MS 777 – **Projeto Supervisionado**

Abordagens Para o Problema de Corte - Enfoque para geração de *No-Fit Polygons*

Orientador:
Prof. Antônio Carlos Moretti

Aluna:
Letícia S. Artese
RA: 086477

Julho 2012

1-Introdução:

Este projeto apresenta uma síntese do artigo ***"Complete and Robust No-Fit Polygon Generation for the Irregular Stock Cutting Problem"*** **[1]** (Burke, Kendall,Whitwell e Hellier), que aborda um dos principais assuntos em pesquisa operacional, o problema de corte de formas irregulares que devem se "encaixar" de tal forma à minimizar o desperdício. Serão abordados vários casos, dado dois polígonos, estudaremos técnicas para garantir o posicionamento ótimo. E por fim apresentamos um problema básico de corte unidimensional, implementado em AIMMS, e uma breve discussão sobre problemas de corte bidimensionais e a obtenção dos padrões de corte.

A motivação desde projeto portanto é de fácil compreensão uma vez que problemas de corte são bastante abrangentes e adaptáveis à vários setores da indústria, onde moldes de peças são cortados a fim de produzir um produto final. Um algoritmo de encaixe destes moldes é de grande valia devido à economia de matéria-prima.

2-Sumário:

Este relatório está organizado segundo as secções abaixo:

1- Introdução: apresentação e motivação do projeto;
2- Sumário: organização do projeto;
3- Síntese do Artigo: principais aspectos do artigo que originou o projeto;
4- Problema: descrição do problema e dados utilizados;
5- Modelagem: descrição do método utilizado para a resolução do problema;
6- Resultados computacionais: descrição dos resultados obtidos e sua análise;
7- Discussão: apresentação e discussão de tópicos envolvidos;
8- Conclusão: descrição da qualidade dos resultados;
9- Apêndices, encontram-se todos os programas utilizados;
10- Referências, artigos utilizados para construção do projeto.

3-Síntese do Artigo:

"Complete and Robust No-Fit Polygon Generation for the Irregular Stock Cutting Problem"

3.1-Introdução:

O *No-Fit Polygon* é uma construção que pode ser usada entre pares de formas para obter-se uma manipulação geométrica rápida e eficiente em problemas bidimensionais de corte e empacotamento. Primeiramente, a técnica de *No-Fit Polygon* (NFP) não era comumente aplicada devido a idéia de sua difícil implementação e pela falta de outras abordagens robustas que possam lidar com todos os casos de problemas sem que precise-se abordar caso-a-caso especificamente. Este artigo introduz um método orbital robusto para se obter o *No-Fit Polygon* (NFP) que não sofre dos problemas típicos das outras abordagens, mais ainda o método envolve apenas dois estágios de manipulação geométrica o que faz com que seja um método de fácil compreensão e implementação. Também é abordado como o método se dá dado problemas degenerados como: polígonos com buracos, polígonos que se encaixam como quebra-cabeças (problema chave-fechadura) e polígonos com concavidades que se entravam.

O problema bidimensional de corte é um problema do tipo NP Completo (não pode ser resolvido com um algoritmo de tempo polinomial) ou seja de difícil solução. Com isso vem sido estudado várias estratégias de se abordar o problema, tais como aproximações de progamação linear, métodos heurísticos e metaheurísticos (nos quais se sacrifica a

obtenção a solução ótima em prol da rapidez computacional) entre outros. De qualquer maneira todas estas formas de se abordar o problema dependem inicialmente da forma geométrica do molde, peça, polígono, logo o problema pode passar a ser altamente complicado dado que a forma inicial seja muito irregular, possuir buracos ou concavidades. A implementação de uma rotina robusta e eficiente de geometria pode ser muito trabalhosa e levar muito tempo, dado que a rotina tem que lidar com todas as interações possíveis entre as formas; se elas estão se tocando, se sobrepondo, e calcular a distância de translação necessária para que não haja mais sobreposição entre as peças. Embora esses tipos de testes possam ser feitos utilizando técnicas de trigonometria, o uso do *No-Fit Polygon* apresenta uma resolução consideravelmente mais eficiente, como veremos.

Entretanto a geração do NFP é uma ferramenta e não uma solução. Talvez seja esta a razão para que muitos artigos façam o uso da técnica mas não apresentem detalhes de sua implementação. Aqui focaremos no *No-Fit Polygon* e faremos uma breve apresentação de técnicas antecedentes de sua criação. Mais que isso, ainda descreveremos e apresentaremos uma implementação completa e detalhada de uma nova abordagem orbital que possa lidar com casos de problemas que a maioria das abordagens não consegue. É esperado que esta abordagem ajude a disseminar ainda mais os benefícios de se usar o *No-Fit Polygon* (em oposição às abordagens tradicionais baseadas em trigonometria), em ambos os setores da indústria e nas comunidades acadêmicas.

3.2- No-Fit Polygon - Uma Visão Geral:

Nesta secção é descrita a funcionalidade de um NFP e comparamos sua eficiência em relação aos mais tradicionais testes trigonométricos de sobreposição e intersecção. São também brevemente apresentadas as principais técnicas anteriores de geração de NFP.

3.2.1 – O NFP:

A principal função de um NFP é descrever a região na qual dois polígonos se interseccionam, para melhor compreenção faremos a análise de um exemplo.

Dado dois polígonos, A e B, o NFP_{AB} pode ser obtido percorrendo uma das formas pelas bordas da outra. Um dos polígonos permanece fixo em sua posição e o outro desliza ao redor de suas bordas, sempre certificando-se que os polígonos se tocam mas nunca se sobrepõem. Neste artigo é adotada a convenção de sempre fixar o primeiro polígono e movimentar/orbitar o segundo. Para encontrarmos o NFP_{AB} primeiramente precisamos escolher um ponto de referência em B, que pode ser arbitrário e no caso usaremos o primeiro vértice de B. É importante armazenar a posição relativa do ponto de referência, pois ele é usado no teste de intersecção conforme exibido na Fig.1

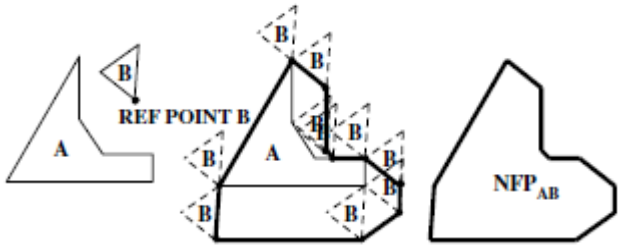


Fig. 1. The no-fit polygon of two shapes *A* and *B*.

Para testar se o polígono B sobrepõe o polígono A, é usado o NFP_{AB} e o ponto de referência de B. Se o ponto de referência do polígono B está posicionado dentro do polígono NFP_{AB} , significa que ele sobrepõe o polígono A. Se o ponto de referência está sobre o limite do NFP_{AB} , significa que o polígono B apenas toca o polígono A. Finalmente,

se o ponto de referência está fora do NFP_{AB} , significa que os polígonos A e B não estão sobrepostos e também não se tocam (Fig.2).

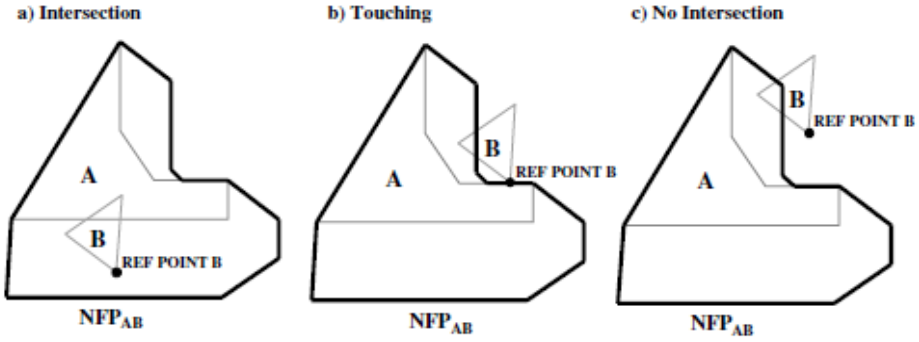


Fig. 2. Using the no-fit polygon to test for intersection between polygons *A* and *B*.

3.2.2- No-Fit Polygon Vs. Detecção trigonométrica padrão de sobreposição:

Apesar do NFP ser uma excelente ferramenta para realização de testes de sobreposição entre dois polígonos, não é o método mais usado em problemas de corte bidimensional. Com isso o método mais comum a ser usado é a abordagem por rotinas trigonometricas. Embora os dois métodos tragam soluções, o uso do *No-Fit Polygon* pode ser muito mais rápido que as mais eficiente das rotinas trigonométricas. Por exemplo, em um problema que se tenha que fazer muitas iterações, a pré-geração de NFP pode reduzir significativamente o tempo total computacional. Quando são necessárias muitas iterações, as abordagens trigonométricas possuem a tendência de detectar e resolver repetidamente o mesmo problema de sobreposição em orientações e posições repetidas. Enquanto uma abordagem de resolução de sobreposição de formas (*nesting*) requer o cálculo de todos os pontos de intersecção entre duas formas que se interceptam, os benefícios da abordagem por *No-Fit Polygon* é ainda maior quando se precisa fazer vários cálculos de intersecção para a detecção completa de colisões. Portanto, utilizando *No-Fit Polygon*, é possível reduzir o problema de detecção de sobreposição (que é um fator importante de sobrecarga computacional do processo de *nesting*) para um teste de "ponto no interior do polígono" significativamente mais barato. Além disso, quando se utiliza a técnica de resolução de sobreposição utilizado por *Burke*, onde a sobreposição é solucionada através da resolução repetida de intersecção das arestas na direção do eixo-y, utilizando o *No-Fit Polygon* a técnica de resolução é mais eficiente, resolvendo a sobreposição no eixo-y em um único movimento completo. Além disso, o *No-Fit Polygon* permitiria a solução da intersecção em qualquer direção, definindo um raio a partir do ponto de referência relevante e encontrando a intersecção mais próxima com a borda do NFP.

Agora serão comparados os processos geométricos necessários para detectar e resolver a sobreposição de dois polígonos, em um exemplo, onde primeiramente aplicaremos a técnica padrão de trigonometria e em segundo o *No-Fit Polygon*.

Dado dois polígonos A e B, A com 7 vértices e B com 6 (Fig.3), desejamos determinar todos os pontos de intersecção entre eles. Todas as bordas são testadas umas contra as outras levando a 42 testes para determinar se há intersecção, caso nenhuma intersecção seja encontrada ainda deve-se testar se os polígons se interceptam nos vértices, ou se um polígono contem completamente o outro. Para isso deve ser feito um teste de "ponto no interior do polígono" para todos os pontos dos dois polígonos, sedo assim necessários mais 13 testes. Portanto no pior dos casos temos um total de 55 testes.

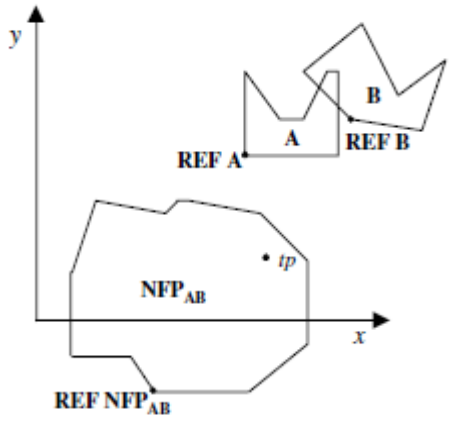


Fig. 3. Intersection testing with the no-fit polygon.

Estes números podem ser considerados como uma grande sobrecarga, porém inevitável para a detecção de intersecção entre quaisquer dois polígonos através de métodos trigonométricos, vale observar que o numero de testes sempre aumentará ao aumentarmos o número de polígonos a serem tratados.

Abordando agora o método *No-Fit Polygon*, o NFP_{AB} que também é apresentado na Fig3, em uma posição arbitrária no espaço, foi obtido pela técnica de se orbitar um polígono ao redor de outro que é mantido fixo. A coordenada em que o NFP_{AB} foi criado dentro do espaço cartesiano não afeta o uso de detecção baseado em intersecção do *No-Fit Polygon*, pois podemos simplesmente gerar o ponto de teste correto (*tp*) com relação à posição do NFP_{AB} , utilizando um cálculo simples, baseado nos pontos referência de A e B

$$\text{tp} = \text{REF NFP}_{AB} + \text{REF } B - \text{REF } A - \text{Offset}$$

onde *Offset* é o vetor que vai do ponto de referência de A (polígono estacionário, fixo), até o ponto de referência de NFP_{AB} (REF NFP_{AB} - REF A). O posicionamento de *tp* em relação ao NFP_{AB} pode ser obtido através do algoritmo de cruzamento de raios - “Ray-Crossing” de *O'Rourke*. Se o ponto *tp* estiver dentro de NFP_{AB} então os polígonos A e B estão colidindo, o termo colisão inclui os polígonos estarem se sobrepondo ou estarem completamente inseridos um no outro. Se o ponto *tp* estiver bem em cima da borda do NFP_{AB} então os polígonos se tocam e por fim se o ponto *tp* estiver fora do NFP_{AB} então os polígonos não estão nem se tocando, nem se colidindo. Portanto calculando todos os NFP para todos os pares de polígonos a serem tratados, consegue-se uma economia considerável de tempo computacional quando trabalhando-se com múltiplos polígonos.

3.2.3- Abordagem da Construção do *No-Fit Polygon*:

Nesta secção as principais técnicas precedentes de construção do *No-Fit Polygon* são apresentadas e para cada método uma breve análise de sua eficiência, destacando as vantagens e desvantagens que seu uso pode trazer.

3.2.3.1 - Formas Convexas:

A forma mais básica de um NFP é quando ele é gerado a partir de dois polígonos convexos A e B (um polígono é convexo se e somente se para todo segmento de reta cujas extremidades pertencerem ao polígono possuírem apenas pontos da região delimitada pelo polígono).

Para gerar o NFP_{AB} basta seguir os passos:

(i) Orientar de maneira anti-horária o polígono A e de maneira horária o polígono B (Fig.4a)

(ii)Transladar todos os vetores de A e B para um único ponto (Fig.4b)

(iii)Concatenar todos estes vetores em sentido anti-horário, para formar o NFP_{AB} (Fig.4c)

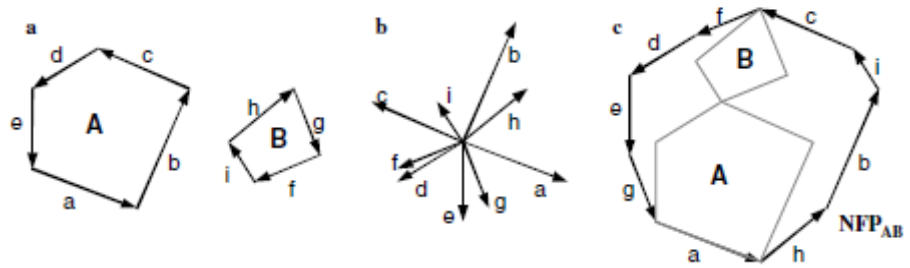


Fig. 4. No-fit polygon generation with convex shapes.

A vantagem da geração no NFP convexo é que ele é simples e extremamente rápido quando usando um algoritmo padrão de classificação combinado com reordenação de vetores através de translação. A desvantagem óbvia desta técnica é que um NFP não pode ser gerado a partir de polígonos não convexos, e a transformação de polígonos não convexos em "cascas convexas" para o uso desta técnica não trazem bons resultados, sendo necessária então outras formas de se abordar polígonos não convexos.

3.2.3.2 – Formas não-convexas:

Existem várias formas de se abordar o problema de polígonos não convexos, porém existem 3 principais que são: Decomposição, soma Minkowski e abordagem orbital.

(i) Decomposição:

Trata-se de decompor formas não-convexas em sub-partes que podem ser manipuladas mais facilmente. Esta técnica geralmente resulta em inúmeras sub-partes, logo vários NFP devem ser criados para os testes de intersecção. Para se realizar os testes de intersecção os NFP podem permanecer decompostos ou podem ser recombinados, um único NFP necessitaria menos tempo para ser efetuar os testes, mas seria necessário mais tempo computacional para se reordenar os NFP decompostos em um único NFP. Isto pode ser muito caro computacionalmente e de difícil realização se houver várias sub-partes, e ainda mais complicado se houverem buracos. Assim o tempo e a dificuldade de se manipular operações de decomposição e reordenação, fazem este método não compensar se as formas forem bastante irregulares. Por exemplo, *Agarwal*, *Flato* e *Halperin* realizaram uma extensa investigação de diferentes operações de decomposição e reordenação em relação à construção de somas de Minkowski de polígonos não-convexos, e chegaram a conclusão de que não é produtivo usar decomposições ótimas pois o tempo computacional requerido superam os benefícios obtidos durante a recombinação. Relatam também que as operações de recombinação são as mais caras, e seu tempo de execução varia bastante dado se a forma for pouco ou muito complexa. *Avnaim* e *Boissonat* discutiram uma abordagem de decomposição e recombinação com base em segmentos lineares, onde estes eram usados para construir paralelogramos que por sua vez eram recombinados para criar o NFP, são fornecidas provas matemáticas de que a abordagem é capaz de lidar com o caso geral e de que é possível manipulá-lo para permitir-se rotações. O algoritmo possui um tempo de complexidade de O(m^3n^3 logmn).

(i-a)Decomposição de formas convexas: Já sabemos que gerar NFP a partir de polígonos convexos é trivial, logo se conseguirmos decompor polígonos não convexos em partes convexas, poderemos aplicar a técnica que já conhecemos. A principal dificuldade continua sendo a abordagem de algoritmos para a

decomposição e reordenação. Existem várias técnicas conhecidas que podem ser usadas incluindo a técnica de triangularização para decomposição sugerida por *Seidal* com complexidade O(nlogn). Porém, para nosso propósito, a triangularização gerará mais sub-partes do que são necessárias, que mais tarde afetará o tempo computacional do programa. Ao contrário da triangularização, que pode ser vista como um caso especifico de decomposição, algoritmos gerais de decomposição tem como objetivo o menor número de sub-partes possíveis. As duas principais abordagens são decomposições subótimas, que tem uma complexidade O(n) (*Hertel* e *Mehlhorn*), e decomposição ótimas que tem uma complexidade O(n^3) (*Chazelle* e *Dobkin*). *Agarwal*, *Flato* e *Halperin* afirmam que geralmente é mais eficiente usar decomposições subótimas devido ao baixo custo computacional, mas eles também sugerem que uma alternativa possivelmente mais eficiente seria a realização de cobertura por uma casca convexa. Uma vez que quaisquer polígonos irregulares forem decompostos em partes convexas, o NFP pode ser gerado pela passagem de cada peça convexa de forma B em torno de cada peca convexa de forma A. A desvantagem desta abordagem é que os NFP das peças convexas podem se cruzar, logo deve se tomar cuidado na hora de recombina-las para formar o NFP geral. Uma dificuldade particular aparece se os polígonos originais possuírem buracos, pois ainda não sabemos se a intersecção dos NFP das sub-partes definem buracos ou regiões que podem ser descartadas.

(i-b)Polígonos de forma-estrela: *Li* e *Milenkovic* decompuseram as formas em convexas e formas-estrela. Um polígono de forma-estrela possui a propriedade de existir pelo menos um ponto interno (núcleo), ou *'kernel point'*, que pode "ver" todas as bordas do polígono. Ao se estender as bordas da concavidade e eliminar regiões invisíveis é evidente que o polígono de forma-estrela possui uma reginao, $R_{kernel,}$ que pode ser definida como onde um *'kernel point'* pode ser colocado para 'ver' todas as bordas do polígono. Assim polígonos de forma-estrela são definidos como formas que estão de certa maneira entre as formas convexas e não-convexas. *Li* e *Milenkovic* afirmam que polígonos forma-estrela são “fechados” no sobre as operacões de soma de *Minkowski* e comprovam que a soma *Minkowski* de dois polígonos forma-estrela também resulta em um polígono de forma-estrela. Os autores não indicam se eles recombinaram as regiões NFP em uma entidade NFP ou se executaram múltiplos testes de intersecção de NFP durante a fase de geração do NFP.

(i-c) Função-Phi(Φ): Uma abordagem alternativa é apresentada por *Stoyan* e se baseia no uso de funções-phi. Funções-Phi descrevem matematicamente as relações de intersecção entre pares de objetos padrões ou objetos 'primários', assim como retângulos, círculos e polígonos convexos. Embora funções-phi não sejam uma abordagem estritamente baseada em *No-Fit Polygon*, ela foi incluida por apresentar resultados promissores. Os autores aprofundaram seu trabalho para possibilitar uma definição matemática da relação de intersecção de polígonos não-covexos através da união, intersecção e complemento de objetos primários. O resultado do teste de intersecção entre duas formas durante a geração do NFP final é dado por comparações de funções-phi entre todos os pares de objetos primários que definem a forma A e a forma B. Funções-phi possuem uma segunda vantagem como são funções baseadas nas distâncias entre objetos primários, pode facilmente ser usada para encontrar a distância entre dois objetos complexos.

(ii)Soma *Minkowski*:

A construção do *No-Fit Polygon* pode ser unificada através do uso da soma de *Minkowski*, que é uma maneira de se somar vetores. O conceito é: dado dois conjuntos de pontos arbitrários, A e B, a soma *Minkowski* de A eB é definida por:

$$A \oplus B = \{a + b : a \in A, b \in B\}.$$

De modo a se produzir NFP, devemos usar também a diferenca *Minkowski* :

$$A \oplus -B.$$

que corresponde a dois polígonos iniciais em orientações opostas e é simples demonstrar através de algebra vetorial. Podemos verificar que este é o caso, usando o nosso caso convexo simples, onde colocamos a forma A em orientação anti-horária e a forma B em orientação horária. Enquanto detalhes não-matemáticos da implementação de tal abordagem são escassas, *Ghosh* e, posteriormente, *Bennell* forneceram explicações excelentes e detalhadas da implementação para o cálculo da região do *No-Fit Polygon*. A principal desvantagem do método é que ele só funciona assegurado que as concavidades das duas formas não irão se entravar. *Bennell, Dowsland* e *Dowsland* afirmam que a abordagem de *Ghosh* causaria um "emaranhado pesado de intersecção das arestas" que seria difícil de recombinar para formar o NFP, também introduzem uma implementação que reduz a quantidade de "bordas emaranhadas" e dão detalhes de implementação do processo e relatam rápidos tempos de geração. No entanto, afirmam que sua abordagem não pode lidar com buracos, pois é difícil detectar qual das arestas internas do NFP podem ser descartadas e quais formam as regiões internas do *No-Fit Polygon*.

(iii)Abordagem de Deslizamento Orbital:

Esta abordagem de deslizamento orbital é o ponto principal deste artigo, e envolve o uso de trigonometria básica para fisicamente deslizar um polígono ao redor do outro. O NFP é definido pela trajetória de um ponto do polígono deslizante ao orbitar o outro. A primeira discussão e implementação de uma abordagem orbital para gerar formas 'envelopes' é de *Mahadevan*. Os elementos chave da abordagem de *Mahadevan* são: calculo de vértices e bordas que se tocam, determinação do vetor de translação e calculo do comprimento da translação. O calculo da intersecção das arestas é realizado usando a noção de funções- D de *Konopasek*. *Mahadevan* modificou o teste da função-D para que também calculasse pontos de contato, que é necessário tanto para o algoritmo de *Mahadevan* quanto para a nossa nova abordagem. Esta informação é então utilizada para selecionar um vetor de translação baseado na aresta tocada. O vetor de translaçnao é então projetado em cada vértice do polígono orbitante, e então o teste de intersecção das arestas é usado para calcular a distância de translação. É também importante projetar o vetor de translação na direção reversa do polígono estacionário. O polígono orbital é então transladado ao longo do vetor de translação pela menor distância (dada da projeção e testes de intersecção). Isso garante que os polígonos nunca se interceptem mas sempre se toquem. O processo continua até que o polígono orbitante volte a sua posição inicial. Não descreveremos a abordagem de *Mahadevan* em mais detalhes simplesmente porque muitas das suas características também são utilizadas na nossa nova abordagem orbital, que onde possível compararemos com a abordagem de *Mahadevan* e estabelecemos nossas melhorias. A maior desvantagem no algoritmo de *Mahadevan* é que não se consegue gerar um NFP completo para formas que envolvam buracos ou algumas concavidades. O problema ocorre quando o polígono orbitante pode ser colocado dentro da concavidade do polígono fixo, mas a "entrada" da concavidade é muito estreita. Neste caso, o polígono orbitante é muito largo e simplesmente passa "reto", como se não houvesse entrada para a concavidade. Mais adiante mostraremos

como nossa nova abordagem pode gerar NFP para todos os casos degenerados conhecidos.

3.3– Contrução do novo algoritmo *No-Fit Polygon*:

Iremos agora apresentar nossa nova abordagem robusta para geração do *No-Fit Polygon* através de métodos orbitais utilizando técnicas básicas de trigonometria. Para o calculo das intersecções em nossa implementação é importante o uso de uma robusta biblioteca de geometria. É também vantajoso implementar rotinas que sejam rápidas e que sejam o mais próximas possível da ótima, em razão de produzir um algoritmo ágil para NFP. Nos implementamos tal biblioteca, embora isso possa consumir muito tempo e seja de difícil compreensão, mas este detalhe de implementação, saber qual rotina geométrica será utilizada, vai além do nosso foco e não será discutida.

3.3.1-Visão geral:

Nossa abordagem pode ser dividida em dois estágios. Secção 3.3.2 descreve o procedimento para o primeiro estágio, onde um polígono desliza em torno do outro para criar o caminho externo (borda) do NFP. Essa parte segue uma lógica de abordagem similar ao algoritmo de *Mahadevan*, entretanto uma modificação na implementação é sugerida. A segunda secção introduz a noção e a identificação das posições de início, *'start positions'*, que permitem o algoritmo encontrar os caminhos (bordas) restantes do NFP(secção 3.3.3). Esses caminhos restantes serão os buracos internos do NFP e não são encontrados usando-se somente o algoritmo de *Mahadevan*.

Para essa secção assumiremos que existem dois polígonos com orientação anti-horária, A e B, que estão em algum lugar de um espaço bidimensional, e desejamos produzir o NFP_{AB} (orbitando o polígono B ao redor do polígono A). A primeira operação a ser executada é transladar o polígono B a fim de que ele toque, mas não interseccione, o polígono A. Mantivemos a mesma abordagem que foi usada por *Mahadevan* pela qual o polígono B é transladado de forma que a maior coordenada y fique posicionada na menor coordenada y do polígono A (Fig.5)

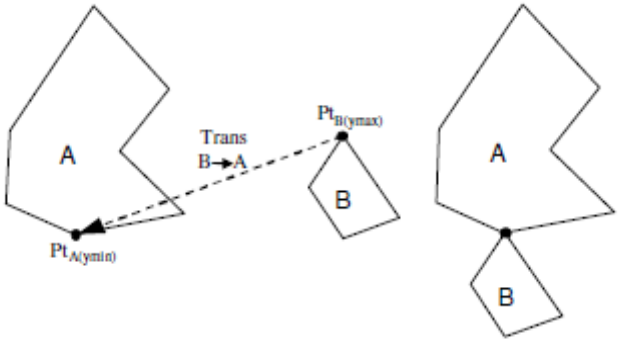


Fig. 5. Initial translation of the orbiting polygon to touch the stationary polygon.

Usando estes dois vértices como alinhamento, garante que A e B não estarão sobrepostos, mas apenas tocando-se. O resultado da translação que resulta em B tocando A é mostrado na fórmulaI:

$$\text{Trans } B \rightarrow A = \text{Pt}_{A(y\min)} - \text{Pt}_{B(y\max)}.$$

Na realidade, qualquer posição inicial pode ser usada dado que os polígonos A e B estejam apenas se tocando. A partir deste momento, o processo de orbitação pode começar a gerar a borda externa do NFP em sentido anti-horário.

3.3.2 – Orbitando/ Deslizando:

O principal objetivo da orbitação (ou deslizamento) é detectar os movimentos corretos que B deve fazer para percorrer ao redor de A a fim de retornar a posição original. Esse é um procedimento iterativo onde cada passo cria uma aresta do NFP. Este processo pode ser quebrado em sub-partes que são descritas nesta seção uma de cada vez. Primeiramente é descrita a detecção de arestas que se tocam. A seguir a criação de potenciais vetores de translação. Na sequência é encontrada uma translação viável. Logo após é explicado o ajuste da translação viável e por fim é aplicado o vetor de translação.

3.3.2.1- Detectando arestas que se tocam:

A capacidade de detectar corretamente arestas que se tocam ou se inteceptam é de suma importância para o sucesso do algoritmo. Isso é feito testando cada aresta do polígono A contra cada aresta o polígono B. Cada par de arestas que se tocam (uma do polígono A e uma do polígono B) é armazenado com relação a posição dos vértices em comum. Fig6 mostra os pares resultantes de arestas que se tocam. Isto se contrasta com a abordagem de *Mahadevan* que executa cálculos usando todas as arestas em um vértice que se toca. Por exemplo na Fig6., *Mahadevan* apresentaria as arestas a_2 , a_3 , b_1 e b_4 no mesmo diagrama e, com nossa experiência podemos dizer que isso faz com que os cálculos necessários fiquem cansativos e de difícil explicação.

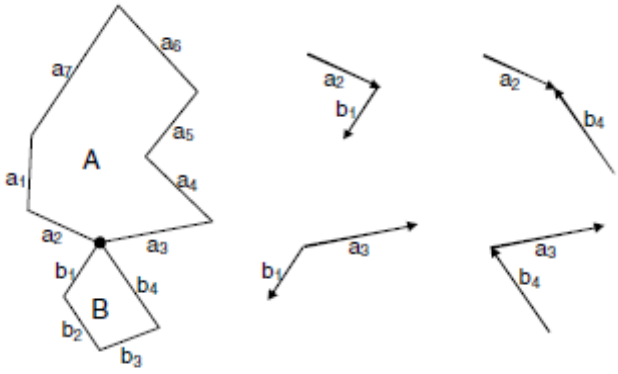


Fig. 6. Identification of touching edge pairs.

3.3.2.2 - Criação de potenciais vetores de translação:

O vetor com o qual o polígono B deve ser transladado para orbitar o polígono A deve ser derivado do polígono A ou do B dependendo da situação. A Fig7. mostra um exemplo de cada caso:

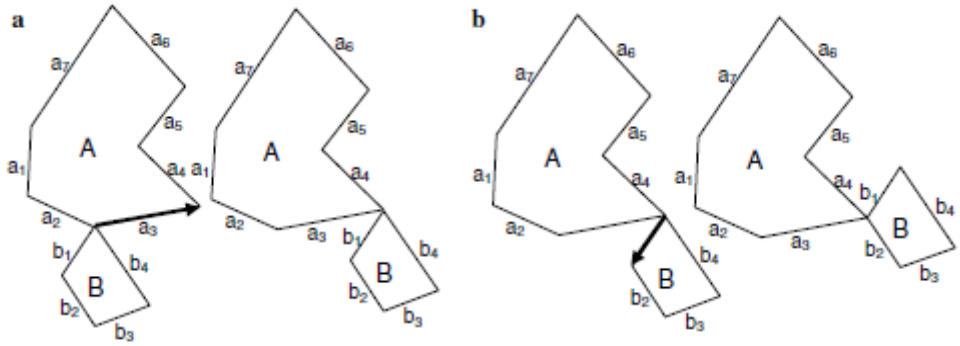


Fig. 7. Translation vector: (a) derived from edge a_3 and (b) derived from edge b_1.

Note que no segundo caso (Fig7.b) devido que o polígono B desliza ao longo de uma de suas próprias arestas, o vetor de translação é encontrado invertendo a aresta. Isso é examinado mais a fundo através do movimento relativo do polígono A em relação a B. O

polígono A é relativamente movimentado ao longo da aresta b_1. Na realidade, o polígono A deve permanecer fixo e B deve ser transladado e por isso o vetor de translação é invertido neste caso. Podemos obter um conjunto de potenciais vetores de translação usando os pares de arestas que se tocam. Existem três possibilidades:

(i)quando ambas as arestas se tocam em um vértice conforme mostra a Fig8.a.
(ii)quando um vértice da aresta móvel toca o meio da aresta fixa conforme mostra a Fig8.b.
(iii)quando um vértice da aresta fixa toca o meio da aresta móvel, exibido na Fig8.c.

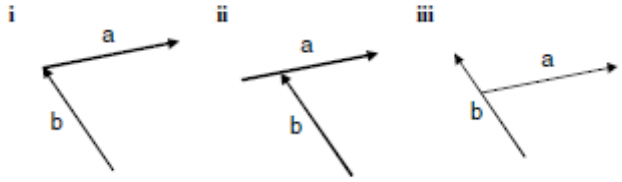
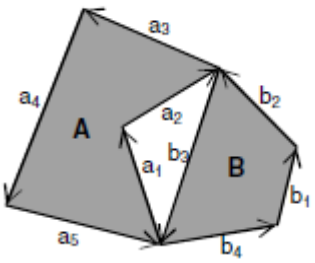


Fig. 8. Touching edge-pair types.

Cada par de arestas que se tocam produz um potencial vetor de translação. No casao(ii), o vetor de translação é simplesmente definido usando o ponto no qual as duas arestas se tocam e o ponto final da aresta estacionária . No caso (iii), usamos um processo semelhante exceto que usamos o ponto final da aresta orbitante e também invertemos a direção do vetor. O caso (i) requer uma identificação correta se o potencial vetor de translação é derivado da aresta fixa ou móvel. Isso pode ser identificado seguindo uma série de regras baseada nos vértices em comum e testando se a aresta móvel, b, está à esquerda ou à direita da aresta fixa, a. O Quadro 1 mostra as diferentes possibilidades e a aresta da qual um potencial vetor é derivado em cada circunstância:

Caso	Vértices das arestas que se tocam		Posição relativa da aresta móvel/fixa	Origem do vetor de translação
	Fixo	Móvel		
1	início	início	esquerda	Aresta móvel
2	início	início	direita	Aresta fixa
3	início	fim	esquerda	-
4	início	fim	direita	Aresta fixa
5	fim	início	esquerda	-
6	fim	início	direita	Aresta móvel
7	fim	fim		-
8			paralelo	Ambas arestas

Agora devemos esclarecer porque alguns potenciais vetores de translação podem ser eliminados em certos casos da tabela (um '-' na tabela indica que nenhuma translação é derivada). Relembrando que quando uma translação é derivada de uma aresta, o vetor resultante é definido por um ponto de contado em comum e ponto do final da aresta (e então o vetor é invertido para uma aresta orbital). Isso nos permite eliminar o caso 7 da tabela porque ambas arestas, móvel e fixa, se tocam no ponto final das arestas, e isso geraria um vetor de translação nulo. Nos casos 3 e 4, a aresta móvel toca no ponto final da aresta, assim a translação não pode ser derivada da aresta móvel. Nos casos 5 e 6, a aresta estacionária não pode ser usada, ou produziria um vetor nulo. Como exemplo, mostramos cada caso usando dois polígonos que se tocam em dois lugares separados(Fig.9)



Case	Edges	Translation Vector Derived From
1	(a_3,b_3)	b_3
2	(a_1,b_4)	a_1
3	(a_3,b_2)	-
4	(a_1,b_3)	a_1
5	(a_5,b_4)	-
6	(a_2,b_3)	b_3
7	(a_2,b_2), (a_5,b_3)	-

Fig. 9. Two polygons touching at two separate positions and vertex–vertex touch cases.

Nos casos 3 e 5, nenhuma translação pode ser derivada porque a aresta do polígono orbital esta à direita da aresta do polígono fixo. Por exemplo, as arestas a_3 e b_2 se tocam no começo e no final de seus vértices respectivamente. Como produzimos uma translação nula usando a aresta b_2 (toca o vértice final), a única possibilidade é derivar o vetor de translação é da aresta a_3 do polígono fixo. De qualquer forma, a aresta b_2 está à direita da aresta a_3 e, se esse vetor de translação fosse usado, a aresta b_2 deslizaria ao longo da parte interna da aresta a_3. Portanto, o teste “do lado direito” elimina as translações por onde os polígonos deslizariam ao longo da parte interna de uma aresta ao invés a parte externa como deve ser. Onde arestas são paralelas, ou a aresta estacionária, ou a aresta móvel deve ser usada.

3.3.2.3 - Encontrando uma translação viável:

Uma vez que os potenciais vetores de translação foram produzidos, o próximo estágio é selecionar um vetor de translação que não resulte em uma intersecção imediata. Por exemplo, na Fig.9 geramos dois potências vetores de translação, a_1 e $-b_3$. Isso pode ser visto como o polígono móvel B deve ser transladado usando o vetor $-b_3$, em razão de mover no sentido anti-horário ao redor do polígono A . Se, ao invés nos transladássemos o polígono B ao longo do vetor a_1, isso resultaria em uma intersecção imediata entre as arestas a_2 e b_3 (e também a_3 e b_3). Mais uma vez o conjunto de pares de arestas que se tocam, gerados da secção 3.3.2.1, contem toda informação necessária para se determinar um vetor de translação viável. O processo consiste em pegar cada um dos potenciais vetores de translação e colocá-lo na posição em que as arestas se tocam. Isso deve ser feito para cada par de arestas. O relacionamento existente entre as arestas fixa e móvel pode ser definido com base na união das regiões da esquerda/direita, o que indica se um vetor de translação em particular será adequado para aquelas arestas. A Fig10. mostra alguns exemplos de pares de arestas que se tocam e o intervalo angular das translações viáveis.

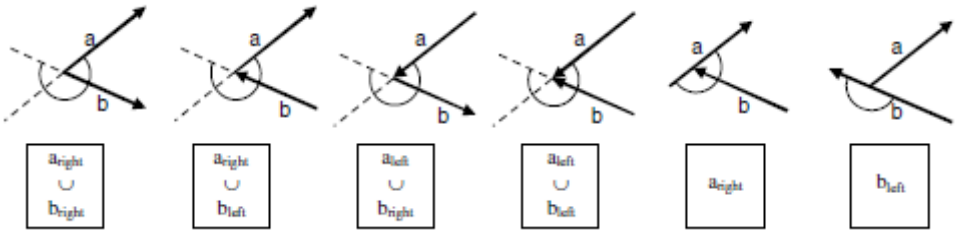


Fig. 10. Identifying the feasible angular range of translations (indicated by an arc).

Para sermos mais breves, não listamos todas as possibilidades que podem ocorrer (os exemplos da Fig.10 fornecem informações suficientes para derivar as combinações omitidas). Na Fig.9 calculamos dois potências vetores de translação, $-b_3$ e a_1, para dois polígonos que se tocam. Fig.11 demonstra como a abordagem elimina o vetor de translação a_1 (omitimos o par de arestas envolvendo a aresta a_1 para sermos breves mas

estes serão translações viáveis pois o vetor de translação também é a_1). Uma vez que uma potencial translação falha em algum destes testes, significa que a translação é inviável e portanto pode ser eliminada sem mais testes.

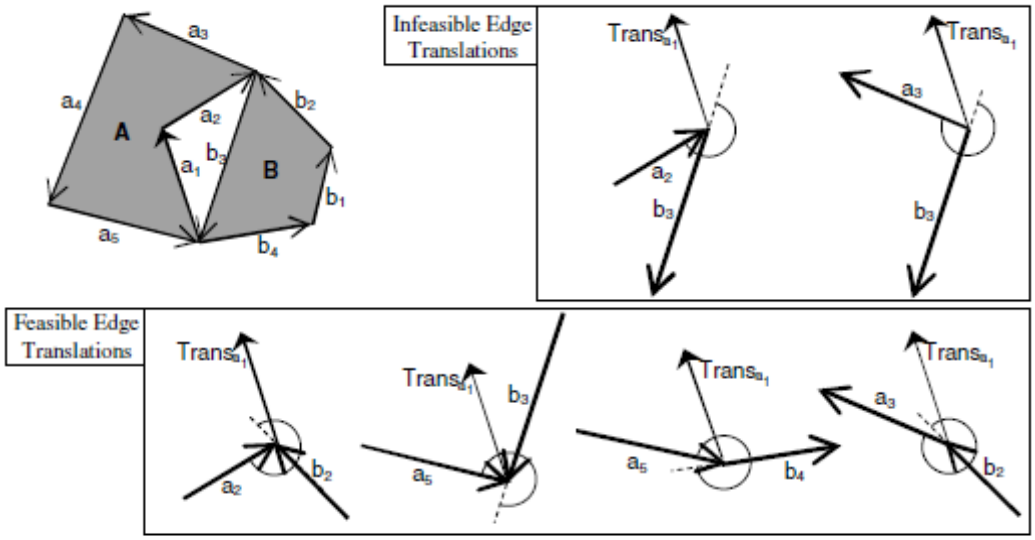


Fig. 11. Elimination of potential translation vector, a_1.

Neste estágio encontramos um (ou mais) vetores de translação viáveis. Normalmente haverá somente um vetor de translação, mas vários podem estar presentes dado circunstâncias especiais, que são ilustradas na Fig.12. O centro do quadrado móvel foi marcado como ponto de referência para clareza.

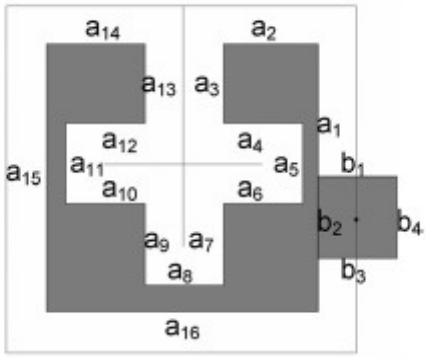


Fig. 12. Two polygons involving exactly fitting 'passageways'.

Isso introduz outro aspecto importante do nosso algoritmo, devemos manter a aresta que foi usada para gerar o vetor de translação anterior. Quando vários vetores de translação viáveis estão presentes, escolhemos a aresta que esta mais perto (na ordem das arestas) do movimento anterior. Sendo assim, retornando a Fig.12, o quadrado entra pela "passagem" usando a aresta a_3 ao invés de deslizar diretamente pela abertura usando a aresta a_{14}. Então, depois de deslizar para o centro do polígono estacionário, o quadrado possui quatro vetores de translação viáveis derivados das arestas a_4, a_7, a_{10} e a_{13}. Entretanto, o vetor de translação viável derivado da aresta a_4 será usado pois a aresta usada anteriormente foi a aresta a_3. Esta é outra melhoria que foi feita da abordagem de *Mahadevan*, significativamente melhorando a generalidade da abordagem.

3.3.2.3- Ajustando a translação viável:

O último passo antes que o polígono B possa ser transladado é ajustar o vetor de translação. Isso é importante porque pode haver outras arestas que podem interferir na translação do polígono móvel. A Fig.13a provê um exemplo onde a aplicação

do vetor de translação por completo resulta na intersecção das duas formas. A fim de prevenir que o polígono móvel "entre" no polígono fixo, a translação viável, a_7, deve ser ajustada de acordo com a aresta a_1, conforme demonstrado na Fig.13b

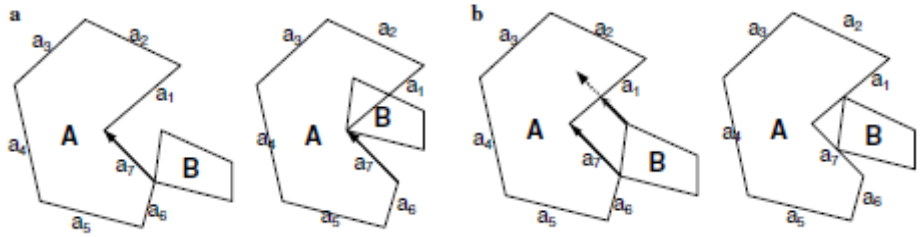


Fig. 13. A translation vector that requires 'trimming' to avoid intersection.

No intuito de encontrar a translação que não causa intersecção, o vetor de translação é projetado em cada um dos vértices do polígono B e é testada a intersecção com todas as arestas do polígono A. Isso garante que identifiquemos corretamente os vértices do polígono B que iram "entrar" no polígono A então diminuímos a distância de translação de acordo com a formulaII:

$$\text{Nova translação} = \text{InstersecçãoPt} - \text{Translação}_{\text{StartPt}}$$

Devemos também projetar o vetor de translação de volta a partir de cada um dos vértices do polígono A (através da translação do ponto do fim do vetor de translação a cada vértice), para então ser testado se há intersecção com alguma das arestas do polígono B. Isso é retratado na Fig.14, onde é usado o mesmo exemplo, porém utilizando um polígono móvel diferente.

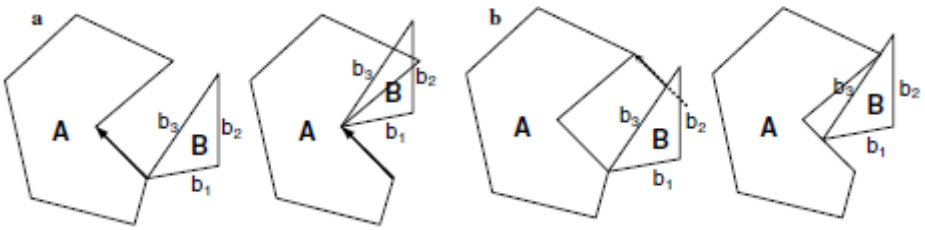


Fig. 14. Trimming with projections from polygon *A*.

Mais uma vez, se há uma interseção, podemos reduzir a distância de translação, aplicando a formulaIII:

$$\text{Nova translação} = \text{Translação}_{\text{EndPt}} - \text{InstersecçãoPt}$$

Nossa abordagem é basicamente a mesma usada por *Mahadevan* exceto que nós ajustamos o vetor de translação ao mesmo tempo da interseção, onde por sua vez *Mahadevan* mantém o vetor de translação o mesmo, mas armazena a distância mínima de interseção e a ajusta só quando todas as projeções foram feitas. Como nosso algoritmo ajusta a cada intersecção, o vetor de translação vai ficando menor a medida que os testes vão ocorrendo, o que pode diminuir o número de intersecções que ocorrem pelo rápido teste de eliminação usando a tabela que é muito mais rápida que a linha padrão de interseção. Na abordagem de *Mahadevan* pode ser necessário calcular mais testes completos de intersecção dado que o vetor de translação é usado na sua forma completa em cada projeção e pode possivelmente precisar de mais computação.

3.3.2.5- Aplicando o vetor de translação viável:

O último passo é realizar a translação do polígono B e adicionar o ponto de referência do mesmo ao final do NFP parcialmente criado. Isso fará com que o polígono mova-se para o próximo ponto de decisão e o processo possa ser reiniciado do ponto de detecção das arestas que se tocam (seção 3.3.2.1). A única verificação adicionada a este processo é a execução de um teste para detectar se o ponto de referência do polígono B retornou ao ponto inicial.

3.3.3-Pontos de inicio:

Na secção 3.3.2, definimos uma abordagem para um polígono móvel orbitando ao redor de outro através de testes de intesecção. O efeito geral desta abordagem é similar a proposta do algoritmo de *Mahadevan*, entretanto algumas melhorias foram propostas fazendo com que seja necessário menos tempo computacional e que permitem a abordagem ser explicada mais facilmente. Entretanto, nossa abordagem até agora, tem algumas das mesmas limitações que as implementações orbitais anteriores apresentam , tais como a incapacidade de gerar NFP completos para as formas que envolvem concavidades que se entravam, peças tipo quebra-cabeças ou buracos. A Fig.15 mostra dois polígonos no qual usando somente o algoritmo de deslizamento não conseguimos produzir o NFP completo. Os polígonos apresentam concavidades que podem se entravar umas nas outras de tal forma a criar uma nova área de NFP, mas essa região nunca será encontrada devido a estreita entrada do polígono fixo (Fig.15b), logo uma nova abordagem deve ser desenvolvida para lidar com estes casos.

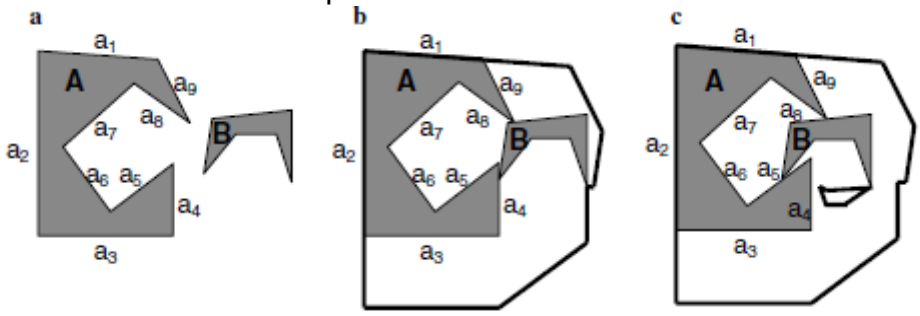


Fig. 15. Interlocking concavities: (a) polygons, (b) no-fit polygon using sliding alone and (c) the complete no-fit polygon.

Nesta secção descrevemos uma abordagem baseada na identificação de pontos de início viáveis que se tocam mas não se interseccioam que em conjunto com a técnica de deslizamento já apresentada geram as demais iterações internas para a construcão do NFP. Por exemplo, se o polígono B pode ser colocado na posição mostrada na Fig.15c, então o algoritmo de deslizamento pode ser aplicado para gerar a região interna do NFP. Pontos iniciais viáveis são encontrados através de uma modificação na abordagem apresentada na secção 3.3.2, onde novamente o processo envolve deslizamentos. Entretanto, não estamos interessados em deslizar para delimitar o NFP mas sim para resolver intersecções de aresta enquanto transladamos ao longo de um vetor-aresta particular. A fim de explicar o processo examinaremos um exemplo, onde desejamos encontrar a posição inicial ao longo de uma borda do polígono A . Esse processo então pode ser facilmente repetido para cada aresta do polígono A e do polígono B, para criar todas as posições inicias viáveis possíveis que permitem o polígono B orbitar o polígono A com o objetivo de gerar o NFP completo. Dado uma borda ***e*** , do polígono A, podemos determinar uma potencial posição inicial para um polígono orbitante ao longo de ***e***. O processo envolve transladar o polígono B tal que cada um de seus vértices, na sua vez, esteja alinhado ao ponto inicial da aresta ***e*** . Para cada posição realizamos os passos:

1) Se os polígonos não se interceptam nesta posição, então ela é tomada como uma posição inicial viável para os dois polígonos.
2) Se o polígono B intercepta o polígono A então devemos realizar mais testes e

por fim aplicar o deslizamento para percorrer ao longo da aresta ***e*** até que uma posição sem intersecção seja encontrada ou o fim da aresta seja alcançado. Em tal caso, o primeiro teste envolve examinarmos se as duas arestas conectadas de B (conectadas no ponto de contato), são paralelas ou estão a direita da borda ***e***. Se alguma das arestas estiver a direita de ***e*** então será transladada para o interior do polígono A, e pode ser eliminada imediatamente, pois nunca vai render uma posição viável de partida ao deslizar ao longo do vetor ***e*** .

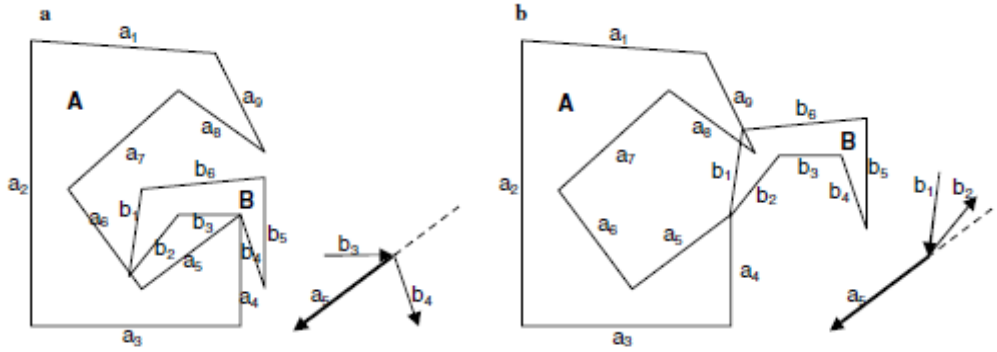


Fig. 16. Vertex alignment of polygon *B* to polygon *A* using edge a_5: (a) invalid alignment and (b) valid alignment.

A Fig.16a mostra um exemplo no qual o par de arestas conectadas, b_3 e b_4, são eliminadas de deslizar por a_5 porque b_4 esta à direita de a_5 (note que este não será eliminado quando usando a borda b_4 do polígono B e as arestas conectadas a_4 e a_5 do polígono A). A Fig.16b mostra um exemplo onde ambas arestas conectadas, b_1 e b_2, estão à direita de a_5.

Agora assumindo ambas arestas conectadas à direita da aresta ***e*** e os polígonos A e B se interceptam, podemos tentar solucionar a sobreposição transladando o polígono móvel ao longo do vetor de translação definido por ***e***. Assim como nas secções anteriores podemos ajustar este vetor (o procedimento é idêntico). O vetor translação resultante pode então ser aplicado para deslizar-se o polígono B ao longo da aresta ***e*** , e então outro teste de intersecção é realizado. Se os dois polígonos continuam a se interceptar então o processo é repetido com o vetor de translação derivado do ponto de contato até o ponto final da aresta ***e*** . Se a intersecção foi solucionada então o ponto de referência de B é um potencial ponto inicial. Se toda a aresta ***e*** é percorrida e ainda existe intersecção então nenhum ponto inicial factível pode ser encontrado ao longo da borda ***e*** e do vértice alinhado das arestas conectadas do polígono B. O próximo vértice do polígono B é então considerado e assim por diante até que todos os vértices das arestas tenham sido examinados. Note que é também importante examinar as bordas do polígono B com os vértices do polígono A mas como isso é um processo idêntico, não o analisaremos. A Fig.17 apresenta este processo do exemplo na Fig.17b. Quando alinhados usando a_5 e o ponto conectando as arestas b_1 e b_2, os polígonos estão inicialmente se interceptando então precisamos solucionar a intersecção ao longo do vetor derivado da aresta a_5.

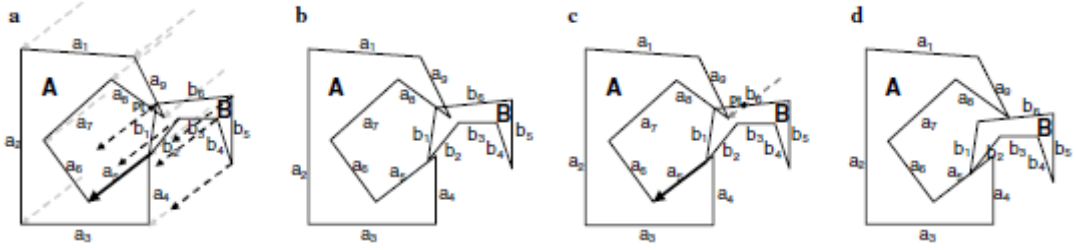


Fig. 17. Start point generation process.

A Fig.17a mostra o processo de ajuste e identificação do ponto de intersecção mais próximo, *Pt*. Então o polígono B é transladado pelo vetor de translação ajustado. A posição resultante é mostrada na Fig.17b. Os polígonos continuam a se interceptar logo o processo deve se repetir. O vetor de translação é calculado a partir do ponto de contato até o ponto final da aresta a_5 e é então ajustado como mostra Fig.17c (somente a intersecção a ser ajustada é mostrada). Depois de aplicar esta translação no polígono B, os polígonos não estão mais se interceptando portando um ponto inicial factível foi encontrado e a abordagem padrão de polígono orbital pode ser aplicada para gerar o NFP. Um procedimento que incluímos em nossa implementação é de que calculamos o contorno externo do NFP usando a abordagem da secção 3.3.2 e então aplicamos o procedimento de ponto inicial nas bordas não percorridas/não marcadas de ambos polí gonos A e B, de cada vez. Durante o processo, logo que uma posição inicial factível é encontrada, calculamos a parte interna do NFP que é derivada dessa posição (marcando as arestas assim que são percorridas como antes). Isso assegura que o algoritmo é rápido pois mais arestas são marcadas assim que novas iterações NFP são realizadas, dessa maneira reduzindo a necessidade computacional de gerar novas posições iniciais factíveis. O procedimento se repete até que todas as arestas tenham sido 'vistas' ou mais nenhuma posição inicial factível esteja disponível e assim retorna o NFP completo. Uma idéia do código para se gerar o NFP e encontrar o ponto inicial factível são apresentadas do Apêndice (A)

3.4 – Casos de Problemas :

Nesta secção serão discutidos os casos que geram problemas nos outros métodos, e mostramos como nossa abordagem é capaz de lidar com eles sem que seja necessária uma implementação específica para cada caso.

3.4.1- Cavidades que se entravam: concavidades que se entravam é um problema típico das abordagens orbitais anteriores. A Fig.18 mostra uma foto de nossa implementação, onde geramos o NFP completo de duas formas idênticas, uma delas rotacionada de 180º. Estas formas envolvem muitas iterações diferentes devido as concavidades, portanto muitos pontos de início factíveis. O *No-Fit polygon* dessas formas foi encontrada após seis iterações.

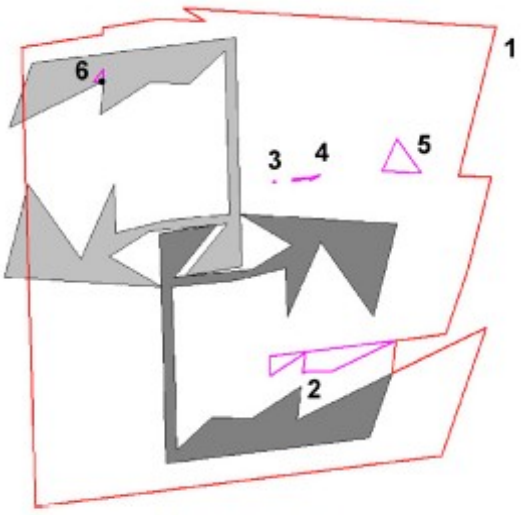


Fig. 18. Multiple interlocking positions.

3.4.2 – Encaixe perfeito: Deslizando: Este caso envolve deslizar através de passagens de tamanho ideal, caso que não pode ser abordado nem por soma de Minkowski, nem pela abordagem orbital de Mahadevan. Nos sugerimos uma extensão no

algoritmo de Mahadevan que possa lidar com tal problema através da manutenção da aresta percorrida anteriormente, assim possibilitando arestas consecutivas do polígono fixo serem usadas na próxima iteração de translação(Fig.19).

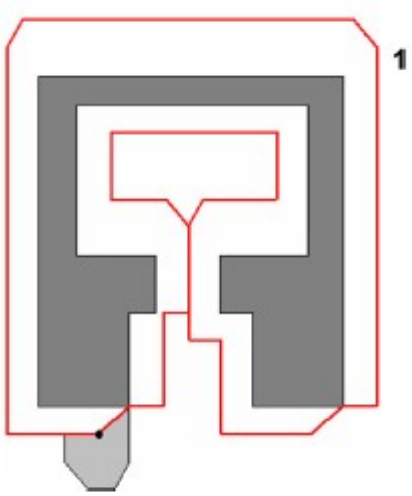


Fig. 19. Exact fit sliding through a "passageway".

3.4.3 – Encaixe perfeito: Peças de quebra-cabeça: O problema envolvendo peças de quebra-cabeça (também conhecido como problema chave-fechadura) trata-se de encaixes perfeitos entre duas peças, este caso pode ser considerado como uma forma de concavidades que se entravam (caso secção 3.4.1). Entretanto a característica das peças de quebra-cabeça é que elas se encaixam sem movimento, assim criando um único ponto viável com o NFP(ao invés de iterações internas como no caso de concavidades que se entravam). Em nossa abordagem, esse ponto é encontrado através da geração da posião inicial factível. É claro que o algoritmo continuará tentando deslizar o polígono móvel ao longo das bordas do polígono fixo, mas neste caso não há vetor de translação viável, e portanto diferenciando que a posição é apenas um único ponto viável (Fig.20). A maioria das abordagens anteriores sofreram com casos de degeneração como este.

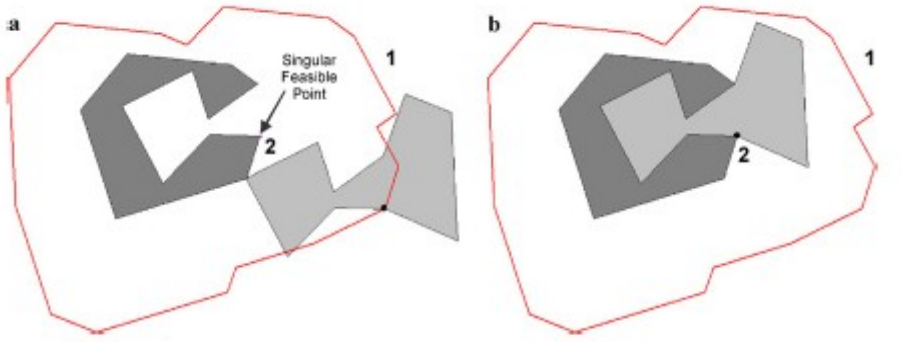


Fig. 20. Jigsaw pieces: (a) outer no-fit polygon loop and (b) singular feasible internal position.

3.4.4.- Buracos: Existem poucas abordagens que conseguem lidar com eficiência com formas contendo buracos. Isso pode ser devido a dificuldade de gerar NFP envolvendo buracos. Contudo, através da detecção de pontos iniciais factíveis, o NFP pode ser gerado facilmente, e mais importante completo.

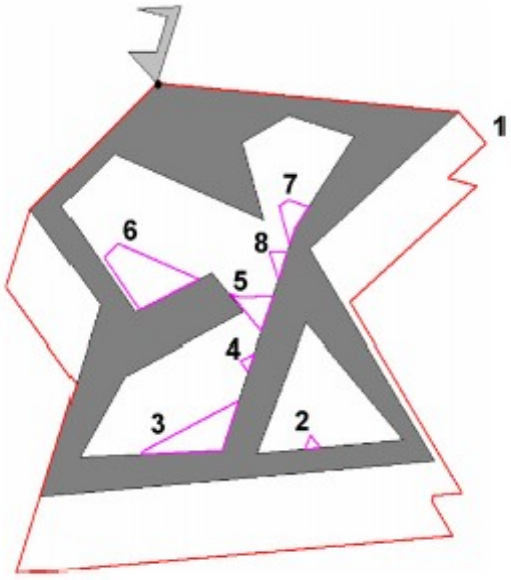


Fig. 21. No-fit polygon of two polygons, one of which involves multiple holes.

A Fig.21 mostra um exemplo envolvendo uma forma contendo dois buracos, e uma forma que pode ser colocada dentro de varias regiões distintas dos buracos. A figura apresenta um caso misto, envolvendo buracos e concavidades que se entravam.

Iteração1- contorno externo do NFP

Iteração 2, 3, 5, 6, 7- casos de buraco

Iteração 4 e 8- casos pelo qual a concavidade da forma orbital interage com as estreitas lacunas no interior dos buracos maiores da forma estacionária. Decomposições em convexos resultaria em um grande número de peças e ficaria ainda mais difícil com a inseção de buracos. Recombinação também é uma estratégia comprometida pelo grande número de interseções dos sub-NFP que precisam ser solucionadas. A Função-phi seria capaz de lidar com a maioria das degenerações, pela manipulação de formas primárias, entretanto essa abordagem causará um impacto na eficiência dos testes de intersecção devido ao número de testes separados que devem ocorrer (especialmete se a forma for muito complexa). Abordagens orbitais anteriores apenas produziram o contorno externo do NFP, e a soma de Minkowski pode se tornar uma abordagem muito difícil dado o "desembaraçamento" das bordas. E não conhecemos nenhum outro método que gere NFP para casos tão complexos com várias degenerações.

3.5 – Tempo de geração nos Problemas de referência:

Com o objetivo de demonstrar a velocidade e capacidade de nosso novo método de gerar NFP, relatamos o tempo de geração para 32 problemas referência
Para cada problema relatamos:

- o tempo total da geração e o número de NFP gerados por segundo;
- "o total número de formas lógicas"(coluna E) dado por E = B*D
- número total de NFP (coluna F) dado por F = E^2 (ié, calculamos o NFP para cada forma lógica junto com outra forma lógica)

Todas as rotinas foram rodadas em um processador Pentium4.2GHz com 256MB RAM.

Table 2
No-fit polygon generation times for 32 datasets of the literature

A	B	C	D	E	F	G	H
Dataset	Number of different shapes	Rotational constraints	Number of rotations per shape	Logical total number of shapes	Total number of NFPs	Total generation time (seconds)	NFPs per second
Albano180	8	180	2	16	256	0.32	800
Albano90	8	90	4	32	1024	0.71	1442
Blasz1	7	180	2	14	196	0.21	933
Blasz2	4	90	4	16	256	0.19	1347
Dagli	10	90	4	40	1600	0.93	1720
Dighe1	16	90	4	64	4096	1.28	3200
Dighe2	10	90	4	40	1600	0.62	2581
Fu	12	90	4	48	2304	0.52	4431
Jakobs1	25	90	4	100	10,000	5.57	1795
Jakobs2	25	90	4	100	10,000	5.07	1972
Mao	9	90	4	36	1296	1.41	919
Marques	8	90	4	32	1024	0.79	1296
Poly1a	15	90	4	60	3600	1.37	2628
Poly2a	15	90	4	60	3600	1.37	2628
Poly3a	15	90	4	60	3600	1.37	2628
Poly4a	15	90	4	60	3600	1.37	2628
Poly5a	15	90	4	60	3600	1.37	2628
Poly2b	30	90	4	120	14,400	7.54	1910
Poly3b	45	90	4	180	32,400	27.14	1194
Poly4b	60	90	4	240	57,600	68.45	841
Poly5b	75	90	4	300	90,000	141.90	634
Shapes	4	90	4	16	256	0.38	674
Shapes0	4	0	1	4	16	0.11	145
Shapes1	4	180	2	8	64	0.19	337
Shirts	8	180	2	16	256	0.33	776
Swim	10	180	2	20	400	6.08	66
Trousers	17	180	2	34	1156	0.73	1584
Profiles6	9	90	4	36	1296	0.86	1507
Profiles7	9	90	4	36	1296	0.58	2234
Profiles8	9	90	4	36	1296	0.56	2314
Profiles9	16	90	4	64	4096	44.30	92
Profiles10	13	0	1	13	169	0.55	307

Tabela2, mostra que a construção do NFP pode ser gerada com um número razoável de frames na maioria dos casos. Geralmente o algoritmo pode gerar cerca de 1000 NFP por segundo embora esse dado pode variar drasticamente dependendo do número de bordas (segmentos) de cada problema. Por exemplo, os dois problemas com os menores taxas para NFP/seg “Swim” e “Profiles9”, envolvem peças com muitos segmentos (média de 67) devido a aproximação dos arcos, e mais “Profiles9” possui muitas formas com buracos (Fig.22)

Os piores tempos de geração total ocorrem com “Poly4b” e “Poly5b”isso é devido ao grande numero de NFP gerados. Enquanto a maioria dos problemas não possuem casos degenerados, “Profiles6” e “Profiles9” possuem vários dos casos degenerados.

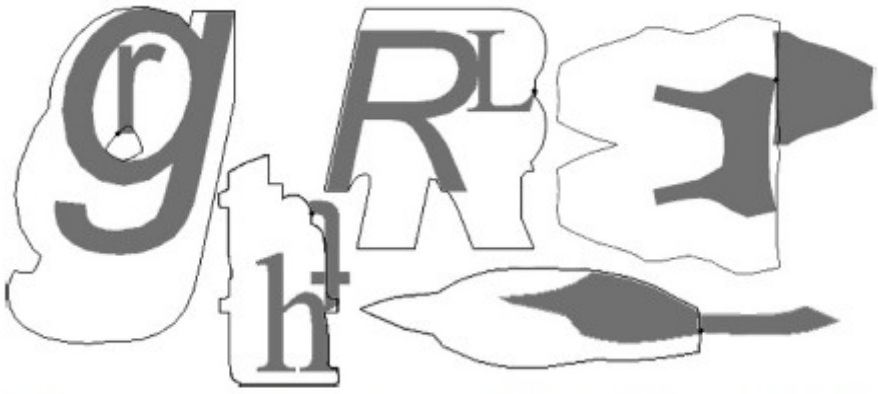


Fig. 22. A selection of pieces and no-fit polygons from the “Profiles 9” (letters) and “Swim” datasets.

Tabela3 apresenta uma comparação do tempo de geração da nossa nova abordagem com a abordagem de Bennell baseada em Minkowski (com um DEC Alpha 3000/400). Acertamos os problemas para que não houvesse rotação, para fins comparativos.

Table 3
Comparison of generation times of the Minkowski approach of Bennell et al. (2001) with the presented orbital approach on five literature datasets

Dataset (0° rotation only)	Total generation times (seconds)	
	Minkowski approach (Bennell et al., 2001) (DEC Alpha 3000/400)	Orbital approach (presented) (Pentium 4 2 GHz)
DS1	0.32	0.23
DS2	0.22	0.04
DS3 (Shirts)	0.23	0.17
DS4 (Shapes0)	0.38	0.11
DS5 (Blasz1)	0.36	0.13

Podemos ver que a nova abordagem gera NFP mais rápido em todos os casos. Apesar desta comparação ser injusta dado as diferenças no processamento computacional, ainda assim é possível afirmar que a nova abordagem pode gerar NFP rapidamente.
Temos então demonstrado que para problemas-referencia, somos capazes de gerar todos os NFP de maneira rápida e robusta.

3.6- Resumo:

Neste artigo pudemos revisar as principais técnicas que vem sendo usadas para geração de NFP e mostramos que são construções importantes para o desenvolvimento de algoritmos mais rápidos em oposição a abordagem baseada em trigonometria básica. Uma implementação completa e robusta do método orbital para NFP também foi apresentada. Essa abordagem pode lidar facilmente com casos degenerados que os métodos anteriores não conseguem resolver. Mais ainda mostramos que os tempos de execução necessário para o processo de geração do NFP em problemas referência são razoáveis. E até onde sabemos não só esta é a primeira vez que uma implementação completa de um processo de geração orbital de NFP é proposta, mas também é a primeira vez que uma abordagem foi tão rigorosamente investigada por sua robustez com os conhecidos casos degenerados e forneceu resultados computacionais em uma variedade tão grande de dados de referência existentes.

4- Problema:

O problema de corte unidimensional surge de processos industriais que precisam cortar rolos de matéria-prima em rolos de tamanhos menores visando atender a uma demanda. O objetivo deste problema é determinar a quantidade de vezes que os padrões de corte serão utilizados de forma a minimizar o desperdício de matéria-prima e o custo de *setup*.

O custo de setup ocorre quando um padrão de corte é trocado por outro. Este custo é atribuído à mão-de-obra, e à perda de produtividade enquanto as facas são posicionadas nos seus lugares e precisamente ajustadas dentro da máquina de corte.

Para facilitar a compreensão da modelagem e resolução trabalharemos com um exemplo.
Uma fábrica produz rolos-mestres de 170m de largura e a tabela abaixo traz as encomendas a serem satisfeitas, os tamanhos dos rolos e as quantidades:

tamanho(m)	quantidade(unid)
30	80
50	120
55	110

Sabemos também que são possiveis 15 padrões de corte dado o tamanho do rolo-mestre e as medidas dos rolos menores a serem cortados:

Padrões															
Tam	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
30	5	4	4	3	3	2	2	2	1	1	1	0	0	0	0
50	0	1	0	1	0	2	1	0	2	1	0	3	2	1	0
55	0	0	0	0	1	0	1	2	0	1	2	0	1	2	3

Agora definiremos as variáveis de decisão, restrições e função objetivo a fim de modelar o problema como um PL.

5-Modelagem:

Parâmetros conhecidos:

L = largura da matriz de corte (matéria-prima);
l_i = largura do item final i após processo de corte;
dem_i = demanda por itens de tamanho final i.
a_{ij} = matriz que define a quantidade de itens i produzidos ao se utilizar o padrão de corte j;
$desp_j$ = desperdício de matéria-prima atribuído ao padrão de corte j;
cs = custo de setup;
Cmp = custo por unidade de desperdício de matéria-prima.

Variáveis de decisão:

x_j = número de vezes que o padrão de corte j será utilizado;
y_j = variável binária que vale um quando o padrão de corte j é utilizado e vale zero caso contrário.
$folga_i$ = excesso de produção dos itens de tamanho final i.

Assim temos o modelo:

$$\min\left\{cmp.\sum_j desp_j x_j + cs.\sum_j y_j + M.\sum_i fol.ga_i\right\}$$

sujeito a

$$\sum_j a_{ij} x_j = dem_i + fol.ga_i \qquad (1)$$

$$x_j - Gy_j \le 0 \quad \forall j \qquad (2)$$

$$x_j \ge 0 \text{ e inteiro} \quad \forall j \qquad (3)$$

$$y_j \in \{0,1\} \quad \forall j \qquad (4)$$

$$fol.ga_i \ge 0 \quad \forall i \qquad (5)$$

Onde, M = constante muito maior que um, G = constante igual à máxima demanda, e os parâmetros *cmp* =10, *cs* =100 e *M* =10 serão sempre fixos nestes valores. *M* funciona na verdade como uma penalidade ao excesso de produção, traduzido na variável $folga_i$.

Analisando as restrições temos que: a restrição (1) garante o atendimento da demanda e permite um excesso de produção que será penalizado na função objetivo. Já a restrição (2) existe para garantir a definição da variável y_j e *G* é uma constante que irá valer o máximo valor da demanda por algum item *i*. Isto porque x_j nunca será maior ou igual a esta demanda. As outras restrições, de (3) a (5) garantem a não-negatividade das variáveis e define x_j como inteiro ey_j como variável binária.

Os códigos da implementação podem ser encontrados no Apêndice (B)

6-Resultados Computacionais:

Possuindo os dados das medidas do rolo-mestre, as larguras de corte e suas demandas, podemos gerar os padrões de corte, inserindo estes dados em nosso programa de AIMMS já modelado obtemos:

x =[0 120 0 0 0 0 0 55 0 0 0 0 0 0 0]

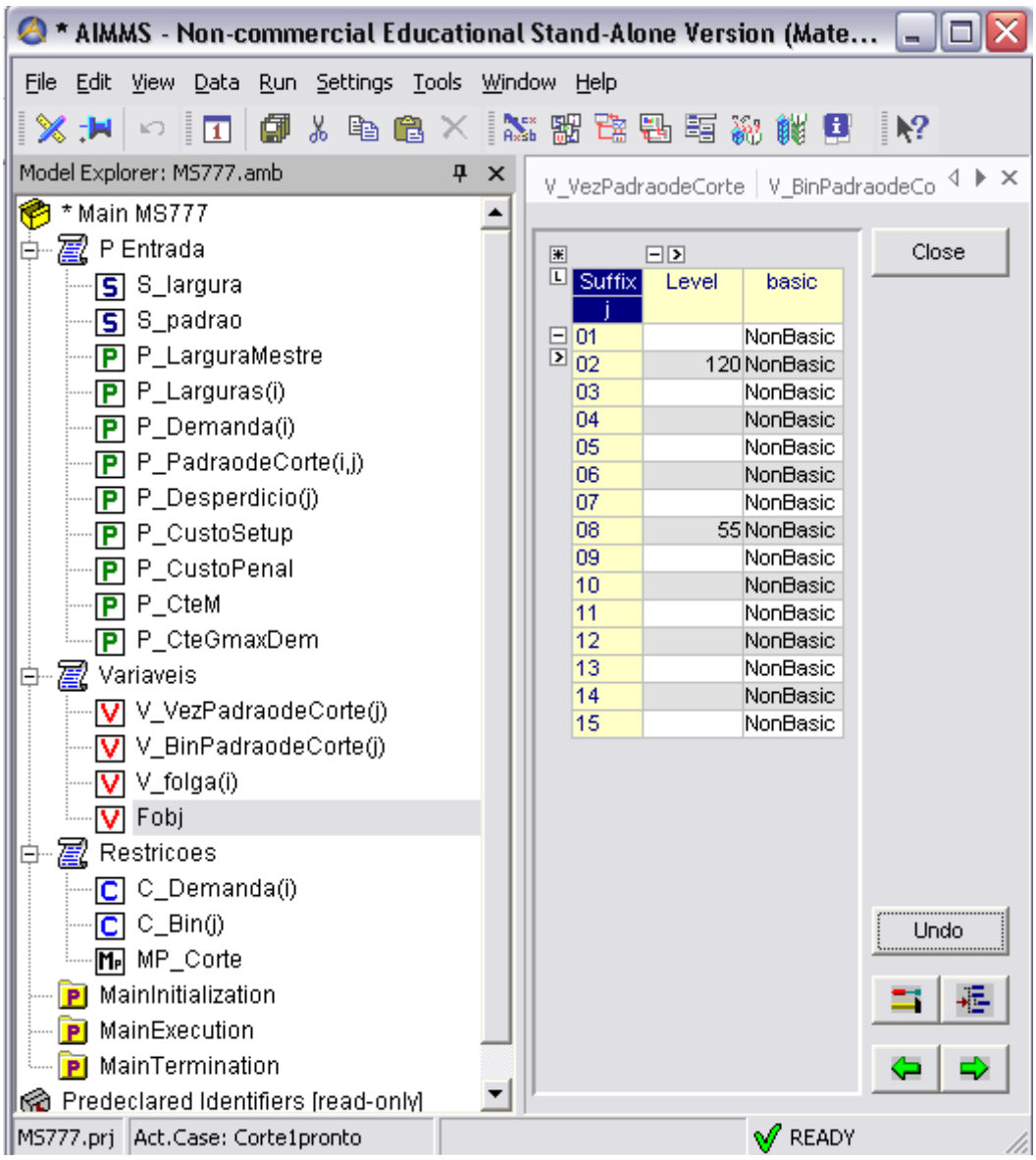
y =[0 1 0 0 0 0 0 1 0 0 0 0 0 0 0]

folga = [510 0 0]

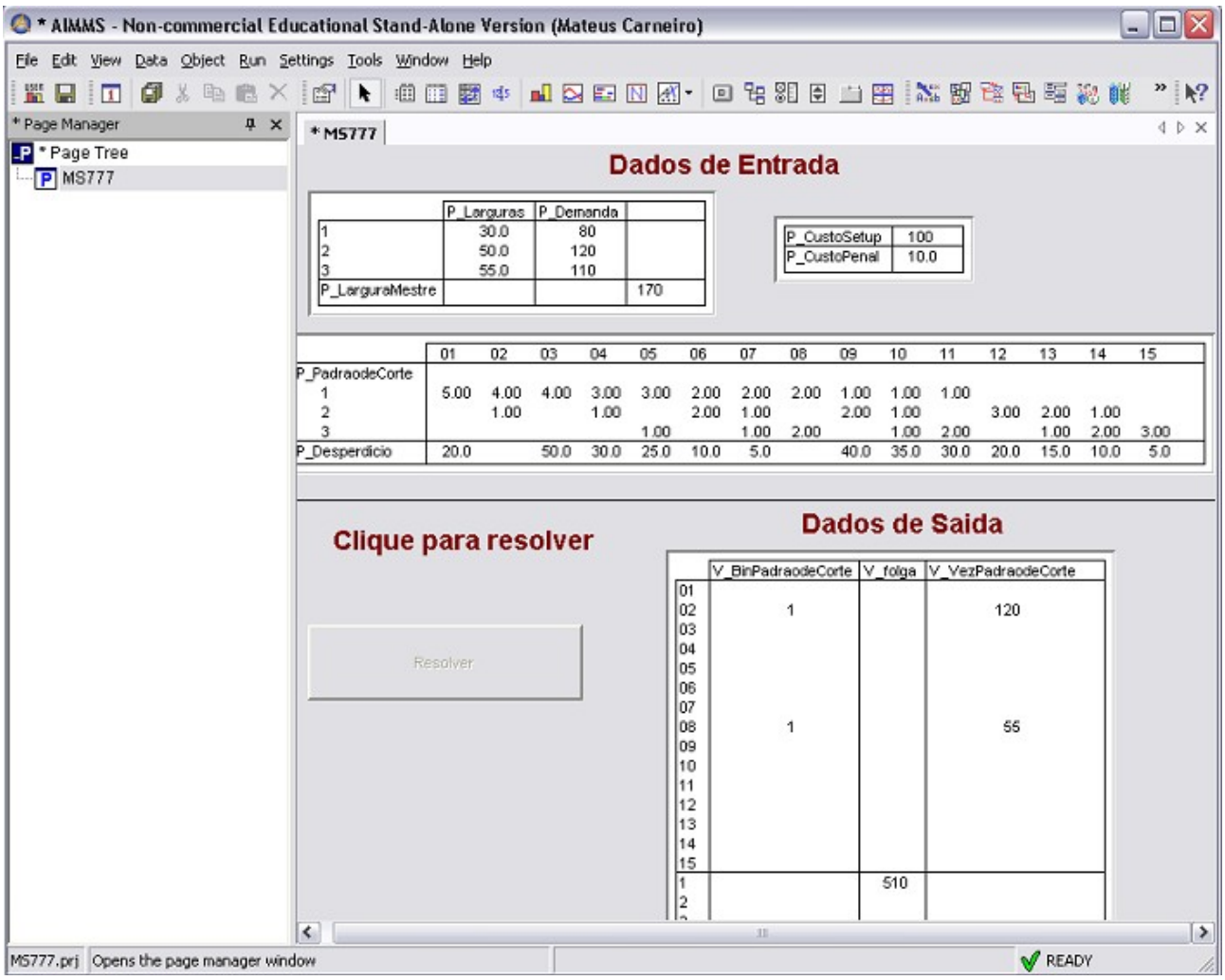
Fobj = 5300

Isto quer dizer que serão usadas 120 vezes o padrão de corte 2 e 55 vezes o padrão de corte 8 que produzirão um excesso de produção de 510 unidades de tamanho 30, estes são os valores obtidos a fim de satisfazer as demandas obtendo o menor custo e o menor despesdício. E o custo total desta produção levando em conta o custo de setup e depenalização pelo excesso de produção é 5300 .

Em seguida temos uma foto da implementação em AIMMS e da página de apresentação:



Suffix j	Level	basic
01		NonBasic
02	120	NonBasic
03		NonBasic
04		NonBasic
05		NonBasic
06		NonBasic
07		NonBasic
08	55	NonBasic
09		NonBasic
10		NonBasic
11		NonBasic
12		NonBasic
13		NonBasic
14		NonBasic
15		NonBasic



7-Discussão:

Nesta secção temos uma breve discussão sobre a geração dos padrões de corte e sobre uma abordagem de um problema de corte bidimenssional.

A maioria das técnicas utilizadas para resolução de Problemas de Corte unidimensional envolve algoritmos baseados em Programação Linear, porém este problema se torna complicado quando existem diferentes tamanhos de rolos de matéria-prima ou diferentes tamanhos de rolos finais. Por exemplo, um rolo de matéria-prima de 200” e demanda por 40 tipos de rolos finais variando de 20” até 80” geram uma quantidade de padrões de corte que excedem a grandeza dos milhões, o que não é viável. Logo métodos alternativos são estudados. O método estudado neste projeto é o que foi apresentado no artigo “*Pattern generating procedure for the cutting stock problem”*. (Saad M.A.Suliman)[2]. A idéia do artigo é gerar uma árvore de possibilidades onde cada nível representa o tamanho do item final, em ordem decrescente, e cada nó a largura da matriz de corte que falta ser cortada. A figura 1 fornece o exemplo de uma matriz de corte de 130 cm e itens finais de 50, 40 e 30 cm. O último nível é o desperdício de matéria-prima de cada padrão. A árvore gerou 8 padrões de corte diferentes que pode ser visualizados na tabela 1

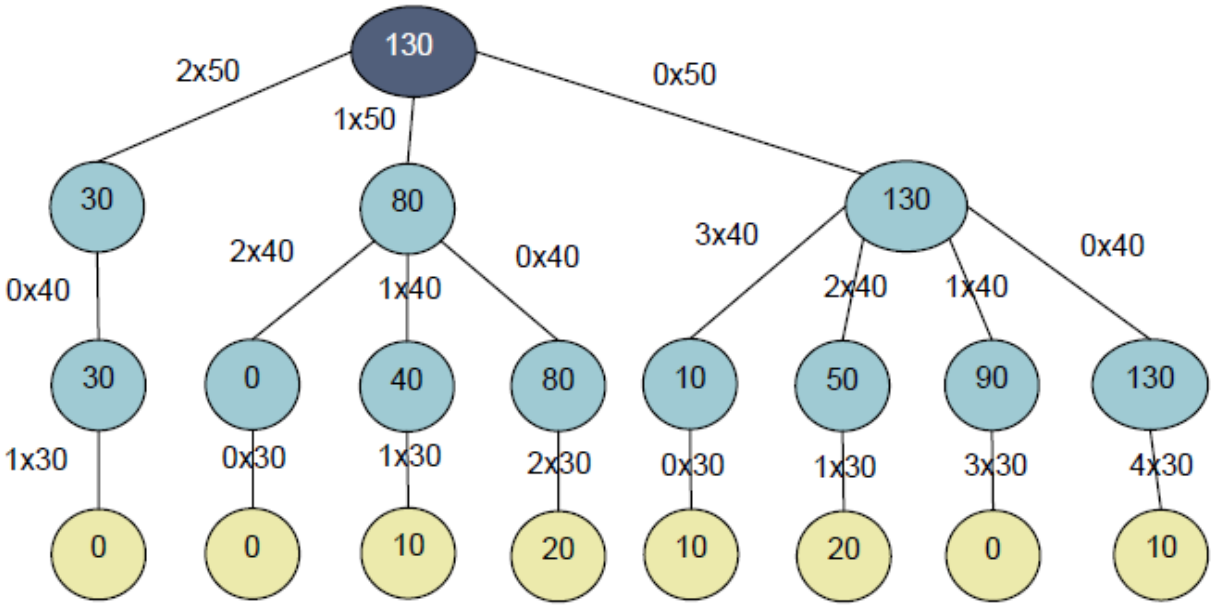


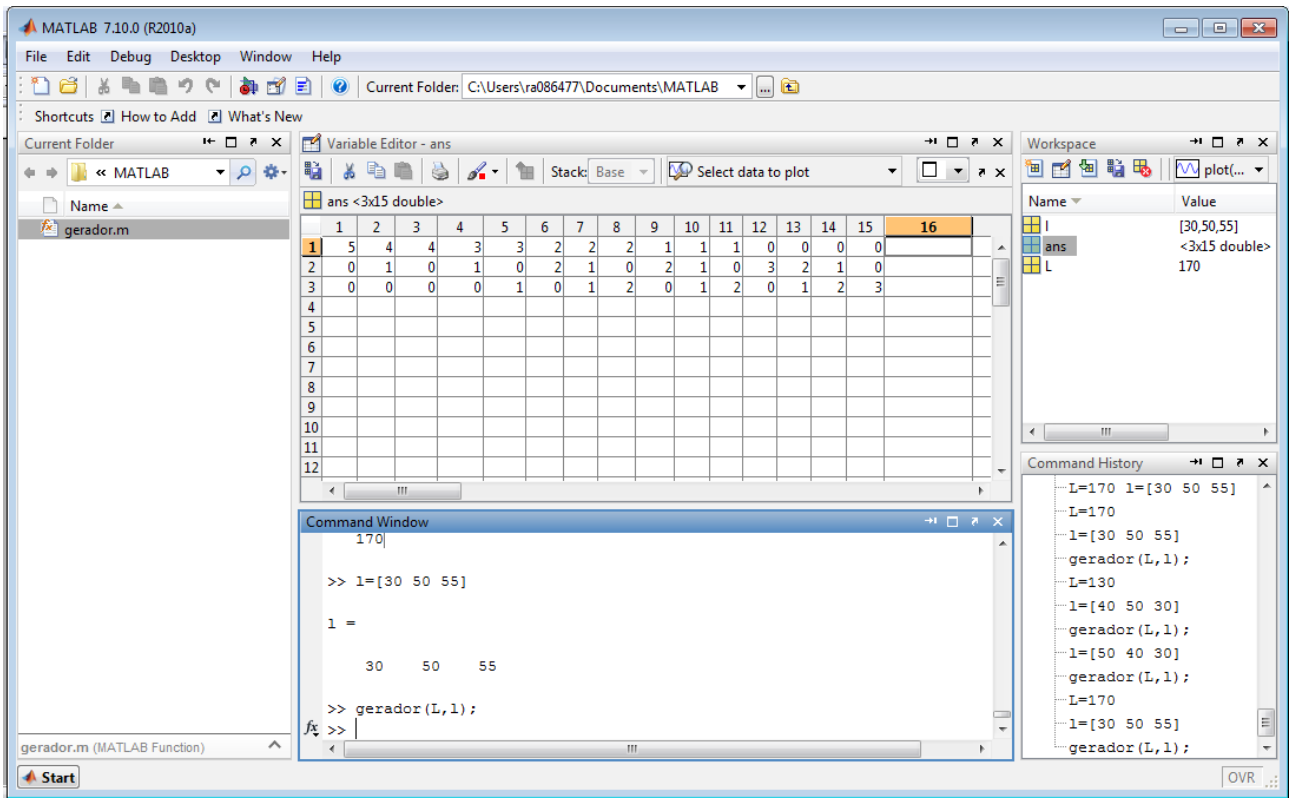
Figura 1: Árvore de possibilidades para matriz de 130cm e itens de 50, 40 e 30 cm.

Tabela 1: Padrões de corte gerados

Padrões	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
50 cm	2	1	1	1	0	0	0	0
40 cm	0	2	1	0	3	2	1	0
30 cm	1	0	1	2	0	1	3	4
Desperdício	**0**	**0**	**10**	**20**	**10**	**20**	**0**	**10**

Note que cada padrão de corte será uma coluna da matriz definida anteriormente como a quantidade de itens finais *i* produzidos ao se utilizar o padrão de corte *j*. Neste exemplo, . Foi criada em Matlab a *function* **gerador** para a geração dos padrões de corte possíveis dados o tamanho da matriz de corte e o vetor com a largura de cada item em ordem decrescente. Entretanto no algoritmo estamos limitando a geração de padrões até 20.000 tipos diferentes. Isso foi necessário porque para que em problemas maiores o tempo de processamento não fosse muito longo. Assim, a matriz a_{ij} *a* possuirá no máximo 20.000 colunas. A quantidade de linhas será determinada pela quantidade de itens finais demandados. O código computacional encontra-se no Apêndice (C).

A imagem abaixo é uma foto da implementação para geração dos padrões do problema sugerido.



Para problemas bidimensionais, como já vimos temos um método bastante eficaz que é a geração de *No-Fit Polygons*, utilizado para verificar se os polígonos estão sobrepostos, tocando-se ou separados, capaz de resolver de problemas simples de encaixe de formas triviais até as formas mais complexas, com muitas irregularidades e/ou buracos. Assim com a geração do NFP somos capazes de determinar a disposição das pecas, moldes no rolo, placa de onde devem ser cortados, sendo assim de certa forma seu padrão de corte.

Agora suponha que temos o seguinte problema de corte bidimensional. Desejamos posicionar um número de objetos em um rolo-mestre, de modo que não exista sobreposição entre os objetos e que o comprimento do rolo seja mínimo.

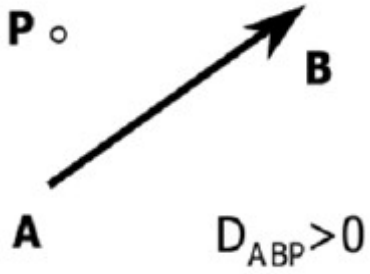
Antes de modelarmos o problema supomos algumas condições, todos os objetos são de geometria convexa, e não são permitidas rotações. Assim temos que nossos dados conhecidos, de entrada, do problema são a largura do rolo-mestre(w), o comprimento maximo do rolo(L), a geometria dos nossos objetos convexos e também o NFP gerado para cada par de objetos. Nossas variáveis serão o comprimento (l) do rolo e as translaões dos polígonos ($T(x,y)$), cada objeto possui um ponto de referência que é um vértice do polígono, a translação é dada em relação ao ponto de referência com a origem do rolo utilizado, neste caso $(0,0)$. A saída portanto será uma lista de pontos (x,y), um para cada objeto, que indicam sua translação sobre a área de corte.

Nosso problema possui apenas duas restriçõess a satisfazer: 1-) todos os objetos devem estar completamente contidos sobre a área de corte disponível e 2-) que não haja sobreposição. A restricao 2 é construida a partir da ideia de NFP, na qual objeto j não sobrepõe o objeto i se seu ponto de referência estiver fora ou sobre o contorno do NFP_{ij}. Isto é equivalente a definir que o ponto de referência deve estar sobre o lado esquerdo de pelo menos uma aresta do NFP_{ij}, assumindo-se uma orientação no sentido horário. Note que os polígonos devem estar dipostos de forma que não se sobreponham, porem é permitido que se toquem. Relembrando o que vimos no artigo, está restrição é imposta através da função – D de Konopasek, que fornece a posição relativa de um ponto P em

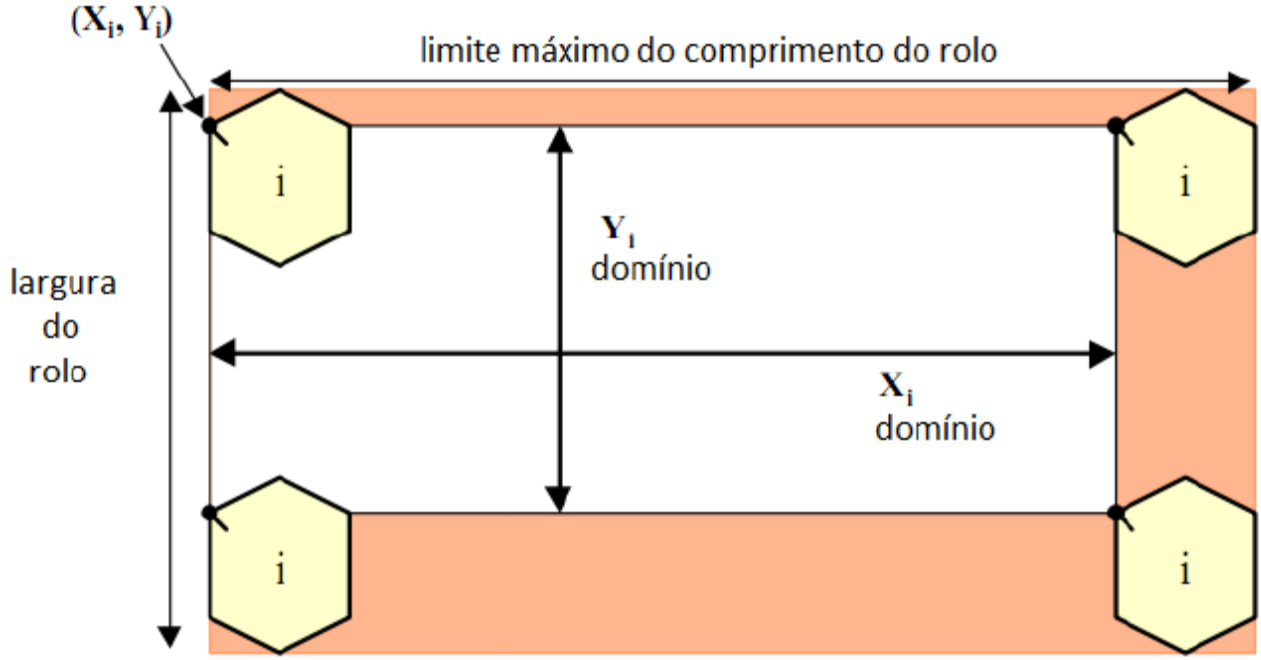
relação a uma aresta orientada *AB*. Ela baseia-se na equação da distância entre um ponto e uma linha reta:

$$D_{ABP} = ((X_A - X_B)(Y_A - Y_P) - (Y_A - Y_B)(X_A - X_P))$$

A linha é considerada infinita e considera um suporte para a aresta do polígono. Sua interpretação \é tal que: Se $D_{ABP} > 0$, então o ponto *P* está do lado esquerdo da linha de suporte da aresta *AB*; se $D_{ABP} < 0$, então o ponto *P* está do lado direito da linha de suporte da aresta *AB*; Se $D_{ABP} = 0$, então o ponto *P* está sobre a linha de suporte da aresta *AB*. Esta interpretação assume que a origem do sistema de coordenadas, onde as coordenadas (X_i, Y_i) estão definidas, seja o canto inferior esquerdo, ou seja, a coordenada *x* aumenta para a direita e a coordenada *y* aumenta para cima. A figura abaixo mostra um exemplo onde o ponto *P* se localiza a esquerda da aresta *AB* .



Tratando agora a restrição 1), uma das maneiras encontradas para se modelar esta restrição foi reduzir o domínio inicial das variáveis de translação dos polígonos de forma que qualquer valor atribuído represente uma posição válida. A Fig.A mostra como estes domínios podem ser reduzidos de acordo com a geometria de cada objeto. A área mais clara representa as posições válidas para o objeto *i*, assumindo-se o ponto *(xi, yi)* como referência. Note que há uma redução do domínio inicial do objeto *i*, conseqüentemente do espaço de busca.



Finalmente temos nossa função objetivo: devemos posicionar todos os objetos de forma que a área necessária para realizar o corte destes objetos seja minimizada economizando matéria-prima, consequentemente minimizando a área de corte o desperdício é menor. Como a largura *w* da bobina já é fornecida, basta então minimizar o comprimento *l*.

Logo a função objetivo é dado por: Min *I*

Agora temos nosso problema de corte bidimensional modelado, porém para solucioná-lo ainda são necessárias outras técnicas de abordagem, tal como uma estratégia de buscas para determinar dentre os pontos gerados quais formam a solução ótima. Além disso dominar algum tipo de programa que permita visualizar o resultado do processo de encaixe e conhecimento das rotinas geométricas é bastante útil.

8 - Conclusões:

Aqui pudemos conhecer vários métodos de se abordar o problema de corte para formas irregulares através do artigo *Complete and Robust No-Fit Polygon Generation for the irregular Stock Cutting Problem[1]* e observar o quão este problema se torna difícil a medida que as formas se tornam mais complexas. Importante também notar a importância da abordagem através dos No-Fit polygon e como este método consegue lidar com as dificuldades que várias outras técnicas não conseguiam, ao se manipular formas bastante irregulares. Por fim a criação de NFP é a melhor maneira de se realizar testes de sobreposição, embora seja necessário vasto conhecimento de rotinas geométricas.

Trabalhamos também na implementação em AIMMS de um problema básico de corte unidimensional onde lidamos com problema da criação de padrões de cortes dados as medidas da matriz de corte principal e as medidas das sub matrizes. Este problema foi abordado pela visão do artigo *Pattern generating procedure for the cutting stock problem*[2], e uma rotina em Matlab. Apresentamos também brevemente como se bordar um problema de corte em duas dimensões e como ao se adicionar uma dimensão o problema torna-se muito mais complexo sendo necessário ainda mais conhecimento das abordagens de corte e necessidade de se mesclar varias técnicas afim de obter a melhor solução possível em um tempo viável, ponto bastante importante já que em problemas mais complexos a maior dificuldade é sempre encontrada no tempo de computação.

Por fim temos claramente a importância de se abordar problemas de corte. São problemas, tanto o unidimensional quando o bidimensional, que encontramos em diversas áreas da indústria, papel, metal, têxtil, entre outras. Enfim qualquer problema que se tenha que encaixar moldes, determinar o corte de partes menores a partir de uma "fonte", podem ser resolvidos adaptando-os a problemas de corte. Não é atoa que a pesquisa e desenvolvimento de algoritmos de corte e encaixe cada vez melhores e mais rápidos é de grande importância na pesquisa operacional.

9-Apêndice

(A)

1. Código para geração de *No-Fit Polygon*:

```
Input : Polygons A and B
Pt_A(ymin) = point of minimum * value of polygon A
Pt_B(ymax) = point of maximum * value of polygon B
IFP = initial feasible position
Bool bStartPointAvailable = true;
Point NFPLoopStartRefPoint;
Point PolygonB_RefPoint;
Array[Line[]] nfpEdges; // an array of arrays of lines to store NFP edges
Int loopCount = 0; // counter for number of loops in NFP
```

```
Begin

Place polygons in IFP using translation Pt_A(ymin) – Pt_B(ymax);
NFPLoopStartRefPoint = Pt_B(ymax);
PolygonB_RefPoint = Pt_B(ymax);


While(bStartPointAvailable)
 {
  bStartPointAvailable = false;
  // find touching points & segments touching those points,
     generate touching structures
  Touchers[] toucherStructures = FindTouchers (A, B);


  // Eliminate non-feasible touchers, ones that cause immediate
     intersection
  Touchers[] feasibleTouchers = CanMove (A, B, toucherStructures);


  // Trim feasible translations against polygon A and B
  Touchers[] trimmedTouchers = Trim(feasibleTouchers, A, B);


  // Sort trimmed translations by length
  Touchers[] lengthSortedTouchers = Sort(trimmedTouchers);


  // Translate polygon B along longest feasible translation vector
  B. Translate(lengthSortedTouchers[0].Translation);


  // Add translation to nfpEdges & mark traversed edge on static
  nfpEdges[loopCount].Add(lengthSortedTranslations[0].Translation;
  A. MarkEdge(lengthSortedTranslations[0].StaticEdgeID);


  If(NFPLoopStartRefPoint == PolygonB_RefPoint) // completed an NFP loop
    {
     Point nextStartPoint;
     // find next feasible start point – reset PolygonB_RefPoint to
     relevant point
     bStartPointAvailable = FindNextStartPoint(A, B, & nextStartPoint, &
     PolygonB_RefPoint);


     if(bStartPointAvailable)
     {
      // Translate polygon B to nextStartPoint
      B.Translate(PolygonB_RefPoint – nextStartPoint);
      NFPLoopStartRefPoint = nextStartPoint;
      loopCount++;
     }
    }
  else
  {
   bStartPointAvailable = true; // allow edge traversal to continue
  }
}
NFP_{AB} = Complete(nfpEdges); // Reconstitute NFP from nfpEdges
End
```

2. Pseudo código para encontrar ponto inicial factível:

```
Input : PolygonA, PolygonB, reference to Point nextStartPoint, reference
to Point PolygonB_RefPoint
Int A_EdgeCount = PolygonA.EdgeCount;
Int B_EdgeCount = PolygonB.EdgeCount;
Edge StaticEdge;
Edge movingEdge;

Begin

for(int i = 0; i < A_EdgeCount; i++)
 {
  if(PolygonA.IsEdgeMarked(i))
    continue;
  else
    staticEdge = PolygonA.GetEdge(i);
  for(int j = 0; j < B_EdgeCount; j++)
   {
    movingEdge = PolygonB.GetEdge(j);


    // translate the PolygonB so that movingEdge start is on the start of
     the static edge
    PolygonB.Translate(movingEdge.Start – staticEdge.Start);
    Bool bFinishedEdge = false;
    Bool bIntersects = PolygonB.IntersectsWith(PolygonA);


    while(bIntersects AND !FinishedEdge)
     {
     // Edge slide until not intersecting or end of staticEdge reached
     Toucher currentToucher = MakeToucher(staticEdge.Start);
     Toucher trimmedToucher = Trim(currentToucher, PolygonA, PolygonB);
     PolygonB.Translate(trimmedToucher.Translation);


     bIntersects = PolygonB.IntersectsWith(PolygonA);


     if(bIntersects)
      {
       If(movingEdge.Start == staticEdge.End)
       bFinishedEdge = true;
      }
     }
    // mark the traversed edge as seen (whether edge start point found or
     not)
    staticEdge.Mark(true);


    if(!bIntersects)
     {
      // set the references to the points passed in to be the
       nextStartPoint
      // and return true;
      nextStartPoint = movingEdge.Start;
      PolygonB_RefPoint = movingEdge.Start;
```

```
      return true;
     }
    }
  }
return false;
End
```

-(B)

Implementação em AIMMS:

-Parâmetros de Entrada:

```
P_LarguraMestre
P_Larguras(i)
P_PadaodeCorte(i,j)
P_Demanda(i)
P_CustoSetup
P_CustoPenal
P_Desperdicio(j)
```

-Variáveis:

```
V_VezPadraodeCorte(j)
V_BinPadraodeCorte(j)
V_folga(i)
```

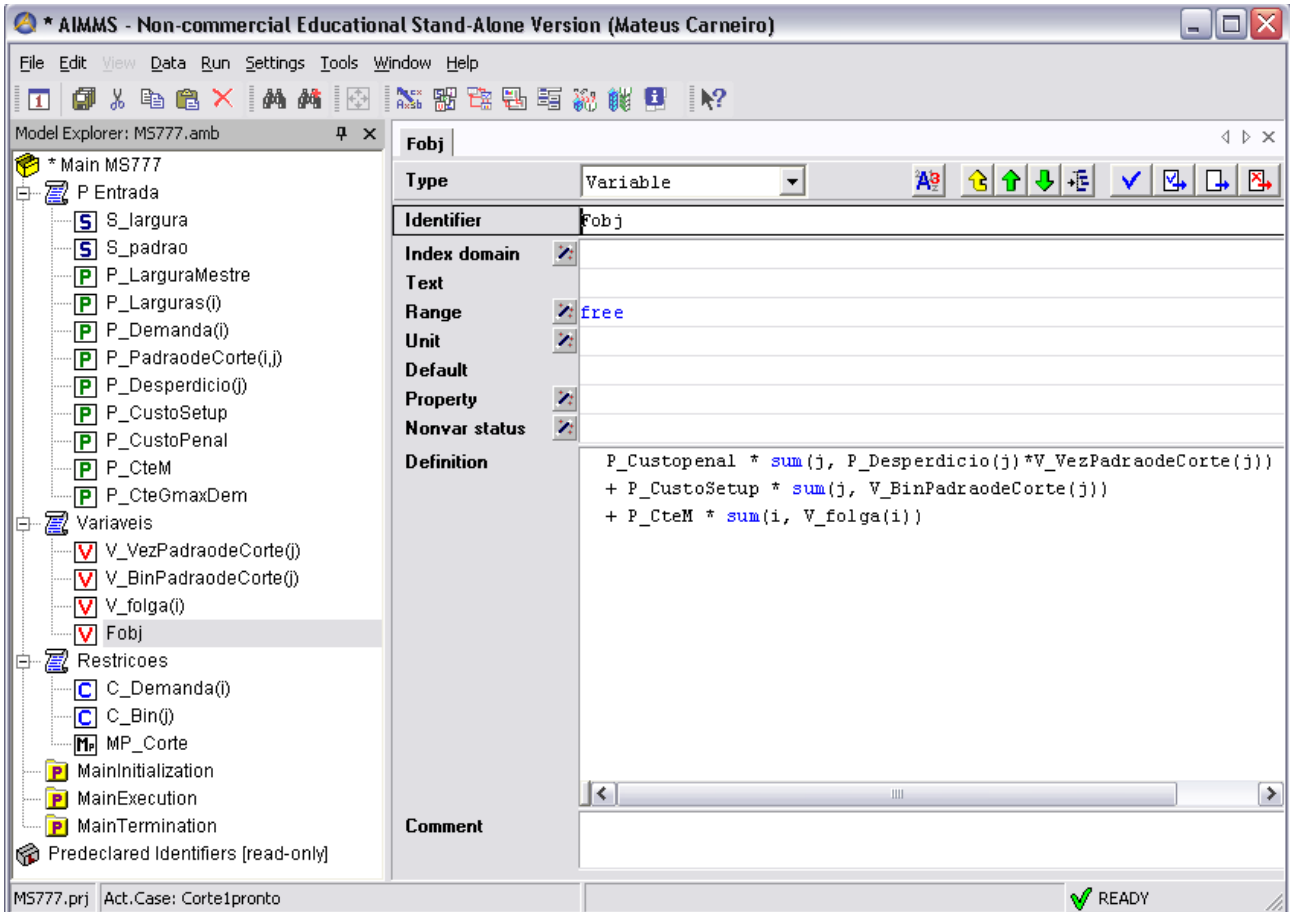
-Função Objetivo:

```
P_LarguraMestre - sum(i, P_PadraodeCorte(i,j)*P_Larguras(i))
```

-Restrições:

```
C_Demanda(i)
     sum(j, P_PadraodeCorte(i,j)*V_VezPadraodeCorte(j))
     = P_Demanda(i) + V_folga(i)
C_Bin(j)
     V_VezPadraodeCorte(j) - P_CteGmaxDem *
     V_BinPadraodeCorte(j) <= 0
```

Temos aqui mais uma foto da implementação:



(C)

Geração de padrões de corte:

```
function [a,desp] = gerador (L,l)
%
%Vetor l: tamanhos dos itens onde l1 > l2 > l3 > ... > lm
%Peenchimento da 1a. coluna de A
%
[m] = length(l);
j = 1;
a(1,j) = floor(L/l(1));
sum = a(1,j)*l(1);
for i = 2:m
    a(i,j) = floor( (L - sum)/l(i) );
    sum = sum + a(i,j)*l(i);
end
desp(j) = L - sum;
%
i = m - 1;
%
while (i > 0 & j <= 20000)
    if ( a(i,j) == 0 )
        i = i - 1;
    else
        j = j + 1;
        zum = 0;
         for z = 1 : (i-1)
            a(z,j) = a(z,j-1);
```

```
            zum = zum + a(z,j)*l(z);
        end
        a(i,j) = a(i,j-1) - 1;
        zum = zum + a(i,j)*l(i);
        %
        for k = (i+1) : m
            a(k,j) = floor ((L-zum)/l(k));
            zum = zum + a(k,j)*l(k);
        end
    desp(j) = L - zum;
    i = m - 1;
    end
end
```

10 - Referências:

[1] Complete and Robust No-Fit Polygon Generation for the Irregular Stock Cutting Problem.
[2] Pattern generating procedure for the cutting stock problem.