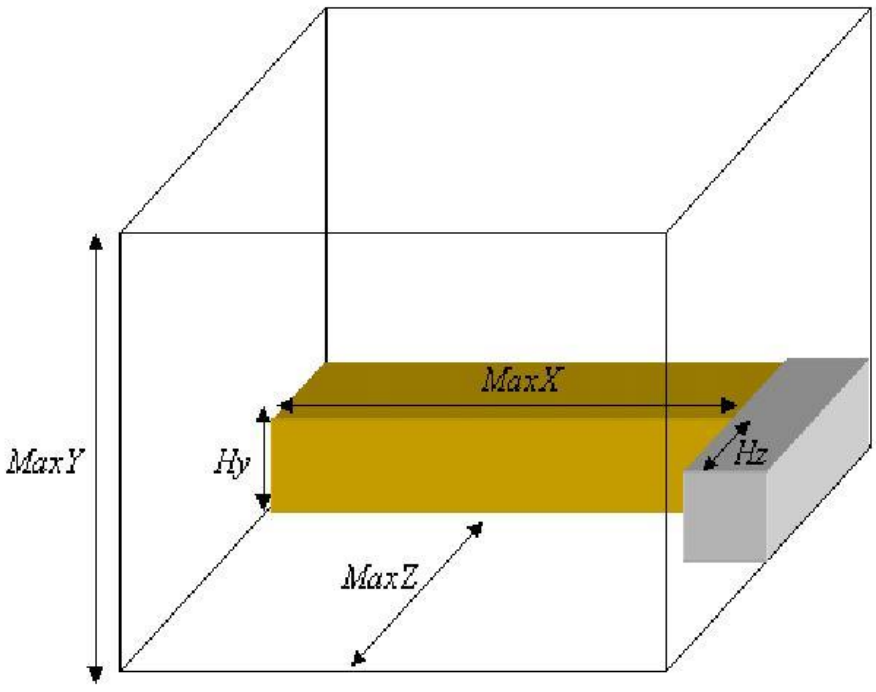

Figura 2.6 – Parâmetros tratados no algoritmo.

2.3.3 – Construção de Subcamadas

Existe a possibilidade de se formarem áreas livres em uma camada devido à escolha de uma caixa de altura maior que o valor da camada utilizada. Chamamos essas áreas de "subcamadas". Uma outra característica importante na HSSI é o tratamento dado a essas subcamadas.

Sempre com a idéia de tentar deixar a superfície o mais regular possível, quando se coloca uma caixa α que ultrapasse a altura da camada considerada, será criada uma subcamada. Neste caso, o problema se restringe temporariamente à região da subcamada formada, empacotando-se tantas caixas quanto possível até o seu limite. Terminando esse processo, retorna-se à consideração normal da camada. Ressaltando que esta camada assume uma altura suficiente para englobar caixa α. A Figura 2.7 abaixo mostra o que foi dito.

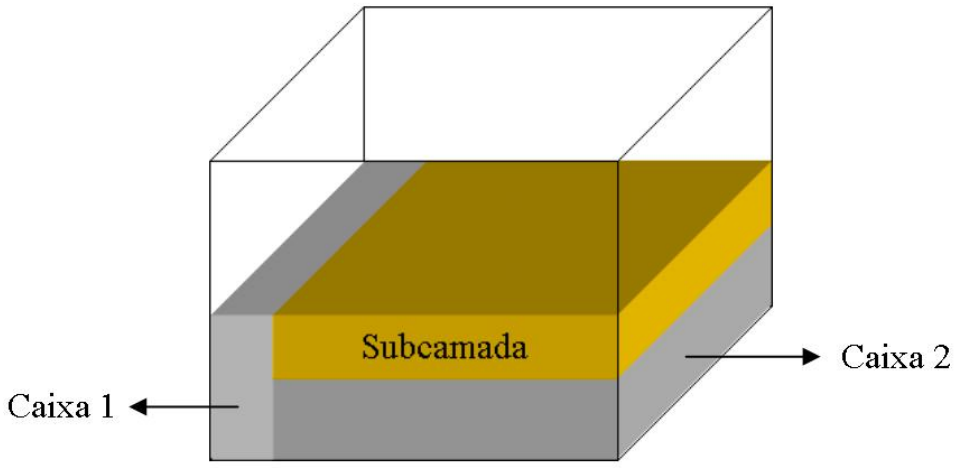

Figura 2.7 – Representação de uma subcamada.

2.3.4 – Tombo e Sobreposição

Apesar de o problema permitir caixas com diferentes orientações, no cotidiano encontramos certas caixas que não podem tombar, ou seja, serem colocadas "de cabeça para baixo". A HSSI, tentando retratar o máximo possível a realidade, considera tal restrição e impede que estas caixas, classificadas no arquivo de entrada como impróprias ao tombo, tenham todas as suas orientações analisadas.

Assim, o algoritmo avalia apenas duas (do total de seis) possíveis orientações da caixa, de acordo com a disposição desta fornecida pelo usuário, como mostrado na Figura 2.8.

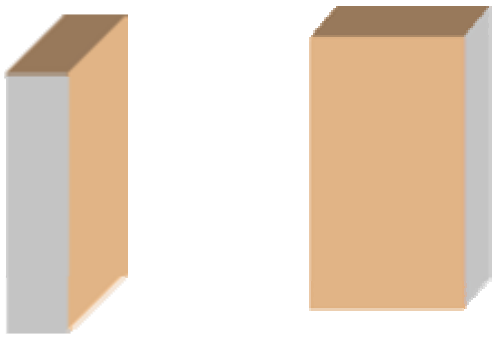

Figura 2.8 – Possíveis orientações de uma caixa restrita ao tombo ($t = 0$).

Uma outra restrição que a HSSI considera é o fato de que certos materiais são bastante frágeis, de forma que, ao empacotá-los, não devam ser colocadas outras caixas sobre eles.

Estas caixas tidas como frágeis são analisadas como caixas normais no decorrer do algoritmo. Entretanto, se escolhidas, para garantir que nenhuma outra caixa se sobreponha a elas, é criada a falsa impressão de que elas preenchem todo o espaço acima delas. Isso é feito fazendo-as crescerem infinitamente de forma fictícia de modo que, para o algoritmo, sua dimensão y completa todo o espaço restante até o topo do contêiner, como mostra a figura 2.9.

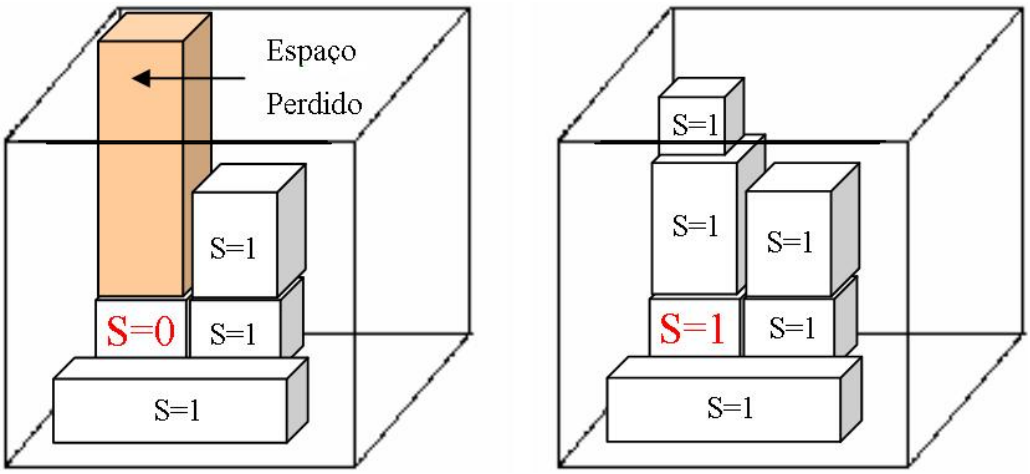

Figura 2.9 – Representação de uma caixa não podendo e podendo ser sobreposta.

2.4 – Otimização Interativa

2.4.1 – Introdução

Durante muito tempo, os avanços nos sistemas designados para resolver problemas de otimização, como roteamento, escalonamento e layout, se limitaram a desenvolver algoritmos de execução automática, onde o papel do usuário era apenas o de especificar o problema e definir critérios e restrições. Tal abordagem desprezava aspectos reais das tarefas de otimização, que são essenciais para obter soluções úteis.

Usuários do sistema devem entender e confiar na solução gerada para fazer uso efetivo delas. Além disso, é praticamente impossível especificar em detalhes (modelar) todas as restrições apropriadas e critérios de seleção para todos os possíveis cenários de um problema real.

Considere, por exemplo, alguém produzindo um plano de trabalho mensal. Essa pessoa deve entender a solução para conduzi-la aos empregados afetados por este plano. Além disso, ela deve entender como fazer modificações com as novas necessidades que surgirem. Esta pessoa provavelmente não pode transferir toda a sua experiência em avaliar soluções candidatas para o computador. Desse modo, métodos automáticos podem produzir relações que conflitam com a sabedoria acumulada das pessoas que os utilizam [58].

Uma forma de contornar esses problemas, e melhorar o processo de otimização, é a construção de sistemas que envolvem pessoas no processo, permitindo a elas guiarem ou dirigirem um algoritmo de otimização. Nesta abordagem, conhecida por Otimização Interativa, levamos em conta que os usuários são mais aptos a entender uma solução que eles ajudaram a criar do que uma que é simplesmente apresentada a eles. Além disso, em um sistema interativo, já que um usuário melhor entende as escolhas avaliadas, ele pode modificar o critério de seleção da solução tão bem quanto dirigir o computador em direção a soluções que são mais apropriadas em prática [58].

O que a Otimização Interativa nos proporciona não é somente a interação propriamente dita, mas também o conhecimento pleno do problema, o qual seremos capazes de modificar, além justificar essas modificações. De acordo com Nascimento [45], existe um número de razões para usarmos a Otimização Interativa:

- O problema pode depender do domínio de conhecimento, incluindo as qualidades da solução, que são subjetivas para o usuário; este tipo de conhecimento pode ser difícil de descrever precisamente;
- O processo de Otimização pode ser dinâmico, no sentido de que alguns objetivos e restrições são previamente desconhecidos no início do processo;
- Métodos de Otimização disponíveis para resolver o problema talvez não tratem todos os objetivos e restrições, apenas um pequeno subconjunto deles;
- Recursos computacionais (poder de processamento e tamanho da memória) talvez não sejam suficientes para resolver o problema devido a sua complexidade ou a um grande número de dados que devem ser manipulados.

Devido a essas razões, a interação do usuário se faz necessária. Existem também diferenças bem conhecidas entre habilidades humanas e de máquina para a resolução de problemas, o qual pode ser explorado pelo uso de interação. Computadores são adequados para computação intensiva, onde soluções para um problema são criadas e numericamente avaliadas; humanos, por outro lado, são bons em identificar padrões que diferenciam soluções boas de más. A maioria dos algoritmos de otimização, para problemas NP – difíceis, facilmente acham um ótimo local (ou seja, soluções na qual não há melhora local), mas têm dificuldades em achar o ótimo global (soluções que são melhores que todas as outras soluções). Com uma boa representação visual, usuários podem perceber caminhos à percorrer para se chegar a uma melhor solução [45].

Em geral, a maioria dos problemas de otimização real é manipulada em dois passos [45]: primeiro, uma ferramenta automática é aplicada, e então uma pessoa manualmente melhora a solução em um estágio de pós-processamento. Existe, então, um interesse em uma relação mais intensa entre os papéis desempenhados por humanos e ferramentas automáticas. Podemos considerar “ferramentas automáticas” como uma extensão natural das capacidades humanas que devem ser usadas não apenas no estágio inicial de uma tarefa de otimização, mas através do processo inteiro.

A seguir, é apresentada uma visão geral sobre "Interação Humano-Computador" (IHC), que complementa o que foi falado nesta seção. Mais adiante, serão apresentadas algumas abordagens relativas à Otimização Interativa.

2.4.2 – Interação Humano-Computador (IHC)

A Interação Humano-computador (IHC) já vem sendo estudada por certo tempo, e hoje já existem normas de procedimento utilizáveis para o desenvolvimento de sistemas interativos "amigos do usuário" [44]. Shneiderman [60] sugeriu, entretanto, substituir este termo por algumas metas mais claras de fatores humanos, que são:

- **Tempo de aprendizagem –** minimiza a quantidade de tempo necessário para aprender a usar o sistema;
- **Velocidade de desempenho –** reduz a quantidade de tempo necessário para desempenhar uma tarefa com o sistema;
- **Grau de erros de usuários –** minimiza o grau de erros quando as tarefas são desempenhadas;
- **Memorização através do tempo –** proporciona aprendizagem, então os usuários ainda lembram como usar o sistema após um longo tempo sem operá-lo;
- **Satisfação subjetiva –** incrementa a satisfação do usuário com o sistema.

Essas metas podem ser alcançadas adotando certos princípios, os quais já têm sido aplicados com sucesso em muitos sistemas:

- Implementar formas consistentes e compatíveis para o usuário entrar com dados no sistema e para o sistema mostrar os dados;
- Permitir aos usuários freqüentemente usarem atalhos para agilizar suas ações principais;
- Reduzir o carregamento de "memória de curto prazo", que é a quantidade de informação que o usuário tem que lembrar quando desempenha uma tarefa com o sistema. Isto pode ser feito adicionando informação extra à interface a fim de ajudar o usuário;
- Oferecer um *feedback* informativo para as ações do usuário;

- Prevenir o usuário de cometer erros sérios ou ajudá-los a corrigir o problema sem ter que refazer o trabalho inteiro;
- Permitir reversão fácil das ações.

Existem muitos sistemas com vários estilos de Interação Homem-Computador, como seleção de menus, formulários de preenchimento, abordagens baseadas em comandos de linguagem, linguagem natural e manipulação direta [60]. Neste trabalho são usados principalmente dois estilos: manipulação direta e seleção de menus. A seguir, será mostrada a descrição destes estilos.

Manipulação Direta

A Manipulação Direta possibilita ao usuário desempenhar tarefas interagindo diretamente com os elementos gráficos existentes na representação visual do problema. As duas principais características desta abordagem são:

- Uma representação contínua dos objetos e ações de interesse;
- O uso de ações físicas, como um clique de botões e pegar e arrastar objetos na tela, ao invés de usar comandos de sintaxe complexos.

Além disso, manipulação direta de objetos permite operações que têm um efeito visível imediato no objeto de interesse. Exemplos de sistemas de manipulação direta incluem editores de texto, planilhas eletrônicas, sistemas de projeto auxiliado por computador (CAD), jogos e sistemas de informação geográfica [44].

De acordo com Shneiderman [60], os benefícios da manipulação direta são:

- Os usuários novatos podem aprender funções básicas rapidamente interagindo com o sistema, ou por demonstrações feitas por um usuário mais experiente;
- Usuários intermediários encaram a abordagem como se ela tornasse os conceitos operacionais mais fáceis de lembrar e permitissem uma execução rápida de tarefas;
- Usuários recebem um retorno imediato de suas ações, podendo mudar a direção da sua atividade se esses resultados não forem positivos.

Uma analogia feita pelo próprio Shneiderman [60] diz que "usar manipulação direta é como dirigir um automóvel. Além da cena ser diretamente visível através do pára-brisa, a execução de ações como parar ou dirigir têm se tornado conhecimento comum em nossa cultura. Para dobrar a esquerda, o motorista simplesmente gira o volante para a esquerda. A resposta é imediata, e a cena muda, fornecendo um feedback para refinar a mudança".

Existem, entretanto, problemas com a abordagem de manipulação direta, os quais são mostrados no uso de uma representação visual [44]:

- A representação pode ser muito grande e levar mais que o espaço visível da tela que esta sendo mostrada. Portanto, pode ser necessária uma rolagem repetitiva e tediosa da visualização;
- A representação visual pode ter um significado para o projetista do sistema, mas não para o usuário final;
- A representação pode também ser enganosa. Por exemplo, o usuário pode entender seu sentido geral, mas pode incorretamente interpretar como interagir com ele;
- Interação direta com a representação visual pode não ser tão eficiente como digitar um comando para alguns problemas particulares.

Apesar desses problemas, manipulação direta ainda aparece como um estilo de interação natural e mais fácil de lembrar que outras abordagens, como uma seleção de menus e digitação de comandos. Este e o caso comumente aplicado para a maioria dos sistemas interativos. Manipulação direta pode também ser combinada com outras abordagens quando uma boa visualização para alguns objetos de interesse não puder ser achada [44].

Seleção de Menus

Em sistemas de Seleção de Menus, usuários lêem uma lista de itens, selecionam o mais apropriado para a sua tarefa, e observam o efeito. Se a terminologia e o significado dos itens são compreensíveis e distintos, então os usuários podem concluir suas tarefas com pequeno aprendizado ou memorização e apenas poucas ações. O grande benefício é o fato de que pode existir uma estrutura clara e objetiva para tomadas

de decisão, desde que todas as escolhas possíveis sejam apresentadas de uma só vez. Este estilo de interação é apropriado para usuários novatos e intermediários e pode ser atraente para usuários freqüentes se os mecanismos de seleção e visualização forem rápidos.

Para desenvolvedores, sistemas de seleção de menus requerem uma análise de tarefas cuidadosa para assegurar que todas as funções sejam suportadas facilmente e que a terminologia seja escolhida cuidadosamente e usada consistentemente. Menus são eficazes porque eles oferecem dicas para deduzir o reconhecimento do usuário, ao invés de forçar o usuário a recordar a sintaxe de um comando. Usuários indicam suas escolhas com um dispositivo de apontamento e recebem um feedback indicando o que ele fez. Seleção simples de menu é especialmente eficaz quando usuários têm pouco treinamento, usam o sistema intermitentemente, não são familiarizados com a terminologia, ou precisam de ajuda em estruturar seus processos de tomada de decisões. Com um design cuidadoso de menus complexos e interação de alta velocidade, seleção de menus pode vir a ser atraente até para usuários freqüentes experientes.

Entretanto, só porque um designer usa uma seleção de menus, não há garantia que a interface será atraente e fácil de usar. Interfaces eficazes emergem apenas após uma consideração cuidadosa de numerosos itens de design, como organização de tarefas relacionadas, seqüência dos itens, tempo de resposta, correção de erros, entre outros [60].

2.4.3 – Processos de Avaliação

Todos os sistemas interativos envolvem tarefas que devem ser desempenhadas pelo usuário, assim como as tarefas que devem ser executadas pelo computador. Um ponto importante para o projetista de tais sistemas e achar um bom balanceamento entre automação e controle humano [44]. Shneiderman [60] recomenda simplificar o papel do usuário pela eliminação da ação humana quando não é preciso julgamento, e evitando tarefas repetitivas, tediosas e propícias a erros. Ao invés disso, o usuário deve se concentrar em tarefas criativas, decisões críticas, planejamento de estratégia, e competição com situações inesperadas. O computador, por outro lado, deve ser usado para manipular grandes volumes de dados, executar repetitivas ações programadas com segurança, monitorar e controlar eventos pré-especificados, e executar complexas operações lógicas e matemáticas.

O projeto de um sistema interativo deve ser testado para verificar se ele atende as metas do fator humano especificadas previamente. O sistema pode ser testado não apenas após a implementação, mas também em estágios iniciais do seu desenvolvimento de forma a ajudar com a escolha e a validação do projeto de solução.

Várias abordagens existem para avaliar um sistema: entrevistas, discussões em grupo, visões gerais sobre os assuntos, experimentos controlados por humanos, etc.

Avaliações baseadas em experimentos humanos seguem metodologias da psicologia. Os passos básicos para este tipo de experimento envolvem declarar uma hipótese testável, construir uma configuração bem controlada, medir os aspectos de interesse com um número significante de assuntos, e analisar os resultados usando estatísticas.

Algumas vezes, grandes experimentos humanos não são possíveis porque a aplicação depende de um domínio de conhecimento especializado, e usuários especialistas são desinteressados ou não estão disponíveis em um número suficiente. Nielsen [47] sugere uma técnica de avaliação para esta situação chamada “avaliação heurística”.

Avaliação heurística é uma inspeção sistemática de uma interface de usuário projetada para problemas usuais. A avaliação é realizada por um pequeno número de avaliadores, os quais examinam a interface e tentam identificar bons e maus aspectos dela, de acordo com uma lista de princípios reconhecidos. Cada avaliador individual inspeciona a interface separadamente; apenas depois é que eles permitem comunicar e agregar seus achados. O melhor número de avaliadores depende da aplicação, mas tem sido recomendado usar em torno de cinco ou, pelo menos, três pessoas. Uma seção de avaliação individual geralmente leva entre uma ou duas horas, dependendo da aplicação. O resultado dos exames pode ser gravado em relatos escritos por cada avaliador, ou por um observador que está presente nas sessões e anota os comentários ditos pelos avaliadores. O observador pode também auxiliar com a operação do sistema (isto pode ser necessário quando o sistema não está completamente implementado, ou quando a interface é muito complexa e o avaliador não tem tempo de ser treinado para usá-lo). Além disso, o observador deve responder questões do avaliador sobre o funcionamento do sistema.

2.4.4 – Alguns tipos de Otimização Interativa

HuGSS (Human-guided Simple Search)

Um bom exemplo do uso de interação humana para problemas de Otimização é o paradigma cooperativo chamado *HuGSS (Human-guided Simple Search* – Busca Simples Guiada por Humano*)*, apresentado por Anderson *et. al.* [5].

HuGSS divide um processo de Otimização em duas sub-tarefas principais realizadas por entidades diferentes: o computador é responsável por achar um mínimo local usando uma busca escalonada simples, enquanto o usuário trabalha em escapar do mínimo local e dirigir a busca para uma melhor solução. O paradigma foi aplicado com sucesso para o problema do roteamento de veículo capacitado com limite de tempo. Um sistema foi desenvolvido onde o usuário pode também mudar uma solução existente para o problema manualmente, ou focalizar um método *Hill Climbing* (Subida da Encosta) para melhorar uma área particular da solução. Uma visualização da solução sendo melhorada fornece um retorno para o usuário e ajuda a decidir a próxima ação na tarefa de Otimização. Experimentos com o sistema mostraram que o desempenho de um algoritmo simples *Hill Climbing* pode ser significantemente aumentado pela interação do usuário. Outras investigações posteriores do paradigma *HuGSS* foram abordadas nesses últimos anos [37, 58].

HuGS (Human-guided Search)

O *HuGS* (*Human-guided Search* – Busca Guiada por Humanos) é uma evolução da abordagem HuGSS, introduzida por Anderson *et al.* [5], que foi considerada a primeira aparição de uma aproximação de interação geral para resolver problemas de otimização que agregam muitas idéias simultaneamente, como foi visto anteriormente.

Em 2002, o paradigma *HuGSS* original foi estendido por Klau *et al.* [35] para incluir o método de busca tabu. Experimentos foram conduzidos para quatro problemas de otimização: (1) o problema de minimização de cruzamentos de arestas para o desenho de camadas de grafos, (2) uma variante do problema do caixeiro viajante onde não existem requisitos para visitar cada localização, (3) uma versão simplificada do problema de cruzamento de proteínas, e (4) o problema de escalonamento de trabalho. Experimentos usando o paradigma demonstraram que uma busca tabu guiada por humanos podia produzir melhores resultados que uma versão não guiada do mesmo

método (com a busca tabu executando no problema inteiro para uma quantia considerável de tempo sem interferência humana). Os experimentos também mostraram que a busca tabu guiada superou os métodos *Hill Climbing* usados no primeiro paradigma *HuGSS* [5].

Desde que os resultados com a busca tabu foram promissores, a abordagem estendida foi então renomeada para estrutura de *Human-guided Search*. A nova estrutura *HuGS* inclui os métodos *Hill Climbing* originais e a busca tabu. As principais ações interativas que podem ser realizadas pelo usuário são:

- Mudanças manuais na solução atual;
- Convocar, monitorar e parar uma busca para uma melhor solução;
- Ajustar a busca pela definição de três níveis de mobilidade (baixa, média e alta) para os elementos do problema;
- Reverter a uma solução prévia ou pré-computada.

Otimização Evolutiva

O tema "Otimização Evolutiva" tem recentemente experimentado um crescimento notável. Novos conceitos, métodos e aplicações têm sido continuamente propostos e explorados para fornecer ferramentas e técnicas eficientes para resolver uma variedade de problemas de Otimização. Essas técnicas incluem Algoritmos Genéticos, Programação Genética, Estratégias Evolutivas e Programação Evolutiva, entre outras, as quais são inspiradas por modelos restritos de evolução natural.

Aplicações de Otimização evolutiva cobrem uma vasta extensão de problemas de engenharia, de Otimização, pesquisa operacional e outros campos relacionados.

Podemos listar algumas técnicas mais conhecidas de Otimização Evolutiva:

- Algoritmos Genéticos
- Programação Genética
- Estratégia Evolutiva
- Programação Evolutiva

2.4.5 – *User Hints* (Dicas de Usuário)

User Hints é uma abordagem interativa que tem o intuito de melhorar os processos colaborativos Homem-Computador. Nascimento [44], fez um estudo detalhado sobre esta abordagem em vários de seus trabalhos. Esta seção apresenta o *framework* (estrutura de trabalho) *User Hints* para problemas de otimização combinatória abordada na referida tese. A estrutura objetiva alcançar as duas metas interativas principais descritas, que são:

- Refinar o problema de otimização;
- Ajudar a convergir para soluções ótimas. Isto é alcançado via atributos combinatórios da Interação Homem-Computador, Otimização Combinatória, e Visualização da Informação.

A Figura 2.10 [44] mostra um diagrama do *framework User Hints*. Ele envolve nove elementos:

1. Um usuário que é especialista no domínio;
2. Um conjunto de objetivos (ou seja, funções objetivo);
3. Um conjunto de restrições que compõem um problema de Otimização combinatória;
4. Um módulo com métodos de Otimização;
5. Uma solução que está atualmente sendo melhorada, chamada "solução de trabalho" (isto é simplesmente uma atribuição de valores para as variáveis do problema);
6. Uma função qualidade (não representada na figura) que mede a qualidade da solução;
7. Um "agente de melhor solução" que salva internamente a melhor solução de trabalho computada até agora;
8. Uma ferramenta de visualização;
9. Uma visualização criada pela ferramenta de visualização que fornece um *feedback* sobre a solução de trabalho e o estado da Otimização. A conexão entre os elementos da estrutura é representada pelas setas.

Um processo de otimização usando o *framework User Hints* consiste em ajustar os objetivos e ter o usuário interagindo com os elementos da estrutura de forma a

produzir uma solução de boa qualidade. Mais precisamente, o processo de otimização funciona da seguinte forma [44]:

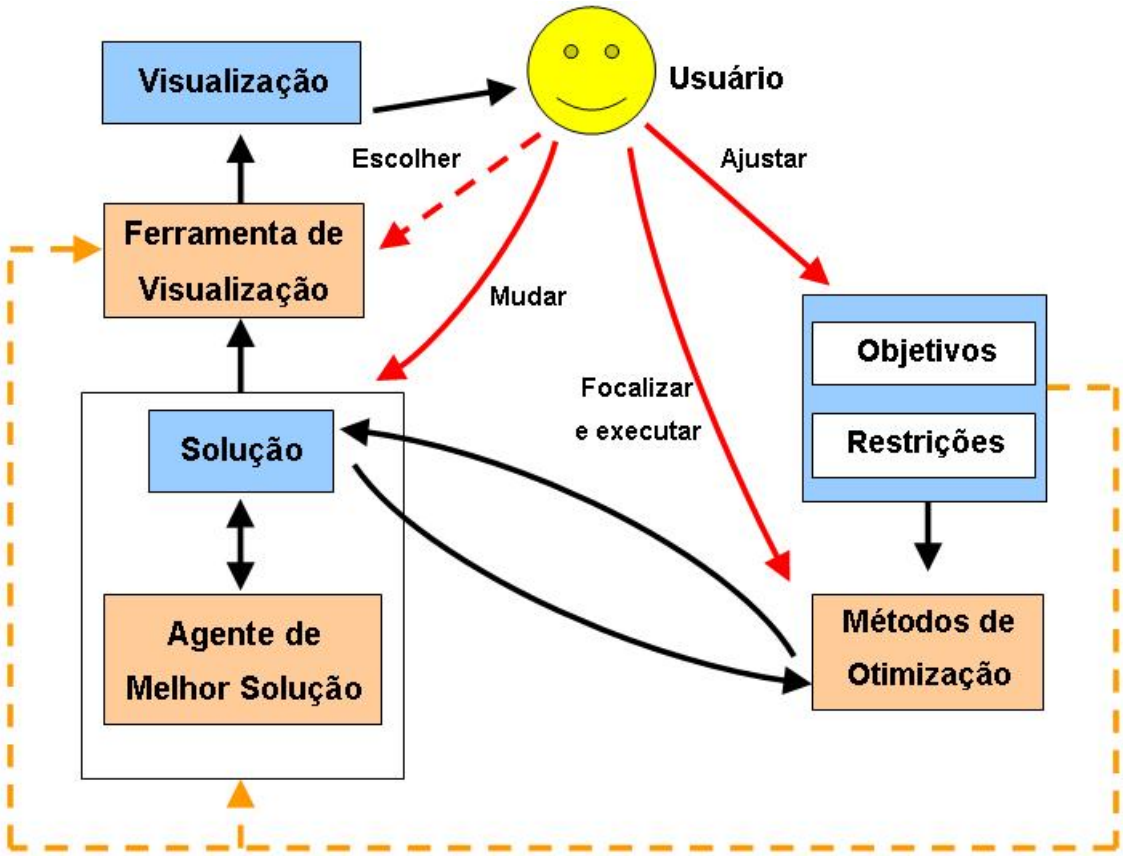

Figura 2.10 – Modelo *User Hints* de ferramenta de Otimização Interativa.

1. Uma solução de trabalho inicial é automaticamente criada, e a ferramenta de visualização é inicializada para fornecer a visualização dessa solução. O agente de melhor solução é também ativado, e ele salva uma cópia da solução inicial como a melhor solução;
2. O sistema então espera por uma ação do usuário, que é o controlador da otimização. O usuário pode também interagir com a "ferramenta de visualização" para escolher uma visualização diferente, ou dar "dicas" ao processo de otimização. Dicas são ajustes que objetivam refinar o problema, inserindo conhecimento de domínio para reduzir o espaço de soluções a serem exploradas, para escapar de um mínimo local, ou para imediatamente produzir uma melhor solução. Tais dicas podem mudar a solução de trabalho;
3. Qualquer mudança, tanto na solução de trabalho, quanto nas restrições ou nos objetivos, automaticamente engatilham a ferramenta

de visualização e o agente de melhor solução. O agente de melhor solução, por outro lado, compara a qualidade da solução de trabalho com a qualidade da melhor solução (salva internamente), e atualiza a melhor solução se necessário;

4. A qualidade da solução é determinada pela função de qualidade, a qual considera objetivos e restrições do problema;
5. Além disso, qualquer que seja a melhor solução, ela é atualizada, o agente de melhor solução engatilha a ferramenta de visualização e fornece um feedback imediato ao usuário sobre este evento.

No fim do processo de otimização, a melhor solução é considerada a solução final para o problema.

Este processo se repete até o usuário estar satisfeito com o resultado.

2.4.6 – Abordagens Interativas usando *User Hints*

Nesta subseção serão mostradas algumas abordagens interativas para problemas de Otimização Combinatória usando *User Hints*. Será apresentada uma visão geral sobre esses sistemas, limitando-se a mostrar o problema original e algumas funcionalidades interativas da abordagem, e sem adentrar em especificações técnicas ou o funcionamento de seus respectivos algoritmos.

User Hints para *Graph Clustering*

Graph Clustering (Agrupamento de Grafos) também é conhecido como *Graph Partitioning* (Particionamento de Grafos), e tem como objetivo dividir um conjunto de vértices de um grafo em subconjuntos disjuntos, que seriam os *clusters* (agrupamentos ou partições), enquanto minimiza-se a conexão entre vértices em conjuntos distintos, satisfazendo algumas restrições. Formalmente, um agrupamento S de um grafo $G = (N, E)$ (onde N é um conjunto de vértices valorados, e E, um conjunto de arestas valoradas) é uma partição de N em conjuntos disjuntos $N_1, N_2,..., N_k$. Existe um grande número de problemas de agrupamento. Um exemplo pode ser: ache um agrupamento S de G, sujeito a algumas restrições, de forma que o número de arestas entre os agrupamentos seja minimizada (ou seja, arestas (u, v) com $u \in N_i$, $v \in N_j$, $i \neq j$). As variações do

problema vêm da escolha de *k*, das restrições e do número de arestas entre agrupamentos. Por exemplo, uma variação comum é fixar o número *k* de agrupamentos e impor um "equilíbrio" no tamanho destes. Este problema é chamado de "Particionamento em *K*-modos" [21]. Quando $k = 2$, nós teremos um caso especial conhecido como "bisseção" ou "particionamento". Muitos outros tipos de restrições podem ser especificados para o problema, como, por exemplo, limitar o peso máximo dos vértices em cada agrupamento [44].

Quase todas as variações do problema de agrupamento de grafos são NP-Árduos (até mesmo o caso de bisseção [25]); por isso, são usados métodos heurísticos para fornecer soluções.

A interação humana com o problema *Graph Clustering* em geral acontece antes de executar os métodos – de forma a definir o problema – ou depois da execução – para refinar o problema quando ele precisar de ajustes. Neste último caso, o método é usualmente re-executado no problema modificado para produzir uma nova solução, descartando a solução anterior [44].

Nascimento [44] desenvolveu um *framework* interativo flexível onde usuários refinam um problema *Graph Clustering* e aperfeiçoam uma solução existente. Após a execução, o usuário pode visualizar o resultado atual e re-executar o algoritmo.

Neste sistema, o usuário realiza duas tarefas básicas:

1. **Inserir conhecimento do domínio -** a visualização da solução atual pode levar o usuário a acreditar que, embora isto satisfaça as restrições atuais e tenha um bom resultado para a função objetivo, isto não está certo no contexto do domínio. O usuário deve dar uma dica ao algoritmo para avançar em direção a uma solução que seja correta no contexto do domínio.
2. **Orientação da busca -** o usuário pode ver que o algoritmo está gastando muito tempo em pequenos ajustes que não trarão melhoras na solução. Neste caso, é possível realizar muitas interações de forma a ajudar o algoritmo a realizar melhoras.

As dicas que o usuário pode fornecer ao sistema são:

- **Ajuste de restrições –** mudando as restrições em tempo de execução, o usuário pode guiar os algoritmos para convergir à uma solução diferente. Exemplos de alterações de restrições são:

número mínimo ou máximo dos agrupamentos; limites da variação de tamanho do agrupamento; ou mesmo ter dois vértices particulares sempre no mesmo agrupamento, ou em diferentes agrupamentos, o que seria um ajuste de restrições locais.

- **Manipulação direta –** o usuário pode diretamente operar em um agrupamento, destruindo um existente ou fundindo dois deles. Um exemplo desta manipulação seria destruir um mau agrupamento e forçar o algoritmo a re-associar seus vértices à outros agrupamentos de modo a sair de um mínimo local.
- **Escolher outros tipos de métodos** - o usuário seleciona um algoritmo mais apropriado para o problema toda vez que o método atual não melhore a qualidade atual da solução. Neste caso, o usuário pode decidir se vai executar o algoritmo em todo o agrupamento ou focalizar apenas uma determinada área da solução que mostra uma baixa qualidade.

A Figura 2.11 [44] mostra o *framework* desenvolvido para o *Graph Clustering*.

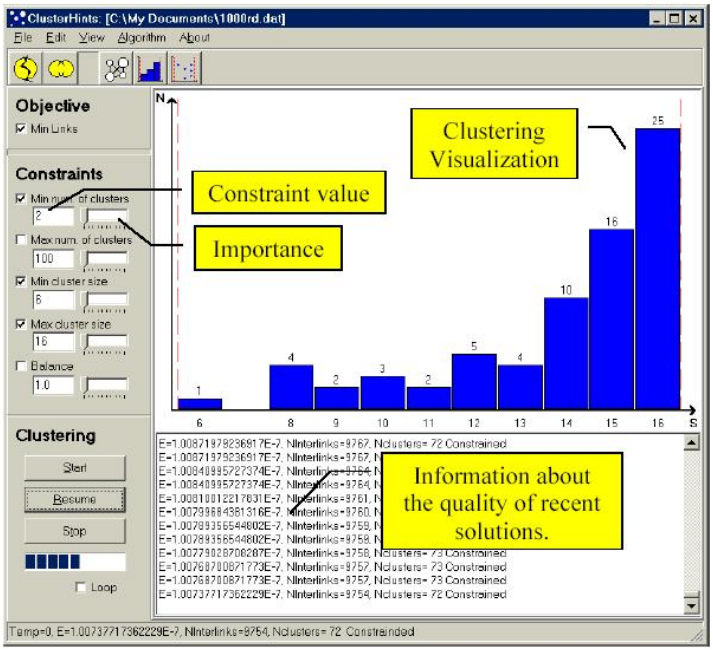

Figura 2.11 – *framework* desenvolvido para o *Graph Clustering*

User Hints para Desenho de Grafos Dirigidos

Desenho de grafos é uma área de pesquisa emergente com fortes aplicações em "Visualização da Informação". O seu objetivo é produzir visualizações "agradáveis" que ajudam a entender o relacionamento entre os vértices de um grafo.

Desenhos de grafos aparecem em diversos locais, como livros científicos, revistas, manuais técnicos; e em aplicações de software para ajudar a projetar, gerenciar, explorar e aprender processos. Quando um grafo contém apenas poucos vértices e arestas, ele pode ser desenhado manualmente com facilidade. Entretanto, quando se aumenta número de vértices e arestas, um procedimento manual de desenho pode consumir muito tempo e ser difícil de gerenciar. A solução para este problema é fazer uso de técnicas automáticas, as quais encaixam alguns critérios estéticos é aplicam algoritmos para achar desenhos esteticamente agradáveis. Alguns desses critérios mais comuns são: mostrar poucos cruzamentos de arestas; apresentar poucas curvas nas arestas (no caso em que as arestas podem ser curvadas; caso contrário, elas são desenhados como linhas retas); mostrar simetria; e minimizar a área necessária para o desenho. Também é desejável mostrar uma orientação uniforme das arestas, por exemplo, tendo elas apontando em declive na maioria dos casos. Embora esse conceito de "agradável" seja um pouco subjetivo, é demonstrado que esses critérios ajudam a melhorar a legibilidade dos diagramas [54].

A abordagem interativa *User Hints* aplicada a este tipo de problema deve ajudar a refinar o problema e melhorar a convergência, ajudando os algoritmos a buscar por desenhos de grafos de alta qualidade de acordo com um conjunto estabelecido de critérios estéticos.

A Figura 2.12 [44] abaixo mostra o *framework* para Desenho de Grafos usando *User Hints*.

Os tipos de interação para esta abordagem podem ser:

- **Focalização –** o usuário pode focalizar os algoritmos em um subconjunto de vértices de um desenho de grafo que estejam fora dos padrões estéticos estabelecidos previamente pelo usuário, reaplicando-os e obtendo uma nova solução. As posições dos vértices que estão fora deste subconjunto não serão mudadas;
- **Restrições de layout –** restrições de layout são úteis para ajudar o sistema a melhorar aspectos de má qualidade de um desenho, ou para remover ambigüidade sobre onde desenhar alguns vértices. Um exemplo

seria desenhar os vértices “de cima para baixo” e “da esquerda para a direita”;

- **Mudanças manuais –** outros aspectos de desenho que não são facilmente controlados por focalização e restrições de layout podem ser ajustados de maneira manual. O usuário realiza mudanças em vértices movendo-os para uma posição diferente do desenho. Em arestas, basta mover os vértices associados.

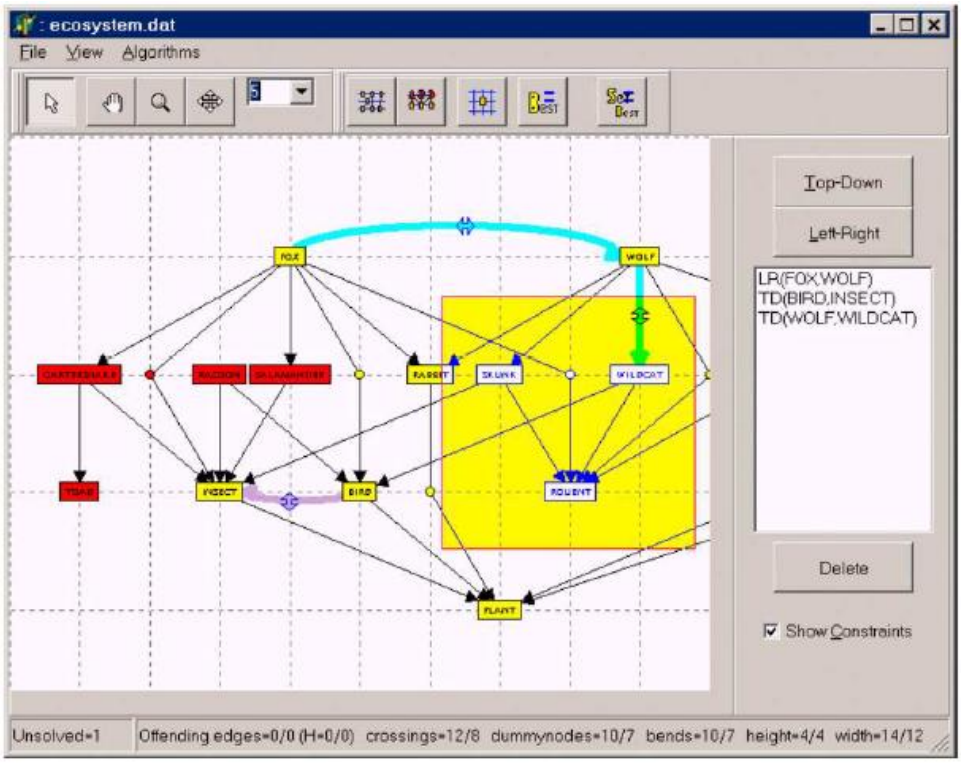

Figura 2.12 – *framework* desenvolvido para Desenho de Grafos.

User Hints para Otimização Interativa de Planos de Corte

O problema geral de empacotamento (já discutido anteriormente) aparece em uma grande variedade de contextos do mundo real. Ele pode ser caracterizado como uma lista de itens que precisa ser colocada em um conjunto de caixas, respeitando-se o limite de espaço nelas existente. Exemplos práticos para este problema são: copiar um conjunto de arquivos para um disquete, colocar caixas na carroceria de um caminhão e decidir ordens de comerciais em uma determinada emissora de televisão [15]. Nos problemas de corte, um conjunto de peças deve ser cortado (retirado) de um conjunto de chapas de um determinado material, de forma que o maior conjunto possível de peças

possa ser retirado (ou a menor quantidade possível da chapa seja desperdiçada). Este é um problema considerado NP-Árduo.

O objetivo desta abordagem é otimizar um "plano de corte" bi-dimensional guilhotinado, que é "determinar as posições dos cortes de forma que um determinado conjunto de peças do pedido pode ser retirado da chapa" [41]. Esta é uma versão mais restrita do corte bi-dimensional, e pode ser caracterizada da seguinte maneira:

Sejam L e H a largura e a altura da chapa, respectivamente; seja também p_1, ..., p_m uma lista de m pedidos de peças de dimensões ($l_i \times h_i$) e em quantidade q_i, a serem cortadas sobre as chapas, com $l_i \leq L$, $h_i \leq H$ e $q_i > 0$, para $i = 1, ..., m$ [1]. Inicialmente, o único pedaço disponível para corte dos pedidos é a própria chapa (chapas adicionais podem ser utilizadas caso a inicial não seja suficiente). "Cortar o pedaço em uma posição o com uma certa orientação (vertical ou horizontal)" significa dividir o pedaço em dois menores, traçando-se uma linha de corte na posição o desejada e com a orientação definida [41].

Após a construção do *plano de corte* ainda podem existir peças cujos pedidos não foram atendidos. Assim sendo, novas chapas podem ser obtidas e cortadas até que todos os pedidos sejam atendidos. A solução do problema é um conjunto de *layouts* de corte tal que aconteça o menor desperdício de material, utilizando-se a menor quantidade de chapas possível.

Uma versão da ferramenta de corte utilizada para este problema foi escrita por Aloise *et. al.* [1] com o objetivo de ser utilizada para teste de uma heurística desenvolvida [44]. A ferramenta chamava-se "Corte de Vidro Automatizado" e sua idéia foi aperfeiçoada por Moreira [41], que incorporou os conceitos de interação homem-máquina propostos pelo *framework User Hints*, tornando-a uma estrutura totalmente adaptada à Otimização Colaborativa.

A seguir, serão mostrados as três formas básicas de interação com o usuário que esta abordagem permite:

- **Re-execução –** dada uma seleção de peças, pode-se re-executar o algoritmo de forma que esta seleção seja considerada como um novo conjunto de entrada. Podem-se selecionar peças individuais através de clique do mouse. Selecionam-se conjuntos de peças através de áreas de seleção;

- **Restrições –** o algoritmo pode considerar restrições impostas pelo usuário, as quais devem ser obedecidas na hora da execução, como por exemplo, a posição de determinadas peças (acima, abaixo, esquerda e direita) em relação a uma outra; o agrupamento de determinadas peças, de forma que elas apareçam sempre agrupadas; e a atribuição hierárquica à uma determinada peça, fazendo com que esta esteja sempre em um nível mais alto, de forma a permitir a sua retirada imediata.
- **Alteração Manual –** Através de operações de pegar, arrastar e soltar, o usuário pode movimentar peças de um lugar para outro em uma chapa. Esta interação aproveita a capacidade do usuário de perceber melhores encaixes de uma peça no *layout*, fazendo melhor aproveitamento no espaço da chapa.

A Figura 2.13 [41] mostra o *framework User Hints* para a Otimização Interativa de Planos de Corte.

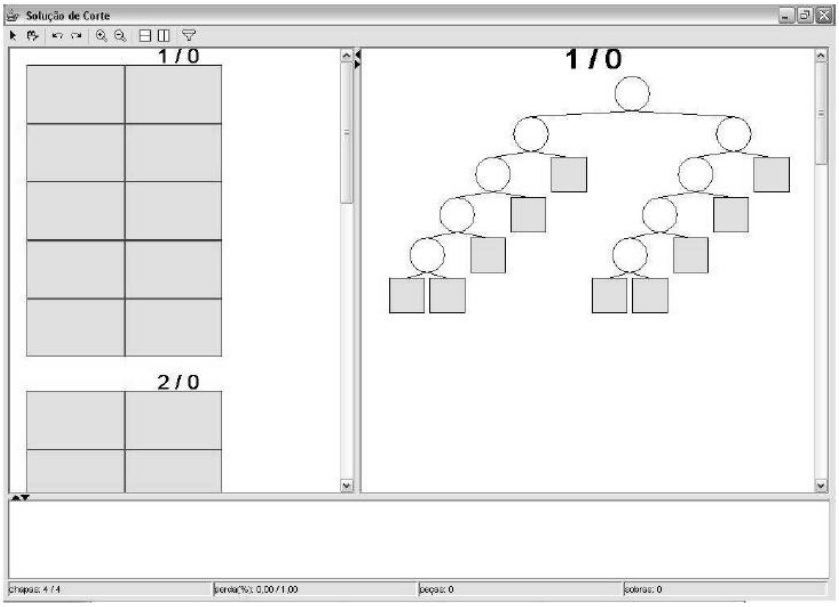

Figura 2.13 – *framework User Hints* para a Otimização Interativa de Planos de Corte.

3 – *Bin Packing 3D*

3.1 – *User Hints* para o Problema *Bin Packing 3D*

3.1.1 – Modelo

O sistema de interação para o problema *Bin Packing 3D*, chamado "BP3D Interativo", foi construído sobre uma base já existente de um programa de empacotamento tri-dimensional desenvolvido por Baltacioglu [9], mais a heurística chamada "Heurística de Suavização de Superfícies Irregulares", desenvolvida por Gonzaga e Bittencourt [3].

Baltacioglu [9] desenvolveu o programa baseado no problema do distribuidor, ou seja, onde se trabalha com diferentes tipos de caixas no que diz respeito a sua altura, largura e comprimento. A abordagem utilizava um algoritmo que empacotava todos os itens formando camadas ou paredes, dependendo da orientação do contêiner. O programa se limitava a empacotar apenas um container, deixando os itens restantes fora. O código-fonte referente a esta abordagem pode ser encontrado em [9].

A "Heurística de Suavização de Superfícies Irregulares" aprimorou a idéia inicial para que camadas e subcamadas geradas fossem "suavizadas" no decorrer do processo. O termo "suavizar" é usado para caracterizar o comportamento que a heurística toma de empacotar a camada atual com itens que possuam uma mesma altura, de forma que a superfície desta camada fique plana e sem "buracos" (espaços vazios). Além disto, a heurística incorpora novas restrições, que são o "tombo" e a "sobreposição", que, respectivamente, significam que um item "pode, ou não, ter suas seis orientações analisadas" e "pode, ou não, ter outros itens acima de sua altura". Essas novas restrições já podem ser caracterizadas como uma melhora na modelagem do problema, tornando o processo mais próximo da realidade.

O que o "BP3D Interativo" acrescenta à resolução do problema é a possibilidade do usuário interagir diretamente com a resposta fornecida, utilizando o seu raciocínio lógico e estabelecendo critérios para um possível melhoramento da solução obtida. O

programa utiliza conceitos de interação homem-máquina existentes na abordagem *User Hints*, que é um *framework* genérico para adição de funcionalidades interativas, em sistemas de Otimização Colaborativa, no intuito de melhorar os processos colaborativos Homem-Computador.

O "BP3D Interativo" obedece aos dois critérios principais das interações com o usuário em processos de Otimização, que são:

- Refinar o problema, inserindo conhecimento de domínio;
- Ajudar a convergir para soluções ótimas.

Foi utilizada a linguagem *C++* juntamente com as bibliotecas gráficas *OpenGL* [59] e *Qt*, além da ferramenta *Qt Designer* [67], para o desenvolvimento da interface gráfica. O sistema conta com as seguintes principais funcionalidades:

- Alteração da disposição das caixas, assim como sua remoção ou adição em um novo contêiner;
- Refinamento da solução dada pelo método de Otimização empregado. Esse refinamento pode ser feito em um subconjunto de caixas selecionadas, ou no empacotamento como um todo, dependendo dos critérios empregados pelo usuário;
- Visualização em tempo real do refinamento da solução, além dos dados referentes a cada contêiner e caixa.

Estas facilidades foram implementadas neste sistema como um processo cíclico onde uma solução inicial de empacotamento é dada e o usuário pode alterá-la para atingir uma melhoria, podendo voltar a empregar o mesmo método de Otimização da resposta original no seu refinamento (o processo cíclico é mostrado na Figura 3.1 abaixo).

3.1.2 – Visualização

Inicialmente o programa recebe um arquivo de texto contendo todos os dados e restrições sobre uma determinada instância do problema. A esse arquivo, demos a extensão de ".bp" (referente a *"Bin Packing"*).

O arquivo contém as dimensões do tipo de contêiner a ser utilizado e os dados referentes a cada caixa a ser usada no empacotamento, que são: índice do tipo de caixa, altura, largura, comprimento, quantidade do tipo de caixa, informação sobre se a caixa pode tombar, e informação sobre se ela pode ser sobreposta (veja Figura 3.1a).

Ao receber o arquivo, o programa aplica a "Heurística de Suavização de Superfícies Irregulares" e executa o processo de empacotamento. Após o processamento dos dados, o programa irá retornar um novo arquivo que contém todos os dados originais sobre as caixas, acrescidos da sua localização e orientação dentro do contêiner. Essa resposta serve como entrada para o sistema e recebe a extensão ".vbp" (referente a *"Visual Bin Packing"*), que vai ser utilizado pelo visualizador do BP3D Interativo (veja figura 3.1b).

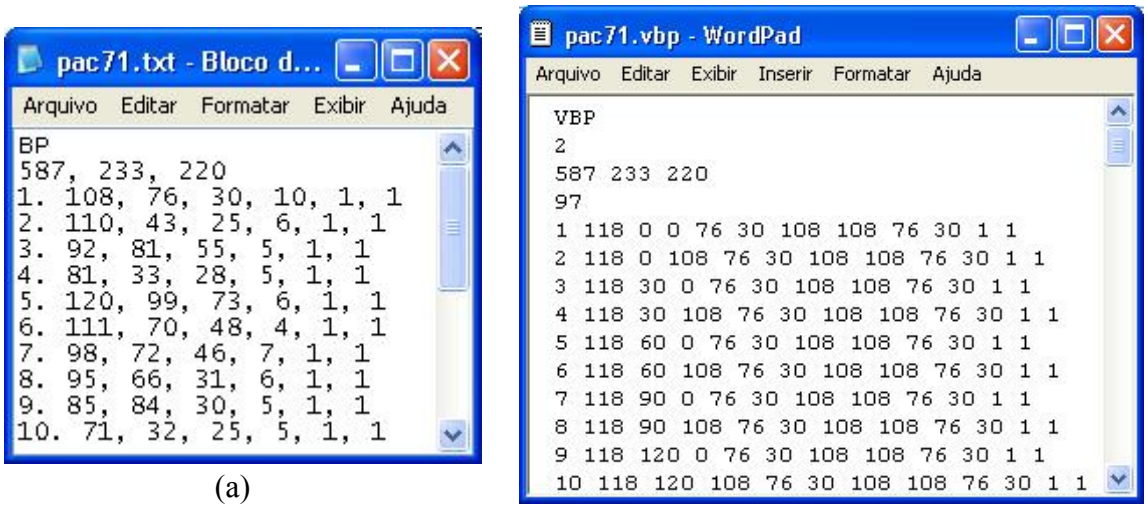

Figura 3.1 – Arquivo de entrada de dados para o empacotamento (a) e para o visualizador (b).

A partir destes dados, o BP3D Interativo consegue reproduzir um modelo tri-dimensional do contêiner e das caixas dentro dele. Para as caixas que não puderem ser empacotadas no contêiner principal, é criado um (ou mais) contêiner(es) secundário(s) para que elas sejam depositadas dentro.

O usuário agora tem total liberdade de: modificar as disposições destas caixas; remover uma, ou várias, caixas que estão em um contêiner; adicionar novas caixas; entre outras funções que serão apresentadas na próxima subseção.

3.1.3 – Tipos de Dicas

O sistema permite ao usuário vários tipo de iterações, todas estas fundamentadas no que foi visto abordado sobre o *framework User Hints*. Na figura 3.2 é mostrado *framework* desenvolvido para o BP3D.

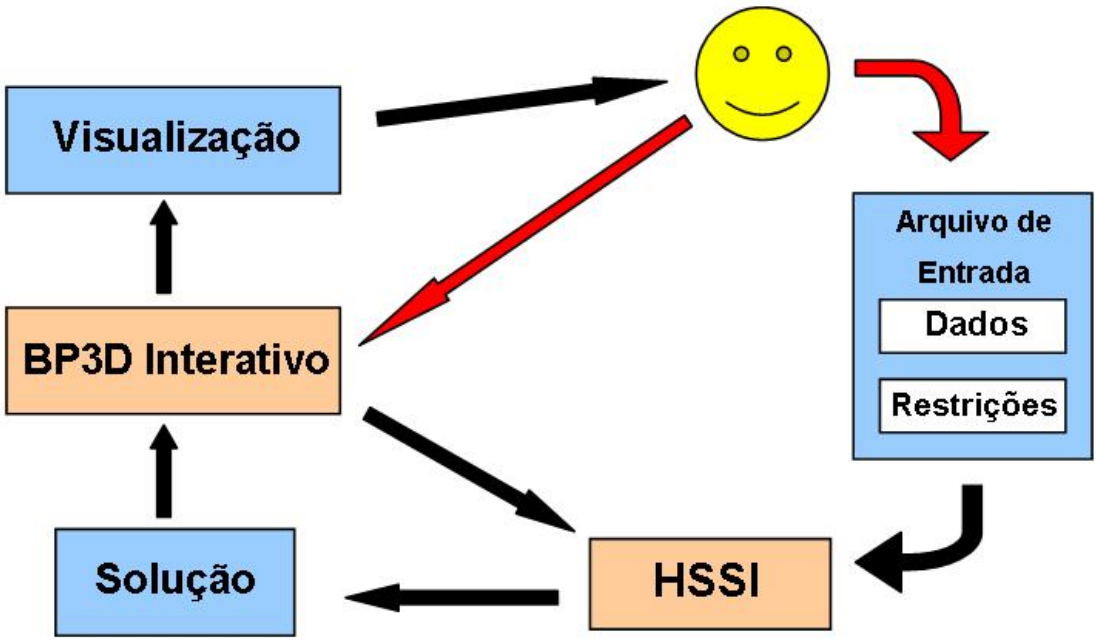

Figura 3.2 – *framework* para o BP3D Inteativo.

Fazendo uma comparação com a figura 2.10 [44], referente ao *framework User Hints*, pode-se reparar que o BP3D Interativo possui uma grande compatibilidade.

Abaixo, estão classificadas todas as funcionalidades do BP3D Interativo:

a) Quanto à aparência e visualização dos dados:

- **Mudança de visualização de contêiner e de caixa –** é permitido o modo sólido (um composto de seis retângulos) ou *wireframe* (estrutura de objeto composta somente por linhas). Além disso, pode-se modificar a iluminação, aplicar textura e, somente para caixas, bordas;
- **Visualizar as informações –** são visualizadas, e constantemente atualizadas, as informações sobre o empacotamento, que correspondem às de cada caixa e contêiner.

b) Quanto seleção de caixa ou de área (focalização):

- **Selecionar caixa –** pode-se selecionar uma caixa por vez ou várias (caso o modo "exclusivo" de seleção esteja desativado);
- **Selecionar área –** o usuário pode determinar um volume, o qual será definido por suas dimensões e origem; ou selecionar várias caixas de uma só vez, delimitando uma área com o mouse.

c) Quanto à modificações aplicadas às caixas:

- **Alterar as dimensões** – pode-se alterar os dados referentes as dimensões da caixa (altura, largura, comprimento, posição e restrições quanto ao tombo e sobreposição) através de entradas feitas diretamente no programa;
- **Permissão de saída e interseção –** pode-se permitir a saída de caixas do contêiner ou que estas façam intersecção com outras;
- **Alinhar** – alinha uma caixa, ou várias, de uma só vez em uma determinada direção;
- **Deslocar** – pode-se deslocar uma caixa por vez em uma determinada velocidade.

d) Quanto à câmera:

- **Alterar posição –** altera a posição da câmera;
- **Alterar ângulo de visão** – o ângulo de visão da câmera pode ser mudado.

e) Quanto à edição dos dados:

- **Copiar, recortar e colar –** pode-se copiar ou recortar uma caixa para que esta seja adicionada em outra localidade;
- **Criar novas caixas –** pode-se criar uma caixa, definindo as suas dimensões e posição onde será adicionada;
- **Re-empacotar –** é a funcionalidade "alvo" desta abordagem. Após uma seleção de um volume, o usuário, através do menu "contêiner", pode

chamar novamente o algoritmo de empacotamento para tentar melhorar o espaço deste volume;

- **Buscar –** procura caixas, bastando fornecer o seu índice (o mesmo que está definido no arquivo de entrada ".bp");
- **Retroceder operações** – uma operação mal sucedida pode ser desfeita. Uma operação desfeita pode, também, ser refeita;
- **Salvar –** pode-se salvar os progressos obtidos em uma execução do programa em um arquivo do tipo ".bp" ou ".vbp";

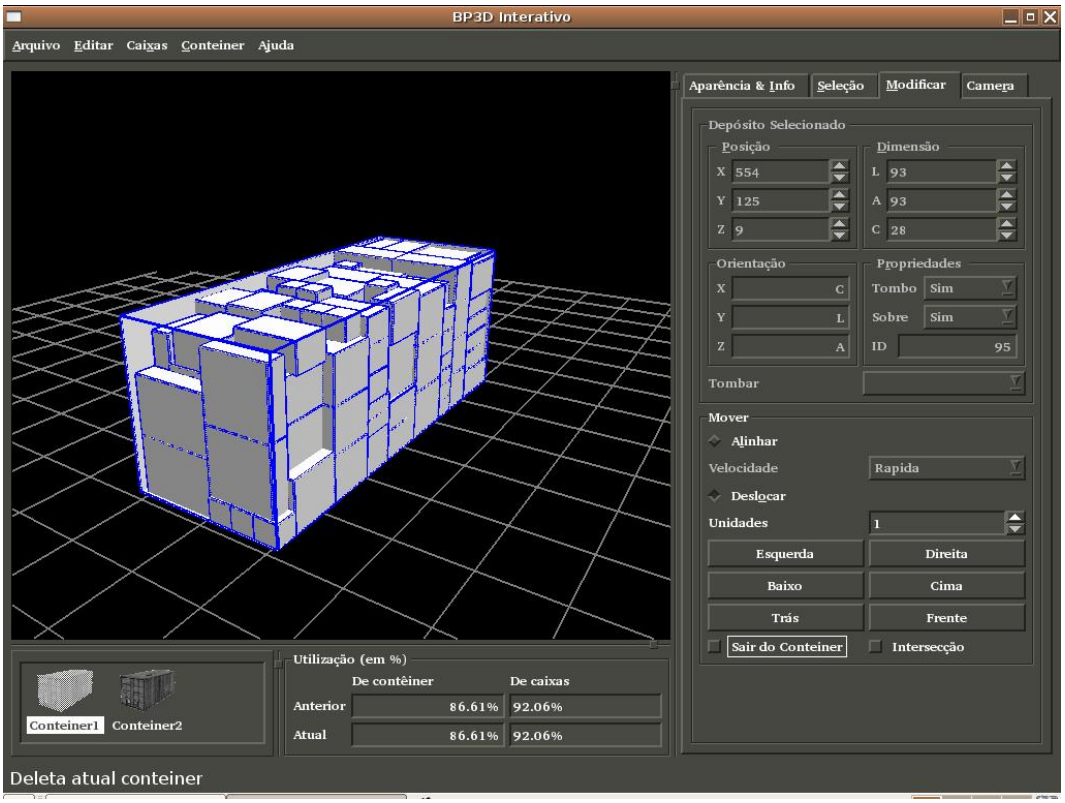

Figura 3.3 – framework User Hints para Empacotamento Tri-Dimensional (BP3D Interativo).

3.2 – O Sistema Implementado

OpenGL é definida como "um software de interface para hardware gráficos" [49]. Na verdade, *OpenGL* é um conjunto de bibliotecas de comandos gráficos ou API (*Application Programming Interface* – Interface de Aplicação de Programação) para desenvolvimento de aplicações gráficas e de modelagem bi (2D) e tri-dimensional (3D). Ela foi introduzida pela *Silicon Graphics, Inc.* (*SGI*), líder mundial em Computação

Gráfica, em 1992, no intuito de dar suporte a várias plataformas de hardware, assim como diversos sistemas operacionais.

As bibliotecas *OpenGL* têm como maior vantagem a sua utilização e rapidez, isto devido aos algoritmos cuidadosamente desenvolvidos pela *SGI*. A maioria das suas rotinas são implementadas na linguagem C, tornando mais fácil a sua utilização em qualquer programa escrito em C ou C++. O acesso a suas bibliotecas é semelhante ao uso das elaboradas para a linguagem C.

As aplicações *OpenGL* são diversas, como por exemplo, ferramentas CAD e CAM, modelagem, animação, jogos, desenvolvimento de ferramentas de realidade virtual e simuladores, análise de dados e mapeamento geográfico, etc [50], além de suas primitivas gráficas. Geralmente dizemos que um programa é baseado em *OpenGL*, ou é uma aplicação *OpenGL*, o que significa que ele é escrito em alguma linguagem de programação que faz chamadas a uma ou mais bibliotecas *OpenGL* [38].

Uma aplicação feita em *OpenGL* conta com diversos recursos, como suporte a iluminação, coloração, mapeamento de textura, transparência, entre muitos outros efeitos. Ao invés de descrevermos uma cena e como ela deve ser renderizada, quando se usa *OpenGL* é preciso apenas determinar os passos, ou seja, as chamadas a API, necessários para alcançar a aparência ou efeito desejado.

As implementações do *OpenGL* geralmente provêem de bibliotecas auxiliares, tais como a GLU (*OpenGL Utility Library* – Biblioteca de Utilidades *OpenGL*), utilizada para realizar tarefas comuns, tais como manipulação de matrizes, geração de superfícies e construção de objetos por composição [70], e a GLUT (*OpenGL Toolkit*) para gerenciamento de janelas.

O *OpenGL* utiliza o que chamamos de "processo em *pipeline*". Esse tipo de processo pode ter dois ou mais passos, dependendo da aplicação. Cada passo corresponde a um estágio de transformação que começa com uma chamada às funções API *OpenGL*, e em seguida os comandos são colocados em um *buffer* de comandos. Este *buffer* é preenchido com comandos, vértices, dados de textura, etc. Quando este é "esvaziado", os comandos e dados são passados para o próximo estágio, que é o de transformações geométricas e iluminação. As transformações geométricas correspondem à manipulação de um modelo, ou seja, a alteração de sua escala, rotação ou posição. Já a iluminação de um objeto é caracterizada pela quantidade de RGB (*red, green and blue* - vermelho, verde e azul) que uma fonte de luz emite e pela percentagem dos mesmos que chegam ao objeto e são refletidos em várias direções. O

terceiro passo é o de "Rasterização" (*Raster* - desenho de linhas, antes de surgir a imagem), que é onde se é gerada a imagem a partir dos dados geométricos de cor e de textura. Por último, a imagem é colocada no *frame buffer*, que é a memória do dispositivo gráfico. Isto significa que a imagem é exibida no monitor [71]. O modelo *pipeline* pode ser observado na figura abaixo:

Figura 4.1 – Modelo do processo em *pipeline*.

Devido ao fato de poderem ser portáteis, programas *OpenGL* não possuem funções para gerenciamento de janelas, iteração com o usuário ou arquivos de entrada/saída (isto é função, por exemplo, da GLUT), foi utilizado neste trabalho o recurso de uma outra biblioteca gráfica chamada *Qt*, e de uma ferramenta chamada *Qt Designer*, ambas da empresa de software *Trolltech*. Esta biblioteca é, ao mesmo tempo, uma classe de biblioteca C/C++ e um kit de ferramentas GUI (*Graphic User Interface* – Interface Gráfica do Usuário) [66].

O kit de ferramentas *Qt* para *C++* tem sido utilizado amplamente para aplicações comerciais, desde 1995, em companhias diversas, como a Adobe, IBM, Motorola, NASA, Volvo, e por numerosas companhias e organizações menores [66].

Uma das vantagens de escolher o *Qt* e o *Qt* Designer para se trabalhar é que ambas são gratuitas (livres para Unix e disponível em versão não-comercial para Windows) e multiplataforma, ou seja, estão disponíveis em vários tipos de plataformas diferentes. Sendo assim, todos os componentes do programa podem ser compilados nos diversos ambientes Unix, Windows e Macintosh, bastando para isto possuir a versão do *Qt* para cada sistema operacional e um compilador *C++*.

Qt é muito mais do que um kit de ferramentas de bibliotecas gráficas. Ela inclui um conjunto vasto de controles que fornecem uma funcionalidade GUI (*Graphic User Interface* – Interface Gráfica de Usuário) padrão. Além das características gráficas básicas, *Qt* suporta a utilização de bases de dados, arquivos, sistemas de arquivos e "sockets", tudo isso de uma forma independente de plataforma [17]. Ela é escrita em *C++*, sendo completamente orientado a objetos é, também, dirigida a eventos *("event driven")*. *Qt* estende a idéia original de objetos, onde existem métodos e atributos,

incrementando esse conceito com alternativas inovadoras para a comunicação entre objetos, chamada *"Signals & Slots"*. *Signals* são funções protótipo que podem ser utilizadas para emitir sinais, e *Slots*, são funções simples que são usadas para capturar os sinais e executar uma determinada ação.

Outras funcionalidades do *Qt* são o fornecimento a modelos de eventos convencionais para cliques de mouse, pressionamento de botões, etc; o uso de funcionalidades de interface de usuário requisitadas por aplicações modernas, como menus, menus de contexto, arrastar e soltar, etc; e também a utilização de um compilador chamado "moc" (*meta object compiler*) que cria meta objetos, ou seja, objetos que descrevem outros objetos. Este compilador permite a criação de componentes para a reutilização.

3.2.1 – Principais Métodos do Sistema

O código da parte visual, escrito em *OpenGL*, conta com dezenas de métodos que estão agrupadas em uma única classe chamada **GLBox**. Esta classe guarda todas as características de cada caixa, do contêiner, modos de seleção, visualização, etc. O *Qt* exige que se tenha pelo menos uma classe responsável pela parte *OpenGL*. No caso, **GLBox**, é considerada pelo *Qt* como uma classe derivada da classe *QGLWidget*, a qual é usada para desenhar cenas *OpenGL* na tela.

Como são muitos os métodos, podemos listar alguns principais desta classe **GLBox**, agrupando-os em:

Métodos de desenho

O método **drawScene( )** é o responsável por chamar os outros métodos de desenho entre os existentes, esperando um pedido de cada um para ser ativado.

O método **drawBoxes( )** é responsável por renderizar (desenhar) todas as caixas no contêiner atual, com uma determinada cor, textura, iluminação. Ele recebe como parâmetro um inteiro que define se irá ocorrer a renderização de toda a área, com caixas, grade, contêiner, etc., ou se apenas irá ocorrer uma seleção de área que não será mostrado.

O método também verifica colisões de caixas através de um outro método chamado **is_outside( )**, que verifica se uma caixa está "fora" de outra. Caso não, **drawBoxes( )** fará com que as caixas que estão colidindo apareçam de cor verde na tela.

O método **drawPlanes( )** desenha os planos que formam o contêiner. Ele não mostra todos os seis planos do contêiner, pois exclui os que se localizam na frente das caixas.

O método **drawGrid( )** desenha a grade que serve como base de onde irá aparecer a visualização do empacotamento.

Por último, temos o método **drawAxis( )**, que desenha os eixos X, Y e Z do espaço.

Métodos de arquivo

Para abrir arquivos, o programa dispõe de dois métodos, o **openBPFile( )**, responsável por abrir arquivos com extensão .bp, e o **openVBPFile( )**, responsável pelos .vbp.

O processo que se realiza ao abrir um arquivo do tipo .bp é o seguinte: o método **openBPFile( )** recebe o nome do arquivo e verifica se seu formato é compatível. Sendo o formato compatível, ele chama o programa de empacotamento, o qual possui a heurística, que gera uma saída do tipo .vbp. Esta saída irá servir de entrada para o método **openVBPFile( )**.

Já processo que se realiza ao abrir um arquivo do tipo .vbp é o seguinte: o método **openVBPFile( )** verifica se o arquivo passado como parâmetro existe. Existindo, ele lê todos os dados no arquivo e os joga em uma estrutura **conteiner**, a qual irá armazenar todas as informações que estão no arquivo, criando o contêiner e as caixas que serão utilizadas.

Para salvar os arquivos, contamos também com dois métodos: **saveAsBP( )** e **saveAsVBP( )**. Ambas pegam todos os dados que estão na estrutura **conteiner** e os escrevem em um novo arquivo de texto, o qual irá receber a extensão .bp ou .vbp, para **saveAsBP( )** ou **saveAsVBP( )**, respectivamente.

Métodos para manipulação de caixas e contêineres

Para manipulação de caixas, contamos com os seguintes métodos:

- **addBox( )** – adiciona caixa ao contêiner atual. Ela cria uma nova estrutura **myBin**, que guarda os dados sobre uma caixa (posição, dimensões, tombo, sobreposição, etc.). Após criar a caixa, ela a insere dentro do contêiner na posição determinada pelo usuário (e que foi gravada dentro da nova estrutura).
- **deleteBoxes( )** – apaga todas as caixas selecionadas. As caixas, quando selecionadas, têm seus índices guardados em um vetor. Quando se recebe o comando de apagar caixas, o vetor é analisado e cada caixa, correspondente a um índice deste vetor, tem seus dados liberados pelo método **freeBox( )**. Ao fim, o volume do contêiner é atualizado.
- **copyBox( )** – copia os dados de uma caixa selecionada e os atribui à uma nova estrutura **myBin**.
- **cutBox( )** – recorta uma caixa selecionada para uma possível reposição posterior desta. Ele utiliza o método **deleteBoxes( )** para a retirada da caixa, porém guarda a informação desta.
- **pasteBox( )** – reposiciona uma determinada caixa que fora recortada ou copiada mais recentemente, utilizando o método **addBox( )**.

Para a manipulação de contêineres contamos com os seguintes métodos:

- **addConteiner( )** – adiciona um novo contêiner. Ele recebe 4 parâmetros, os quais os 3 primeiros são as dimensões do contêiner atual, e o último é um identificador de número do contêiner.
- **deleteConteiner( )** – o procedimento é o mesmo de apagar uma caixa, sendo que não existe um vetor, já que só podemos selecionar um contêiner por vez. Neste processo, utiliza-se um método chamado **freeConteiner( )**, que libera a memória de uma estrutura **conteiner**.

Método de re-empacotamento

O método responsável pelo re-empacotamento das caixas é o **pack( )**. O processo se dá conforme os passos abaixo:

1. Verifique se a opção de seleção por “Volume” está ativada. Caso não, apresente mensagem de erro;

2. Verifique se todas as caixas selecionadas estão dentro do volume de seleção (volume não ajustado). Caso não, apresente mensagem de erro;
3. Crie um arquivo temporário do tipo .bp e adicione o volume de seleção e os dados das caixas selecionadas. Feche o arquivo;
4. Re-execute **bp3d** com os dados do arquivo temporário e gere uma saída (nova solução) temporária do tipo .vbp. Fim da execução de **bp3d**;
5. Apague todas as caixas selecionadas no volume de seleção;
6. Abra o novo arquivo temporário .vbp, leia os dados e armazene em variáveis (dados referentes as caixas re-empacotadas);
7. Coloque as caixas re-empacotadas no contêiner.

Métodos para verificação

Os métodos de verificação existem para procurar irregularidades no empacotamento. Irregularidades estas que podem ser: flutuação, colisão ou sobreposição de caixas, e se existe caixa fora do contêiner.

Para verificar se existe flutuação de caixas, o programa conta com o método **verifyFlutuation( )**. Para cada caixa do contêiner, ele verifica se o seu ponto de origem toca o plano referente a base do contêiner, ou seja se a altura Y de sua origem é igual a 0 (zero), ou se toca a altura de uma outra caixa. Caso não ocorra pelo menos uma dessas duas situações, ele considera que a caixa está flutuando.

Para a verificação de colisão, temos o método **verifyColision( )**, que verifica se existe espaço ocupado em comum para duas ou mais caixas. Para cada caixa, verifica se existe uma das outras caixas dentro dela. Para isto, o método chama **is_outside( )**, que pega duas caixas e verifica se existe intersecção em todos os seus planos de projeção (intersecção espacial).

Para a verificação de sobreposição, usamos o método **verifyOn( )**. Ele verifica para cada caixa se existe uma caixa abaixo dela com proibição de sobreposição. Caso haja, considera-se que esta caixa está em posição irregular.

Por último, para verificar se existe caixa fora do contêiner, é utilizado o método **verifyOutConteiner( )**. Ele verifica se algum dos planos que formam a caixa está fora do contêiner, utilizando para isto o método **is_inside( )**, que observa se o conjunto de pontos (x_i, x_f), (y_i, y_f) e (z_i, z_f) (sendo i e f, início e fim, respectivamente) da caixa estão dentro do contêiner.

Método para ajuste de volume

O método **adjustVolume( )** ajusta o volume de seleção de forma que todas as caixas, ou estão totalmente dentro, ou totalmente fora, deste volume. Ele funciona da seguinte maneira: para todas as caixas, faz-se um teste com **is_outside( )** e **is_inside( )** para verificar se a caixa está nem totalmente fora, e nem totalmente dentro, respectivamente, do volume de seleção de forma que este englobe a caixa. Para aumentar este volume, é necessário saber por onde a caixa está saindo, ou seja, qual dos seus planos está fora do contêiner. Assim, faz-se com que o volume cresça até atingir este plano.

3.3 – Testes e Resultados Obtidos

Os testes aplicados ao sistema foram extraídos de 15 instâncias usadas por Gonzaga & Bittencourt [3] para o problema do distribuidor, mais 5 instâncias usadas por Baltacioglu [9] para o problema do produtor.

Os testes foram feitos em um processador Intel Pentium 4, 1.4 GHz e 128 Mb RAM, utilizando-se o sistema operacional Linux Ubuntu. A escolha do Linux para o desenvolvimento do sistema foi obvia: é gratuito. Além do mais, a atualização da ferramenta de design *Qt* é feita para Linux, gratuita, dispensando o uso de uma versão não comercial restrita, ou de uma versão comercial limitada e com prazo de expiração.

3.3.1 – Metodologia de Avaliação

A metodologia de avaliação empregada contou com algumas restrições, pois se trata de um sistema de Interação Humano-Computador.

A avaliação foi feita com 20 instâncias divididas em duas subclasses, que são referentes às duas subclasses do problema de empacotamento tri-dimensional: o problema do produtor (PP) e o problema do distribuidor (PD).

As instâncias PD, chamadas Pac7##, possuem dados de mais de três tipos de caixas. Já as do PP, chamadas Set_##, possuem, no máximo, dois tipos de caixa.

Foi estabelecido que, para cada uma das instâncias, independente da sua classe, seria dado o tempo máximo de 10 minutos para que o usuário fizesse as modificações e tentasse melhorar os resultados.

3.3.2 – Resultados Obtidos

Os resultados obtidos mostraram que existe uma diferença de dificuldade entre o problema do produtor e o do distribuidor, tendo o emprego da Heurística de Suavização de Superfícies Irregulares se mostrado mais satisfatório no segundo caso, já que a variedade de caixas é maior, fato que gerou mais espaços vazios para serem preenchidos com o uso da interação.

A tabela 3.1 abaixo compara o os resultados obtidos pela aplicação da HSSI e os resultados obtidos no *BP3D Interativo* com a ajuda do usuário. O valor em porcentagem, dado nas colunas Baltacioglu, HSSI e BP3D Interativo, se referem ao aproveitamento de cada contêiner. O valor, também em porcentagem, dado na coluna Ganho Aproximado, utiliza a seguinte fórmula: (BP3D Interativo)*100/HSSI – 100. Isso significa que o ganho é o percentual obtido pelo BP3D Interativo sobre a HSSI.

Instância	Baltacioglu	HSSI	BP3D Interativo	Ganho
Pac71	86,13 %	87,81%	88,01 %	0,23%
Pac72	86,10 %	86,08%	87,89 %	2,10%
Pac73	86,89 %	87,93%	87,93 %	0,00%
Pac74	86,72 %	89,96%	89,96%	0,00%
Pac75	86,61 %	88,83%	90,12%	1,45%
Pac76	88,36 %	88,93%	89,16%	0,26%
Pac77	86,73 %	88,71%	88,99%	0,32%
Pac78	100,00%	100,00 %	100,00 %	0,00%
Pac79	86,25 %	87,05%	**89,55%**	**2,87%**
Pac710	86,93 %	86,54%	**88,96 %**	**2,80%**
Pac711	87,95%	87,95%	88,39%	0,50%
Pac712	86,73%	87,62%	88,37%	0,86%
Pac713	87,75%	88,08%	88,56%	0,54%
Pac714	88,19%	88,76%	88,76%	0,00%

Pac715	85,50%	86,63%	88,23%	1,85%
Set_12	100,00%	100,00%	100,00%	0,00%
Set_13	100,00%	100,00%	100,00%	0,00%
Set_14	100,00%	100,00%	100,00%	0,00%
Set_15	100,00%	100,00%	100,00%	0,00%
Set_16	98,09%	98,09%	**98,44%**	**0,36%**

Tabela 3.1 – Resultados obtidos pelo BP3D Interativo.

Os testes delimitados pelo tempo mostraram um ganho máximo de 2,87% para o caso do distribuidor (as Figuras 3.1a e 3.1b uma *screenshot* de antes e depois de um ganho obtido). Já para o caso do produtor, apenas em uma instância (Set_16) houve ganho de 0,36%. A justificativa para isto se deve ao fato de que, no problema do produtor, o número de caixas iguais é muito maior, o que proporciona um melhor cálculo de aproveitamento do espaço do contêiner no momento em que se executa o programa de empacotamento. A Figura 3.2 mostra um empacotamento feito para este caso.

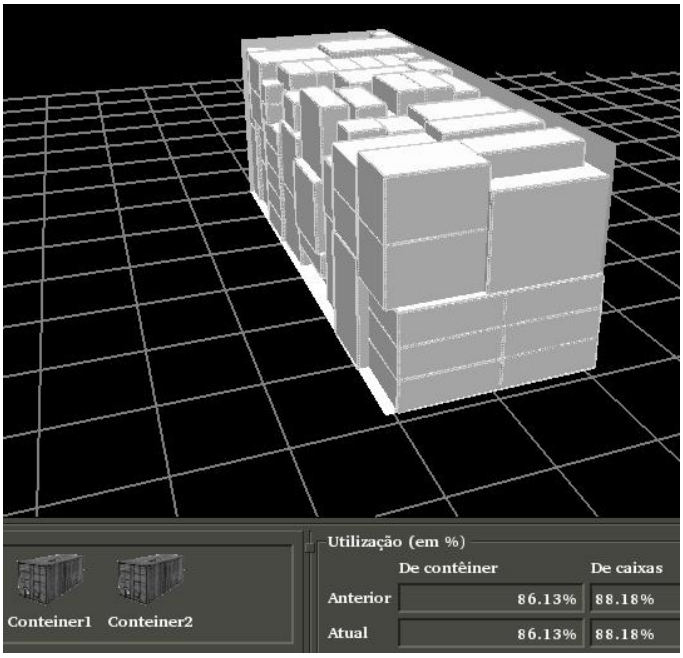

Figura 3.1a – Empacotamento antes da interação com o usuário (86,13% de aproveitamento).

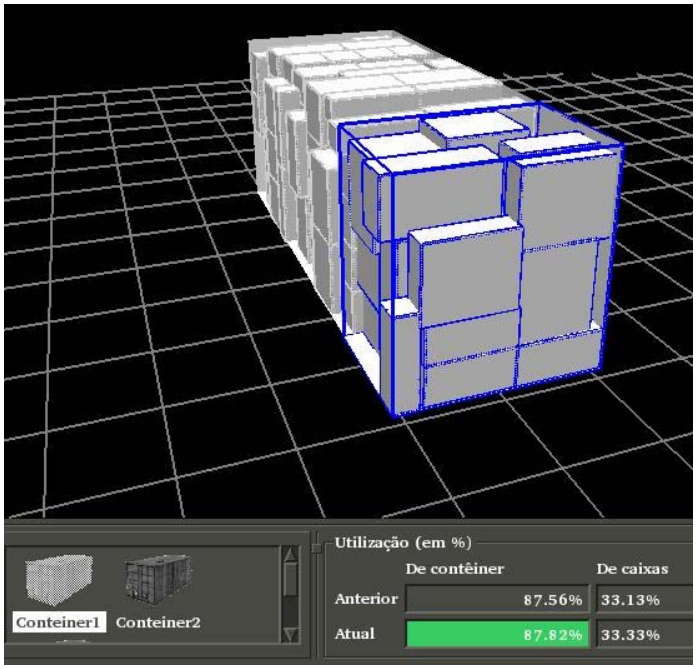

Figura 3.1b – Empacotamento depois da interação com o usuário (87,82% de aproveitamento).

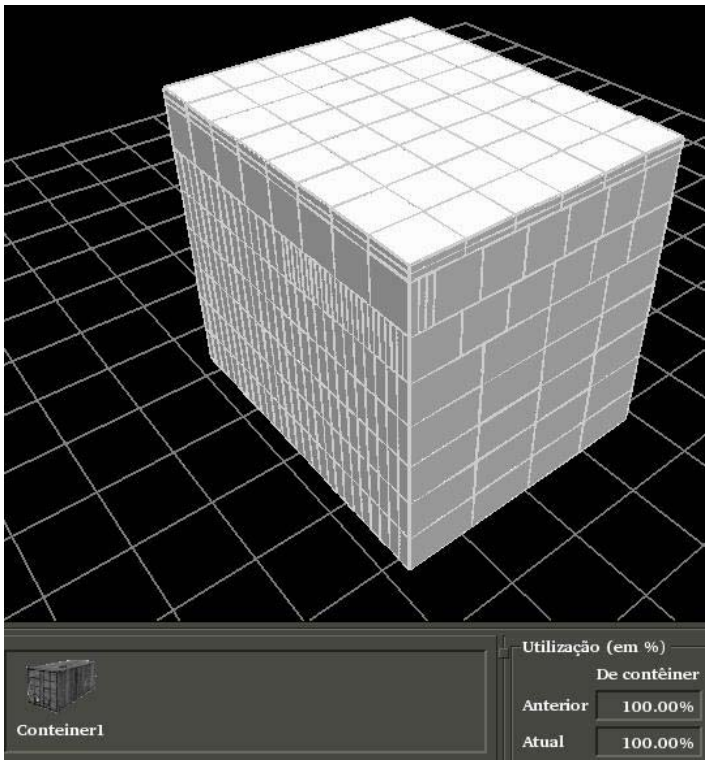

Figura 3.2 – Empacotamento para o problema do produtor (100% de aproveitamento).

Apesar de a melhoria apresentada ser pequena, a abordagem mostrou ganho. Isso mostra que o senso crítico humano ainda se sobressai quando comparado a uma abordagem inteiramente automática.

4 – Conclusão e Trabalhos Futuros

Neste capítulo serão listadas algumas possibilidades de expansão para o *BP3D Interativo,* assim como as considerações finais.

4.1 – Conclusão

A ferramenta de Otimização interativa *BP3D Interativo* mostrou-se satisfatória, pois foi capaz de mostrar a consistência de um processo de Otimização combinatória, unindo-a com a flexibilidade da iteração com usuário.

Os resultados deste trabalho apresentaram uma nova alternativa para resolução do problema de empacotamento tri-dimensional, proporcionando uma contribuição a pesquisa científica na área.

4.2 – Trabalhos Futuros

O *BP3D Interativo* pode ser expandido em suas várias funções, além de poder incorporar novas outras, assim como novos algoritmos. Podemos listar algumas possibilidades de expansão:

Integração e aplicação de novos algoritmos

Novos algoritmos poderão ser integrados ao programa, dando ao usuário mais opções de processamento para a obtenção de uma solução final. Abaixo encontram-se descritas algumas maneiras:

- Escolha do algoritmo para o processamento – o usuário será capaz de escolher qual o algoritmo que deseja executar antes de abrir uma instância, ou seja, na primeira interação com o programa;
- Escolha do algoritmo para o re-processamento - também existe a possibilidade de, no decorrer da interação do usuário com o programa, este oferecer mais de um algoritmo de re-processamento. Este pode ser aplicado em uma solução por inteiro, ou em uma seleção de área desta. Em outras palavras, o usuário poderá trabalhar no melhoramento de uma

determinada solução seguindo *N* passos e, durante estes, utilizar as diferentes opções de re-processamento integradas ao programa.

Expansão da área de visualização

A área de visualização do *BP3D Interativo* pode ser expandida com outras funcionalidades, como por exemplo:

- Visualização múltipla simultânea - por proporcionar apenas um tipo de visualização padrão por vez (acima, abaixo, esquerda, direita, frente, trás, diagonal e câmera livre), o *BP3D Interativo* limita um pouco a percepção do usuário na hora de verificar regiões mal aproveitadas no empacotamento. Uma expansão desta área para visualizações múltiplas ao mesmo tempo poderá ajudar neste processo. Como por exemplo, o usuário pode escolher ter quatro visualizações simultâneas na tela do monitor, sendo estas "à direita", "à esquerda", "acima" e "câmera livre", ou outras combinações quaisquer;
- Projeção de áreas mal aproveitadas – para facilitar ainda mais ao usuário achar áreas mal aproveitadas no contêiner pode-se incluir uma projeção dessas áreas na visualização atual ou e uma outra específica para este fim. Na visualização atual, isto poderia ser aplicado de duas formas: ver os itens em *wireframes* e os espaços desperdiçados solidificados; ou ver uma projeção "fantasma" dos espaços desperdiçados acima (ou em outra posição próxima) ao contêiner. A outra forma específica comentada seria acrescentar a opção de visualização para esta finalidade, a qual dividiria o espaço da tela com outros ângulos de visão que foi citado no item anterior.

Adição de novas interações

Pode-se listar várias novas formas de interação, como por exemplo:

- Recortar ou copiar vários itens de uma só vez, poupando o trabalho manual do usuário.
- Utilizar re-empacotamentos simultâneos de dois ou mais contêineres, possibilitando a permutação de itens entre contêineres. A vantagem disto seria, por exemplo, minimizar as perdas obtidas por dois ou mais

volumes (contêineres) que se deseja transportar ao mesmo tempo, levando-se em consideração que os dois teriam o mesmo destino.

- Adicionar prioridades aos itens, de forma que o programa leve em consideração certos itens que não podem ficar de fora do empacotamento.

Banco de dados de solução

O *framework User Hints* sugere que se armazene a solução que se está trabalhando em um banco e que, associado a esse, também exista um *agente* de melhor solução, que verifica quando a solução de trabalho, ou a do algoritmo, se torna melhor do que a definida como melhor até o momento, notificando o usuário da mudança.

Apesar de o *BP3D Interativo* avisar quando houve melhora, ele não guarda o resultado. Apenas é feita uma comparação com o resultado anterior.

Adaptação para outros problemas de empacotamento

Apesar de o *BP3D Interativo* ter sido projetado para o caso tri-dimensional de empacotamento, podemos trabalhar com os casos uni e bi-dimensional. Porém, para que isto seja possível, se é necessária à adaptação das instâncias para os padrões do programa. Como por exemplo, para se processar um empacotamento bi-dimensional com, digamos, 100 itens, teríamos que acrescentar mais uma dimensão para cada item do arquivo de entrada. Embora essa dimensão tenha um valor simbólico (por exemplo, 1), seria um desperdício de tempo para o usuário fazer esta adaptação.

O objetivo aqui será “adaptar” o *BP3D Interativo* para cada caso. Não só se tratando de dimensões do problema, mas também de restrições a ele impostas.

Referências Bibliográficas

[01] ALOISE, D. J., NASCIMENTO, H. A. D.; LONGO, J. H. "Uma heurística O(mn) para o corte bi-dimensional guilhotinado". Anais do XXXI Congresso da Sociedade Brasileira de Pesquisa Operacional, 1999.

[02] ALOISE, D. J., *et al*. "Otimização do Emprego da Unidade Móvel de Pistoneio através de GRASP". XXXII Simpósio Brasileiro de Pesquisa Operacional. Viçosa, MG, 2000.

[03] ALOISE, D. J.; GONZAGA, C. S; BITTENCOURT, V. G. Uma Heurística de Suavização de Superfícies Irregulares para a solução do Problema Bin Packing 3D. Artigo apresentado no XXXV SBPO. Natal-RN, Brasil, 2003.

[04] ALOISE, D. J.; NORONHA, T. F.; Silva, M. M. Um algoritmo memético de grupamento para o problema de Bin Packing 1-D. Artigo. Novembro de 2001.

[05] ANDERSON, D., *et al.* "Human-Guided Simple Search". National Conference on Artificial Intelligence (AAAI), 2000. ISBN 0-262-5112-6.

[06] ANGEL, E. "OpenGL: A Primer." Addison-Wesley. 2002.

[07] ATKINSON, M. "A greedy randomized search heuristic for time-constrained vehicle scheduling and the incorporation of a learning strategy". Journal of the Operational Research Society, 49:700–708, 1998.

[08] BÄCK, T. e SCHWEFEL, H.-P. "An overview of evolutionary algorithms for parameter otimization". Evolutionary Computation, 1(1):1-23. 1993.

[09] BALTACIOGLU, E., First Lieutenant, TUAF. "The Distributor's Three-Dimensional Pallet-Packing Problem: A Human Intelligence-Based Heuristic Approach". Air Force Institute of Technology. Wright-Patterson Air Force Base, Ohio – EUA. Tese de Doutorado. Março de 2001.

[10] BANON, G. J. F. "Bases da Computação Gráfica". Rio de Janeiro. Campus. 1989.

[11] BERNARDI, R. "Aplicando a técnica de times assíncronos na Otimização de problemas de empacotamento uni-dimensional". Dissertação de Mestrado. USP, São Paulo, 2001.

[12] BITTENCOURT, G. Inteligência Computacional. Disponível em: <http://www.das.ufsc.br/gia/softcomp/node15.html>. Acesso em: 01 ago. 2005.

[13] BOLLOB´AS, B. "Extremal Graph Theory", Academic Press, New York, 1978.

[14] CHVATAL, V., Linear programming, W. H. Freeman, New York, 1983.

[15] COFFMAN, E. G.; CSIRIK, J.; WOEGINGER, G. J. "Approximate solutions to bin packing problems". In Handbook of Applied Optimization. Oxford University Press, Inc. 1999.

[16] CONSTANTINO, A. A. Projeto e Análise de Algoritmos + Estrutura de Dados + Matemática = Otimização Combinatória. Disponível em: <http://www.din.uem.br/~ademir/Otimizacao.html>. Acesso em: 12 ago. 2003.

[17] COSTA, D. G. "Uma Aplicação de Áudio como Ferramenta de Suporte ao Ensino do Padrão ITU H.323". Universidade Federal do Rio Grande do Norte. Natal-RN. Outubro de 2004.

[18] DYCKHOFF, H. "A Typology of cutting and packing problems". European Journal of Operational Research. 44, 145-159, 1990.

[19] EILON, S.; CHIRISTOFIDES, N. "The Loading Problems", Man. Sci, 17, pp. 259-268, 1971.

[20] FILHO, L. C. T. C., Primeiro Tenente QEM; BRITO, J. L. N. S., Coronel QEM R/1. "Projeto e-foto: Uma Estação Fotográfica Digital Educacional". XI Simpósio Brasileiro de Sensoriamento Remoto. Belo Horizonte, Brasil. Abril de 2003.

[21] FJALLSTROM, P.-O., "Algorithms for graph partitioning: A survey". Linköping Electronic Articles in Computer and Information Science, 1998.

[22] FOGEL, D. B. "Evolving artificial intelligence". Ph. D. Thesis, University of California, San Diego, CA. 1992.

[23] FOGEL, L. J.; OWENS, A. J. e WALSH, M. J. "Artificial Intelligence Through Simulated Evolution". John Wiley and Sons, NY, 1966.

[24] GAREY, M.; JOHNSON, D.. Computer and Intractibility: A guide to the Theory of NPCompleteness, Freeman, San Francisco, 1979.

[25] GAREY, M.; JOHNSON, D.; STOCKMEYER, L. "Some simplified NP-complete graph problems". Theoretical Computer Science, Vol. 1, 1976, pp. 237–267.

[26] GILMORE, P. C.; GOMORY, R. E. "A linear programming approach to the cutting-stock problem", Opera. Res., Hamps, v. 9, p. 848-859, 1961.

[27] GLOVER, F. “Scatter search and star-paths: Beyond the genetic metaphor”. OR Spektrum, 17:125–137, 1995.

[28] GLOVER, F. “Tabu Search: Part I (1989) and Tabu Search: Part II (1990)”. ORSA Journal on Computing.

[29] HEITKOETTER, J.; BEASLEY, D. The Hitch-Hiker's Guide to Evolutionary Computation: A list of Frequently Asked Questions (FAQ). Disponível em: <http://www.faqs.org/faqs/ai-faq/genetic/part1/preamble.html>. Acesso em: 29 ago. 2005.

[30] HOLLAND, J. “Adaptation in nature and artificial systems”. University of Michigan Press, Ann Harbor, 1975.

[31] IMHOF, E., “Positioning names on maps,” Amer. Cartogr., Vol. 2, 1975, pp. 128–144.

[32] KATSURAYAMA, D. M. Página Pessoal. Disponível em: <http://www.lac.inpe.br/~massaru/pce.htm>. Acesso em: 23 jul. 2003.

[33] KIRKPATRICK, S.; GELLAT, D. C.; VECCHI, M. P. “Optimization by Simulated Annealing”. Science, v. 220, p. 671-680. 1983.

[34] KLAU, G. W. “A Combinatorial Approach to Orthogonal Placement Problems”. Ph.D. thesis, Fachbereich Informatik, Technische Fakultät I, Universität des Saarlandes, Saarbrücken, Germany, Sep. 2001.

[35] KLAU, G. W., et al.. "Human-Guided Tabu Search", National Conference on Artificial Intelligence (AAAI), ISBN: 0-262-51129-0, pp. 41-47, July 2002.

[36] KOZA, J. R. “Genetic Programming: on the programming of computers by means of natural selection”. Cambridge, MA: MIT, 1992.

[36] KOZA, J. R. “Genetic Programming: On the Programming of Computers by Means of Natural Selection”, MIT Press, 1992.

[37] LESH, N.; MARKS, J.; PATRIGNEME, M. “Interactive Partitioning”. International Symposium Graph Drawing, LNCS 1984, pp. 31-36. (2000).

[38] MANSSOUR, I. H. Introdução à OpenGL. Disponível em: <http://www.inf.pucrs.br/~manssour/OpenGL>. Acesso em: 14 maio 2005.

[39] MIRANDA, M. N. “Algoritmos Genéticos: Fundamentos e Aplicações”. http://www.gta.ufrj.br/~marcio/genetic.html acessado em 01/08/2005.

[40] MIYAZAWA, F. K., Algoritmos de Empacotamento Tri-dimensional – novas estratégias e análises de desempenho. Dissertação de Mestrado. USP, São Paulo, 1993.

[41] MOREIRA, R. A. "Otimização Interativa de Planos de Corte – Estrutura de Dados e Algoritmos". Trabalho de Conclusão de Curso. Goiânia, Goiás. 2004.

[42] MOURA, A. O que é Otimização Combinatória?. Disponível em: <http://goa.pos.dcc.unicamp.br/otimo/motivacao/main.html>. Acesso em: 21 fev. 2004.

[43] MÜLLER, F. M. Heurísticas e Meta-Heurísticas. In: V Escola Regional de Informática, 1997, Florianópolis e Maringá, 1997.

[44] NASCIMENTO, H. A. D., "User Hints for Optimization Processes", School of Information Technologies, The University of Sydney. Tese de Doutorado. Novembro de 2003.

[45] NASCIMENTO, H. A. D.; EADES, P. "User Hints for Map Labeling". XXVI Australasian Computer Science Conference (ACSC). Australian Computer Society, 2003. v. 16, p. 339-347.

[46] NEYER, G., "Map labeling with application to graph drawing," Drawing Graphs: Methods and Models, edited by D. Wagner and M. Kaufmann, Vol. 2025 of Lecture Notes in Computer Science, Springer-Verlag, 2001, pp. 247–273.

[47] NIELSEN, J. "Usability Engineering", AP Professional, Cambridge, 1993.

[48] NORONHA, T. F.; SILVA, M. M.; ALOISE, D. J. . "Uma Abordagem sobre Estratégias Metaheurísticas". Revista Eletrônica de Iniciação Científica (REIC), Ano I, Vol I, No. I, 2001.

[49] OPENGL ARCHITECTURE REVIEW BOARD, *et al*. OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.4, Fourth Edition (Paperback). Ed. Addison-Wesley Professional; 4a edition; 2003.

[50] OPENGL. OpenGL: The Industry Standard for High Performance Graphics. Disponível em: <http://www.opengl.org/applications/windows/>. Acesso em: 4 ago. 2005.

[51] OPTIMAL SPEKTRUM. Corte e Empacotamento & Correlatos. Disponível em: <http://www.mat.uel.br/spektrum/corte.htm>. Acesso em: 10 mar. 2004.

[52] PARKER, R. G.; RARDIN, R. L., Discrete optimization. Computer Science and Scientific Computing Academic Press, Inc., 1988

[53] PINHO, M. S. Tópicos Especiais em computação Gráfica. Disponível em: <http://www.inf.pucrs.br/~pinho/TACCII/>. Acesso em: 29 ago. 2005.

[54] PURCHASE, H., "Which aesthetic has the greatest effect on human undestanding?" Graph Drawing (Proc. GD '97), Vol. 1353 of Lecture Notes Comput. Sci., Springer-Verlag, 1997, pp. 248–261.

[55] RODRIGUES, E. "Programação Genética na Otimização de Circuitos Digitais". IV Encontro Nacional de Inteligência Artificial. Campinas, São Paulo, Brasil. 2003.

[56] ROGGIA, I. B.; MULLER, F. M.; Toscani, L. V. . "Método Heurístico Para Solução Do Problema De Sequenciamento Cíclico de n Tarefas em m Processadores Paralelos Idênticos". Artigo. UFRGS, 1998.

[57] SCHWEFEL, H.-P. "Evolution and Optimum Seeking". John Wiley & Sons Inc., New York, 1995.

[58] SCOTT, S. D.; LESH, N.; KLAU, G. W. . "Investigating Human-Computer Optimization", ACM Conference on Human Factors in Computing Systems (CHI), ISBN: 1-58113-453-3, pp. 155-162, April 2002.

[58] SCOTT, S., LESH, N.; KLAU, G. "Investigating Human-Computer Optimization". In Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI), ISBN: 158113-453-3, pp. 155-162. (2002).

[59] SGI. Sillicon Graphics, Inc. Disponível em: <http://www.sgi.com/>. Acesso em: 29 ago. 2005.

[60] SHNEIDERMAN, B. "Designing the User Interface: Strategies for Effective Human-Computer Interaction". Addison-Wesley Publishing, Reading, MA, 2nd ed., 1992.

[61] SHNEIDERMAN, B.; KANG, H. "Direct annotation: A drag-and-drop strategy for labeling photos," Proc. IEEE Int. Conf. on Information Visualisation (IV'00), edited by E. Banissi, M. Bannatyne, C. Chen, F. Khosrowshahi, M. Sarfraz, and A. Ursyn, London, 19–21 Jul. 2000, pp. 88–95.

[62] SILVA, J. L. C; SOMA, N. Y. "Um Algoritmo Polinomial para o Problema de Empacotamento de Contêineres com Estabilidade Estática da Carga". Instituto Tecnológico de Aeronáutica, São José dos Campos – SP. Artigo. 2002.

[63] SYSLO, M. M. "Disscrete Optimization Algorithms: With Pascal Programs". Prentice Hall (August 1, 1983).

[64] TATIBANA, C. Y.; KAETSU, D. Y. Homepage de Redes Neurais. Disponível em: <http://www.din.uem.br/ia/neurais/>. Acesso em: 10 abr. 2004.

[65] TICHY, W. F. “Should computer scientists experiment more?” IEEE Computer, Vol. 31, No. 5, May 1998, pp. 32–40.

[66] TROLLTECH. Qt 3.3 Whitepaper. Disponível em: <http://www.trolltech.com>,. Acesso em: 4 ago. 2005.

[67] TROLLTECH. Trolltech Software. Disponível em: <http://www.trolltech.com/>. Acesso em: 29 ago. 2005.

[68] VIANNA, A. C. G., “Problemas de Corte e Empacotamento: Uma Abordagem em Grafo”. XXXIII SBPO, São Paulo, 2001.

[69] WAKABATASHI, Y. “Otimização Combinatória”. http://www.ime.usp.br/mac/otimizacao.comb.html acessado em 12/08/2003.

[70] WANGENHEIM, A. Von. Tutorial de OpenGL. Disponível em: <http://www.inf.ufsc.br/~awangenh/CG/opengl.html>. Acesso em: 4 ago. 2005.

[71] WRIGHT, R. S. Jr.; SWEET, M. “OpenGL SuperBible”. 2nd ed. Indianapolis, Indiana: Waite Group Press, 2000.

[72] YOELI, P., “The logic of automated map lettering,” Cardographic J., Vol. 9, 1972, pp. 99–108.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo