Escalonamento de Atividades de Desenvolvimento de Poços de Petróleo: GRASP

Romulo Albuquerque Pereira

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Escalonamento de Atividades de Desenvolvimento de Poços de Petróleo: GRASP

Romulo Albuquerque Pereira

Novembro de 2005

Banca Examinadora:

- Prof. Dr. Arnaldo Vieira Moura (Orientador)
- Prof. Dr. Ricardo Dahab
  Instituto de Computação – UNICAMP
- Prof. Dr. Vinícius Amaral Armentano
  Faculdade de Engenharia Elétrica e de Computação – UNICAMP
- Prof. Dr. João Meidanis (Suplente)
  Instituto de Computação – UNICAMP

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 16 de dezembro de 2005, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Vinícius Amaral Armentano
FEEC / UNICAMP

Prof. Dr. Ricardo Dahab
IC / UNICAMP

Prof. Dr. Arnaldo Vieira Moura
IC / UNICAMP

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Bibliotecário: Maria Júlia Milani Rodrigues - CRB8a / 2116

Pereira, Romulo Albuquerque

P414e Escalonamento de atividades de desenvolvimento de poços de petróleo: GRASP / Romulo Albuquerque Pereira -- Campinas, [S.P. :s.n.], 2005.

Orientadores : Arnaldo Vieira Moura; Cid Carvalho de Souza

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Otimização combinatória. 2. Heurística. 3. Pesquisa operacional. 4. Poços de petróleo - Perfuração. I. Moura, Arnaldo Vieira. II. Souza, Cid Carvalho de. II. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Título em inglês: Scheduling of development activities of oil wells: GRASP

Palavras-chave em inglês (Keywords): 1. Combinatorial optimization. 2. Heuristic. 3. Operational research. 4. Oil wells drilling.

Área de concentração: Otimização Combinatória

Titulação: Mestre em Ciência da Computação

Banca examinadora: Prof. Dr. Arnaldo Vieira Moura (IC-UNICAMP)
Prof. Dr. Ricardo Dahab (IC-UNICAMP)
Prof. Dr. Vinicius Amaral Armentano (FEEC-UNICAMP)
Prof. Dr. João Meidanis (IC-UNICAMP)

Data da defesa: 16/12/2005

Escalonamento de Atividades de Desenvolvimento de Poços de Petróleo: GRASP

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Romulo Albuquerque Pereira e aprovada pela Banca Examinadora.

Campinas, 16 de novembro de 2005.

Prof. Dr. Arnaldo Vieira Moura (Orientador)

Prof. Dr. Cid Carvalho de Souza
(Co-orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Resumo

Este trabalho de mestrado procurou estudar e resolver um problema real de escalonamento das atividades de desenvolvimento de poços de petróleo em alto mar. Uma versão mais simples deste mesmo problema foi provada ser NP-difícil. Nosso estudo se concentrou no problema real enfrentado pela Petrobras, com todas suas características e nuances.

Antes que locais promissores de bacias petrolíferas sejam efetivamente desenvolvidos em poços de petróleo produtivos, é necessário realizar diversas atividades de perfuração, completação e interligação nesses locais. O escalonamento dessas atividades deve satisfazer várias restrições conflitantes e buscar a maximização da produção de petróleo em um dado horizonte de tempo. O problema foi atacado em duas etapas: uma sem considerar o deslocamento de recursos e outra considerando-os. Para tal, adotamos a estratégia *Greedy Randomized Adaptive Search Procedure (GRASP)* e incorporamos várias técnicas específicas para obter melhores desempenho e qualidade da solução final. Os resultados são comparados com outros produzidos por uma ferramenta computacional baseada em Programação por Restrições (PR). Esta última, já em uso e bem aceita na empresa, foi desenvolvida pela Petrobras. Resultados comparativos realizados em instâncias reais indicam que a implementação GRASP supera a ferramenta de PR produzindo soluções com expressivos aumentos de produção.

Abstract

This dissertation aimed at studying and solving a real world scheduling problem. We deal with the scheduling of offshore oil well development activities. A simpler version of this same problem was proved to be in NP-hard. Our approach treats this problem as faced by Petrobras, with all its characteristics and details.

Before promising locations at petroliferous basins become productive oil wells, it is often necessary to complete activities of drilling, completion and interconnection at these locations. The scheduling of such activities must satisfy several conflicting constraints and aim at the maximization of oil production. The problem was solved in two parts: one without considering resource displacements and other taking into account such displacements. For such, we used a Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic and used several techniques and variants in order to obtain more efficiency and produce better solutions. The results are compared with schedules produced by a well-accepted constraint programming implementation. Computational experience on real instances indicates that the GRASP implementation is competitive, outperforming the constraint programming implementation.

Agradecimentos

Agradeço...

À Deus, meu Pai. Por me encorajar, me iluminar e me inspirar a sempre fazer o que é bom. Por ser um excelente oleiro. Por me amar e por me dar tantos e muitos outros motivos para Lhe agradecer.

Aos meus pais, por terem me ensinado a aprender. Por terem me dado o gosto pela leitura e, principalmente, pela escrita. Por me amarem e serem tão especiais!

À minha família, por terem acreditado em mim e por sempre orarem por mim. Sou muito grato a todos!

Aos meus orientadores, por me iniciarem no mundo da pesquisa e por trabalharem duro junto comigo. Por me mostrarem que na seriedade de um trabalho, pode-se se divertir e contar com excelentes amigos.

À Josy, por me deslumbrar e deixar um escritor sem palavras...

Ao República Futebol Clube: Vinícius Pato, Vinícius Beiço, Ulisses Monge e Harry Umbrella por momentos inesquecíveis de grande amizade.

À Petrobras, por me empregar como terceirizado e por nos fornecer os dados para análise. Ao Fernando, por ser o "chefe" que todo empregado gostaria de ter. E aos meus colegas de trabalho, pela amizade e companheirismo.

Por que?
Por que são tão loucas as coisas loucas
da vida?
E por que são tão poucas?
Porque são não ocas
Delas cria-se um universo
Fazem-se esses versos
Os fazem belos
Elos ao eterno?
Sim.

Romulo A. Pereira
"Muitos, que Digam Muito"
A ser lançado

Conteúdo

Lista de Tabelas

Capítulo 4

Capítulo 6

Capítulo 7

Lista de Figuras

Capítulo 2

Capítulo 3

Capítulo 5

Capítulo 6

Capítulo 7

Capítulo 1

Introdução

A pesquisa científica sobre problemas de escalonamento (*scheduling*) é intensa e tem elevado teor prático, existindo desde a Segunda Guerra Mundial. Notadamente, quase toda empresa de certo porte, em que a organização, redução de custos e produtividade são indispensáveis para competir no mercado, já aplica técnicas da pesquisa operacional. Ocorre, porém, que a grande maioria destes problemas são comprovadamente classificados como NP-difíceis (Garey & Johnson (1979)). Isso praticamente exclui algoritmos determinísticos eficientes para solucionar esses problemas, especialmente para instâncias de entrada suficientemente grandes. Na ausência de algoritmos polinomiais, uma alternativa promissora é o uso de heurísticas eficazes que possam auxiliar no processo de busca de boas soluções. Em vista disso, e dadas as especificidades de cada problema, é raro encontrar-se ferramentas comerciais acabadas que atendam às necessidades de cada situação.

Petróleo e gás são combustíveis fósseis de larga utilização na sociedade atual, sendo aproveitados na fabricação de plásticos, tintas, querosene, combustível veicular e de aviação, gás de cozinha e em muitas outras aplicações. Boa parte desses combustíveis fósseis é hoje extraída de bacias oceânicas, como a de Marlim, no Estado do Rio de Janeiro. A variação na oferta e nos preços desses combustíveis, aliada à necessidade da redução dos seus custos de produção, com manutenção da produtividade, torna cada vez mais importante a utilização de mecanismos que permitam a organização racional das atividades de produção de um campo petrolífero (Hasle et al. (1997)). Algumas das atividades inerentes ao processo de exploração de petróleo possuem um custo muito elevado, principalmente devido ao aluguel de equipamentos de grande porte, tais como barcos e sondas.

A Petrobras é uma das empresas pioneiras e líderes na exploração de petróleo em águas profundas, sendo também uma das 20 maiores empresas petrolíferas do planeta. Ela explora diversas bacias petrolíferas, cada uma com centenas de locais promissores onde poços de petróleo podem ser colocados em produção. Entretanto, esses locais precisam ser desenvolvidos antes de se tornarem produtivos. Para realizar as atividades envolvidas no

desenvolvimento de um poço de petróleo, a Petrobras utiliza recursos — sondas e navios — alguns alugados, outros de sua propriedade. Esses recursos são limitados e de alto custo, seja em sua aquisição ou no preço de aluguel, e, dessa forma, devem ser utilizados eficientemente.

Ciente disso, a empresa resolveu utilizar ferramentas computacionais que pudessem melhor escalonar suas atividades realizadas na Bacia Petrolífera de Campos. Além do escalonamento das atividades, ela também deseja que o mecanismo resolvedor considere o deslocamento dos recursos. Num primeiro passo, a empresa procurou por programas comerciais que solucionassem seu problema específico. Porém, logo verificou que tais programas não estão disponíveis no mercado. Assim, a Petrobras expôs seu problema à Unicamp e sugeriu uma parceria de pesquisa sobre algoritmos e heurísticas que pudessem auxiliar na busca de uma solução satisfatória (Moura & de Souza (2000)). Este projeto se insere no escopo dessa cooperação. A empresa procurou obter, com o desenvolvimento deste trabalho, além de maior produtividade e outros benefícios, uma boa redução de custos na operação de seus equipamentos.

1.1 Organização do Texto

O texto dessa dissertação encontra-se dividido em quatro partes principais:

A Parte I traz uma descrição detalhada do problema tratado. Apresentamos todas as restrições operacionais e de engenharia envolvidas no escalonamento, e mostramos um breve histórico de como esse problema foi resolvido.

A Parte II apresenta, resumidamente, os fundamentos teóricos necessários para um entendimento adequado do trabalho descrito nesta dissertação. O Capítulo 3 expõe as idéias básicas a respeito da metaheurística GRASP. O Capítulo 4 contém uma introdução à metodologia de Programação por Restrições.

Na Parte III, estão incluídos uma descrição da ferramenta ORCA (Capítulo 5) e dois artigos resultantes deste trabalho de mestrado. Os artigos estão escritos em inglês. A cada um dos artigos estão anexadas uma nova introdução e uma nova conclusão estendidas, ambas em português. O Capítulo 6 discute a solução do problema de escalonamento no desenvolvimento de poços de petróleo sem deslocamento de recursos e o Capítulo 7 trata do mesmo problema, mas com deslocamento de recursos.

Por fim, a Parte IV expõe as conclusões finais alcançadas com este trabalho e enumera algumas possíveis direções de pesquisa que poderiam ser tomadas no sentido de dar continuidade a este estudo.

Parte I

Descrição do Problema

Capítulo 2

O Escalonamento no Desenvolvimento de Poços de Petróleo

Quando um ponto em alto mar é considerado um promissor poço de petróleo, sondas são enviadas ao local para realizar as devidas operações de *perfuração*. O desenvolvimento de um poço, desde a determinação da sua localização até ser colocado em produção, envolve várias etapas.

Após perfurado, inicia-se o processo de preparação do poço para a extração de petróleo (*completação*). Em primeiro lugar, é instalada a "Árvore de Natal Molhada (ANM)" [1] sobre a boca do poço para que a matéria-prima não extravase para o mar. Posteriormente, um navio LSV[2] leva uma tubulação da qual uma extremidade é conectada à ANM e a outra extremidade é encaixada num *manifold*[3], ou então sobe direto para a superfície, sendo conectada a uma plataforma. Esse estágio é conhecido como *interligação*. O *manifold* é instalado por uma sonda, ou por uma BGL[4], e seu uso evita que cada poço necessite de tubulações exclusivas que o conecte desde o fundo do mar até à superfície. Assim, as mangueiras de vários poços relativamente próximos podem se interligar ao *manifold* e, deste, uma única tubulação sobe até a superfície. Na ausência de *manifolds*, e com o metro de tubulação tendo alto valor, o processo todo teria valor excessivamente elevado.

Completado o processo de interligação, parte-se para a extração do petróleo propriamente dita. Para tal, são colocadas bases de captura de petróleo na superfície do mar, as chamadas UEPs (Unidade Estacionária de Produção), onde se armazenará o produto até

[1] Complexa estrutura metálica com tubulações onde se encaixam válvulas para extração do petróleo. Vide Figura 2.1.

[2] Navio de apoio.

[3] Um *manifold* é uma estrutura para a junção de tubulações no fundo do mar.

[4] Barcaça guindaste.

Figura 2.1: Árvore de Natal Molhada.

que navios venham recolhê-lo e levá-lo para terra. Se a vazão de petróleo for muito elevada, pode-se optar pela instalação local de uma plataforma petrolífera.

O problema de *Escalonamento no Desenvolvimento de Poços de petróleo em alto mar* (EDP) pode ser definido como: dados um conjunto de poços, as atividades a serem executadas em cada poço e os recursos disponíveis para a execução dessas atividades, determinar um seqüenciamento das atividades em um dado horizonte de tempo, indicando seus instantes de início e fim e o recurso que a realizará, de forma a otimizar uma função objetivo, a saber, maximizar a produção de petróleo. Este seqüenciamento deve levar em conta diversas restrições operacionais e de engenharia. Além disso, pode ser considerado o deslocamento dos recursos. Nesse trabalho, o EDP é estudado, levando em conta as especificidades encontradas pela Petrobras na exploração de petróleo em águas profundas. As restrições operacionais e de engenharia serão apresentadas em detalhes, bem como uma metaheurística para a resolução do problema.

Propostas de solução já foram desenvolvidas para problemas similares (do Nascimento (2002)). Entretanto, o EDP na forma aqui estudada inclui restrições de grande relevância prática que não são tratadas em outros trabalhos, e que nos foram fornecidas pela Petrobras. Por exemplo, no EDP com deslocamento de recursos (que chamamos de EDPDR) considera-se que a movimentação dos recursos entre os poços consome tempo e que, nesse período, eles ficam indisponíveis.

2.1 Produção de Petróleo e Objetivos

A produção de petróleo de um poço é calculada da seguinte forma. Cada poço tem uma vazão associada e uma atividade cujo propósito é indicar o início da produção. Quando esta última atividade é concluída, o poço é considerado em produção. A produção é calculada multiplicando-se a vazão de petróleo do poço pelo tempo restante desde o início da produção até o horizonte de produção estabelecido. Se o início da produção ocorrer após o horizonte de produção, a produção do poço correspondente não será considerada.

O objetivo é obter um escalonamento para todas as atividades de desenvolvimento, satisfazendo todas as restrições e maximizando a produção de petróleo. Outros objetivos a serem atingidos são:

1. *Gerar soluções mais rapidamente.* Soluções criadas por engenheiros levam muitas horas, até dias, para serem construídas. Um método mais rápido permitiria a análise de diferentes cenários para o mesmo problema como, por exemplo, acrescentando-se ou removendo-se recursos. Além disso, imprevistos não resultariam em novas horas ou dias em replanejamento.

2. *Melhor alocação de recursos.* Com a automação do escalonamento, engenheiros altamente especializados, antes responsáveis por planejar o escalonamento, podem ser realocados para outras áreas da empresa.

3. *Ganhos.* O escalonamento automatizado deverá gerar soluções pelo menos tão boas quanto as manuais. A ferramenta desenvolvida deve também gerar soluções pelo menos tão boas quanto àquelas geradas pelo ORCA, uma aplicação em uso na Petrobras.

2.2 As Restrições do EDP

As principais restrições envolvidas no processo de escalonamento das atividades de desenvolvimento dos poços de petróleo são:

C1. *Precedência Tecnológica*: estabelece uma ordem entre as atividades. Há quatro categorias de precedência tecnológica:

   (a) FS(A,B): Atividade A deve terminar depois do início da atividade B.

   (b) SS(A,B): Atividade A deve iniciar depois do início da atividade B.

   (c) SF(A,B): Atividade A deve iniciar depois do fim da atividade B.

   (d) FF(A,B): Atividade A deve terminar depois do fim da atividade B.

C2. *Marco-Atividade*: uma atividade deve terminar antes ou iniciar depois de uma determinada data, ou *marco*, com ou sem *lag*. Essa data está geralmente relacionada com algum evento externo, *e.g.*, a instalação de uma plataforma petrolífera.

C3. *Baseline*: estabelece uma data de início para uma determinada atividade.

C4. *Restrições no Uso dos Recursos*: para executar uma atividade, dependendo da sua natureza, é necessário um recurso que atenda às suas características operacionais. Para uma atividade que requer um barco, deve ser verificado se os equipamentos do mesmo podem operar na profundidade especificada. Para uma atividade que requer uma sonda, deve ser verificado:

   (a) Seu tipo: ancorada, SSDP, NSDP ou navio-sonda.

   (b) Suas capacidades: TOP DRIVE, HPHT, BOP 16, BOP 18, Perfuração.

   (c) Sua máxima e mínima profundidade de operação.

   (d) Sua máxima profundidade de perfuração.

C5. *Concorrência*: duas atividades do mesmo poço não podem ser executadas simultaneamente. Similarmente, duas atividades não podem ser executadas por um mesmo recurso simultaneamente.

C6. *Indisponibilidade*: recursos podem estar indisponíveis por um período de tempo, para manutenção ou devido a término de contrato.

C7. *Sequências de Poços Definidas pelo Usuário*: o usuário pode especificar uma sequência desejada para as atividades de perfuração ou para as atividades de "início de produção" de diferentes poços. A sequência é uma lista ordenada de dois ou mais poços de tal forma que se o poço A precede o poço B na lista, então a atividade F_A do poço A deve terminar antes do início da atividade S_B do poço B. As atividades F_A e S_B são ou atividades de "perfuração" ou atividades de "início de produção" de seus respectivos poços, dependendo do tipo da sequência. Essas sequências são especificadas por engenheiros de modo a evitar perda de pressão no campo petrolífero.

C8. *Restrição de Superfície*: também conhecida como *bolha assassina*, essa restrição representa uma área de segurança, definida pelo usuário ao redor de cada poço, para que as sondas usadas em atividades desses poços não colidam. A área restrita é especificada por um polígono fechado definido por coordenadas ao redor do poço. Quando o centro do primeiro poço está dentro da área restrita do segundo poço, as atividades executadas por sondas nesses poços não podem ser simultâneas. Essas restrições devem ser verificadas entre cada par de sondas móveis, e entre cada par de sondas

móveis e ancoradas. Por exemplo, na Figura 2.2, o centro do poço JUB-16 está dentro da área de segurança do poço JUB-02-I, e portanto atividades executadas por sondas nesses poços não podem ser simultâneas.

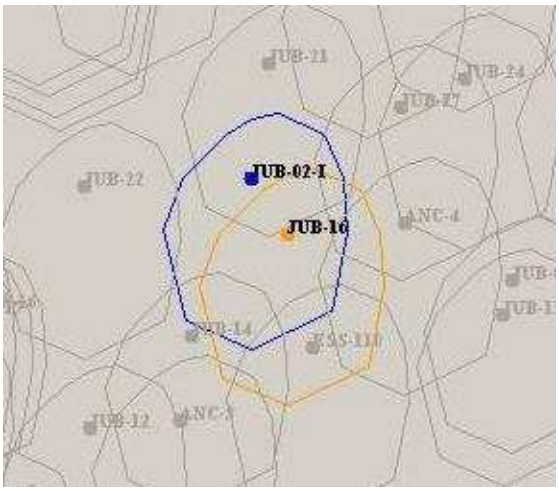


Figura 2.2: Restrição de Superfície.

C9. *Cluster*: uma atividade pode ser parte de um *cluster*, que é um conjunto de atividades que devem usar o mesmo recurso.

C10. *Deslocamento de Sonda*: Essa restrição não aparece no EDP puro, mas é considerada no EDPDR. Ela estabelece que quando uma sonda se deslocar de um poço para outro, um tempo de *set-up*[5] deve ser considerado. Dessa forma, deslocamentos desnecessários devem ser evitados, por exemplo, fazendo com que uma sonda execute o máximo de atividades de um mesmo poço.

Pela descrição do problema e de suas restrições, pode ser observado que, além de lidarmos com uma aplicação real e de grande importância, trata-se um problema com várias particularidades, que o diferencia em relação à problemas descritos na literatura (do Nascimento (2002)).

2.3 Relação entre o EDP e o *Job Shop Scheduling*

Em várias decisões e escolhas de parâmetros em nosso resolvedor, tomamos como base estudos e técnicas utilizadas para o problema clássico do *Job Shop Scheduling (JSP)*. Isso

[5]Tempo para desancorar, deslocar, e ancorar novamente.

foi feito porque o EDP e o JSP podem ser facilmente relacionados, como se mostra a seguir, apesar de o EDP ser um problema bem mais complexo.

Um JSP pode ser definido como: um conjunto M de máquinas, um conjunto J de tarefas, e para cada tarefa $j \in J$ está associada uma coleção ordenada de operações $t_k[j]$, $1 \leq k \leq n_j$, onde n_j é o número de operações associadas com a tarefa j. Para cada operação t (t é uma abreviação de $t_k[j]$) está associada uma duração $l(t)$ e uma máquina $m(t)$. O problema é encontrar um escalonamento para as operações que otimize uma dada função objetivo. Esse escalonamento deve obedecer à ordem das operações nas tarefas e não pode permitir que a mesma máquina execute mais de uma operação simultaneamente. A relação entre o JSP e o EDP pode ser observada considerando-se as tarefas como poços de petróleo, e as operações das tarefas como as atividades dos poços. A relação de precedência entre atividades dos poços corresponde à ordem das operações das tarefas. Os recursos do EDP seriam as máquinas que executam as operações no JSP. Note, entretanto, que o EDP é um problema bem mais complexo que o JSP, já que as atividades não têm necessariamente um recurso determinado para executá-las. Além disso, uma solução do EDP deve satisfazer a muitas e variadas restrições e sua função objetivo é a maximização da produção de petróleo.

Com base nessa relação, do Nascimento (2002) provou que uma versão mais simples do EDP é NP-difícil.

2.4 Histórico

Para o entendimento da magnitude do problema e de sua importância para a Petrobras, apresentamos um breve histórico das etapas do desenvolvimento do projeto:

- Início de 2000: Ocorrem as primeiras reuniões entre engenheiros da E&P (Exploração e Produção) e desenvolvedores da TI (Tecnologia da Informação), ambas áreas da Petrobras. O problema é formalmente descrito e inicia-se um estudo de como resolvê-lo de forma automática. Até então, ele é resolvido manualmente por engenheiros especialistas da Petrobras.

- Meados de 2000: A Petrobras contrata uma mestranda do INPE especialista em heurísticas e inicia contatos com a UNICAMP (Moura & de Souza (2000)).

- Início de 2001: Os professores doutores Arnaldo Vieira Moura e Cid Carvalho de Souza fazem reuniões com os graduandos Romulo Albuquerque Pereira e Vinícius José Fortuna e lhes explicam o problema.

- Meados de 2001: Aprovadas as bolsas de IC (Iniciação Científica) para os alunos Romulo e Vinícius, que iniciam os primeiros trabalhos em relação ao problema. A Petrobras, por sua vez, inicia trabalhos próprios para lidar com o problema.

- Início de 2002: Juliana Martins do Nascimento inicia seu Mestrado atacando o mesmo problema.

- Meados de 2002: Fim da IC de Romulo e Vinícius. O primeiro desenvolveu modelos matemáticos, que se mostraram ruins para o problema, e modelos de programação por restrições (Pereira et al. (2002)). O segundo desenvolveu uma eficiente ferramenta que utiliza da Busca Tabu para resolução do problema (Fortuna et al. (2002)). Romulo passa a trabalhar como terceirizado na Petrobras, cuidando do desenvolvimento da ferrramenta que a Petrobras vinha fazendo.

- Início de 2003: Romulo inicia seu mestrado no mesmo problema, porém com mais restrições e dados reais.

- Fim de 2003: Juliana defende sua tese de mestrado, mostrando uma ferramenta híbrida que utiliza programação por restrições e busca tabu para atacar o problema (do Nascimento (2002)). Como a Petrobras, voltada para seu próprio desenvolvimento, não mais manteve contato, Juliana resolveu um problema parcial, incompleto e mais simples.

- Meados de 2004: Romulo termina sua ferramenta GRASP para o EDP. A Petrobras requisita que se considere o deslocamento de recursos.

- Início de 2005: Artigo do EDP é aprovado no *4th International Workshop on Efficient and Experimental Algorithms*[6] (WEA2005). Romulo vai apresentar o trabalho na conferência, realizada na ilha de Santorini, Grécia. O trabalho é publicado no livro Pereira et al. (2005*a*). Finda-se também um GRASP para o EDP com deslocamento de recursos (EDPDR).

- Meados de 2005: Artigo do EDPDR sagra-se um dos vencedores do Prêmio Petrobras de Tecnologia[7]. A ferramenta desenvolvida é incorporada ao ORCA, ferramenta da Petrobras para o EDPDR, e passa a ser utilizada pelos engenheiros da Petrobras em processos da Bacia de Campos e Bacia do Espírito Santo.

- Fim de 2005: Submissão do artigo do EDPDR ao *Journal of Heuristics*[8]. A tese de mestrado de Romulo é redigida.

[6] http://ru1.cti.gr/wea05/
[7] http://www2.petrobras.com.br/tecnologia2/port/pptecnologia.asp
[8] http://www.kluweronline.com/issn/1381-1231

Parte II

Fundamentos Teóricos

Capítulo 3

GRASP

Neste projeto foi utilizada a metodologia GRASP para buscar soluções para o problema tratado. Alguns artigos de referência do GRASP são: Resende & Ribeiro (2002), Souza et al. (2004), Binato & Oliveira (2002), Faria et al. (2005), Srinivasan et al. (2000), Ribeiro (2002) e Festa & Resende (2002).

A metodologia GRASP (*Greedy Randomized Adaptive Search Procedure*) é um processo iterativo, onde cada iteração consiste de 2 fases, **Construção** e **Busca Local**. A fase **Construção** constrói uma solução factível, onde a vizinhança é explorada posteriormente pela fase **Busca Local**, sendo este processo realizado por diversas iterações. A melhor solução dentre todas iterações do GRASP é selecionada como resultado final.

```
1: procedure GRASP(ListSize, MaxIter, Seed)
2: for k = 1 to MaxIter do
3:    Solution ← Construct_Solution(ListSize, Seed);
4:    Solution ← Local_Search(Solution);
5:    Update_Solution(Solution, Best_Solution_Found);
6: end for
7: return Best_Solution_Found;
8: end GRASP
```

Figura 3.1: Pseudo-código da Metaheurística GRASP

A Figura 3.1 ilustra uma implementação genérica do bloco principal do GRASP, em pseudo-código. As entradas para o algoritmo incluem parâmetros para o ajuste do *tamanho da lista de candidatos (ListSize)*, *número máximo de iterações do GRASP (MaxIter)*, além da *semente aleatória (Seed)* para uso inicial na geração de valores aleatórios. As iterações do GRASP são mostradas nas linhas 2 à 6. Cada iteração do GRASP, como dito anteriormente, consiste de uma fase Construção (*Construct_Solution*), linha 3, uma fase Busca

Local (*Local_Search)*, linha 4 e, se necessário, uma atualização da solução encontrada, linha 5. O algoritmo *Construct_Solution* gera uma solução factível para o problema, atualizando a variável *Solution*. De posse da solução factível gerada, o algoritmo *Local_Search* procura por uma solução melhor visando a minimização ou a maximização de uma função objetivo, e atualiza a variável *Solution*. O algoritmo *Update_Solution* compara a solução encontrada com a solução armazenada, que é a melhor solução dentre todas que já foram encontradas. Caso a solução gerada seja melhor que a atualmente armazenada, este substitui a solução armazenada pela solução atual. Este processo de *Construção, Busca* e *Atualização* é realizado um número *MaxIter* de vezes relativamente grande, a fim de procurar obter a melhor solução possível para o problema.

Fase Construção

Na fase Construção é construída uma solução factível, um elemento de cada vez. Em cada iteração da Construção, o próximo elemento a ser adicionado é determinado ordenando-se todos os possíveis elementos em uma lista de candidatos com respeito a uma função gulosa (*greedy*) que mede o benefício de selecionar cada elemento. Esta lista é chamada de *RCL (Restricted Candidate List)*. A componente adaptável da heurística vem do fato de que os benefícios associados com cada elemento são atualizados em cada iteração da fase Construção a fim de refletir as alterações incorporadas pela seleção dos elementos anteriores. A componente probabilística do GRASP é caracterizada pela escolha aleatória de um dos melhores candidatos da RCL, mas geralmente não o melhor deles. Esta maneira de realizar a escolha permite a obtenção de diferentes soluções para cada iteração do GRASP, não necessariamente descaracterizando a componente gulosa adaptável. As soluções geradas pela fase Construção do GRASP não são garantidamente ótimas com respeito às definições da vizinhança. Portanto, é quase sempre benéfico aplicar uma busca local para tentar gerar uma solução melhor.

A Figura 3.2 ilustra uma implementação genérica do bloco Construção do GRASP, em pseudo-código. As entradas para o algoritmo incluem parâmetros para o *ajuste do tamanho da lista de candidatos (ListSize)*, além da semente aleatória *(Seed)* para uso inicial na geração de valores aleatórios. As iterações do bloco Construção são mostradas nas linhas de 2 à 8.

Conforme mencionado anteriormente, este algoritmo constrói uma solução factível através da avaliação incremental dos possíveis candidatos, selecionando os candidatos dentre aqueles incluídos na RCL. A seleção de um candidato s da lista é realizada aleatoriamente e a variável *Solution* é atualizada, incluindo-se o candidato selecionado. De posse deste candidato, é realizada uma reavaliação incremental do custo da solução. Este processo termina quando uma solução é determinada.

```
1: procedure Construct_Solution(ListSize, Seed)
2: Solution ← 0;
3: Evaluate the incremental costs of the candidate elements;
4: while Solution is not a complete solution do
5:    Build the RCL(ListSize);
6:    Select an element s from the RCL at random;
7:    Solution ← Solution ∪ {s};
8:    Reevaluate the incremental costs;
9: end while
10: return Solution;
11: end Construct_Solution
```

Figura 3.2: Pseudo-código da Fase Construção

Fase Busca Local

Um algoritmo de busca local funciona de um modo iterativo, sucessivamente trocando a solução atual por uma solução melhor de sua vizinhança. O algoritmo termina quando não é encontrada solução melhor na vizinhança com respeito à função objetivo.

A Figura 3.3 ilustra uma implementação genérica da Busca Local do GRASP em pseudo-código. A entrada para o algoritmo inclui como parâmetro a *Solução encontrada na fase Construção (Solution)*. As iterações da Busca Local são mostradas nas linhas de 2 à 4.

```
1: procedure Local_Search(Solution)
2: while Solution is not locally optimal do
3:    Find s' ϵ N(Solution) with f(s') < f(Solution);
4:    Solution ← s';
5: end while
6: return Solution;
7: end Local_Search
```

Figura 3.3: Pseudo-código da Fase Busca Local

Este algoritmo procura otimizar a solução encontrada na fase Construção através da busca de um vizinho melhor s', isto é, um elemento de menor (minimização) ou maior (maximização) valor segundo a função objetivo. Esta busca na vizinhança pode ser implementada usando-se tanto a estratégia *"best-improving"* quanto a estatégia *"first-improving"*. No primeiro caso, *best-improving*, todos os vizinhos são analisados e o melhor dentre eles é selecionado. Já na estratégia *first-improving*, a seleção é realizada para o primeiro candidato que apresentar um valor melhor para a função objetivo, sendo a solução atual atualizada

para este valor.

Aplicações do GRASP

A metaheurística GRASP tem sido aplicada com sucesso em diversos tipos de problemas de otimização combinatória, incluindo-se, *set covering*, *quadratic assigment*, roteamento de veículos, problemas de localização, conjunto máximo independente, *feedback vertex set*, planejamento de rede de transmissão, e planarização de grafos, entre outros. Para uma bibliografia detalhada, vide Festa & Resende (2002).

Capítulo 4

Programação por Restrições

O trabalho Jaffar & Lassez (1987) estabeleceu os fundamentos teóricos a partir dos quais diversas linguagens de programação por restrições puderam ser desenvolvidas[1]. A idéia básica é substituir o mecanismo de inferência lógica tradicional (unificação) por um mecanismo mais genérico e eficiente, conhecido como *manipulação de restrições*. Este mecanismo já é usado no campo da Inteligência Artificial desde a década de 80, na resolução dos chamados *Constraint Satisfaction Problems.* (CSPs). Um CSP é composto por

- Um conjunto de *variáveis* $X = \{x_1, ..., x_n\}$;
- Para cada variável x_i, um conjunto finito de valores, D_i, denominado seu *domínio*;
- Um conjunto de *restrições*, C, que atuam sobre subconjuntos das variáveis de X, limitando os valores que lhes podem ser atribuídos.

Uma solução para um CSP é uma atribuição de valores às variáveis de X, obedecendo as restrições de C.

Muitos problemas em Pesquisa Operacional, tais como escalonamento de tarefas, alocação de horários e outros problemas combinatórios podem ser representados como CSPs, e a Programação por Restrições vem se afirmando como uma ferramenta poderosa para abordar tais problemas (Lever et al. (1995), Le Pape (1994), Hasle et al. (1997), Yunes et al. (2000*a*) e Yunes et al. (2000*b*)).

Um programa em Programação por Restrições geralmente obedece à estrutura apresentada na Tabela 4.1. Note que essa estrutura se assemelha a um CSP.

As variáveis com seus domínios definem o espaço de soluções. As restrições estabelecem *relações* entre as variáveis, limitando os valores que elas podem assumir concomitantemente

[1]Entretanto, vale ressaltar que desde a década de 60 já existem linguagens que lidam com restrições.

<Declaração de Variáveis e Domínios>
<Imposição de Restrições>
<Busca por Soluções (*labeling*)>

Tabela 4.1: Estrutura básica de um programa em Programação por Restrições

e reduzindo o espaço de busca. Essa redução se processa através de um mecanismo denominado *propagação de restrições*, cuja função é garantir a *consistência parcial* do sistema como um todo.

O mecanismo de propagação de restrições funciona da seguinte forma. Sempre que o domínio de uma variável é alterado, e.g. pela remoção de um de seus elementos, esta informação é transmitida (propagada) para as demais variáveis associadas a ela por uma ou mais restrições. Durante esse processo, procura-se sempre manter a consistência do sistema, i.e. a satisfação das restrições do programa. Por exemplo: sejam X e Y variáveis que podem assumir valores inteiros no intervalo (domínio) $[1, 10]$. Isto significa que, inicialmente, os domínios de X e Y são os mesmos: $D_X = D_Y = [1, 10]$. Impondo-se a restrição de que $X < Y$, os domínios se alteram para $D_X = [1, 9]$ e $D_Y = [2, 10]$. Isto porque, se $X = 10$ e $X < Y$, conclui-se que $Y \geq 11$. Contudo 11 não pertence ao domínio de Y. Analogamente, mostra-se que Y não pode receber o valor 1. Suponha agora que a restrição $X \geq 5$ seja imposta. Isto faz com que $D_X = [5, 9]$. Como Y está relacionada a X pela restrição $X < Y$, o domínio de Y é automaticamente alterado para $D_Y = [6, 10]$.

A consistência do sistema é dita *parcial* porque não há algoritmos polinomiais conhecidos que garantam a consistência *total* de um CSP genérico[2] (Marriott & Stuckey (1998)). Além disso, por questões de eficiência, os algoritmos normalmente utilizados para detectar violações das restrições não são capazes de identificar certos tipos de estados inconsistentes. Um desses algoritmos, denominado *arc-consistency*, funciona assim: sejam duas variáveis, A e B, relacionadas por uma restrição r, e cujos domínios são, respectivamente, D_A e D_B. Diz-se que um valor $v_A \in D_A$ tem *suporte* no domínio D_B se, ao atribuir-se v_a a A, existir um valor $v_B \in D_B$ que pode ser atribuído a B sem violar r. O algoritmo, então, removerá do domínio de cada variável todos aqueles valores que não têm suporte no domínio de uma outra variável qualquer, associada a ela por alguma restrição (Marriott & Stuckey (1998)).

Em resumo, o que acontece é o seguinte: os domínios originais das variáveis são inicialmente reduzidos pelas restrições do programa. Neste instante, é possível que haja alguma inconsistência ainda não detectada. A seguir, dá-se início ao processo de *labeling* (vide Tabela 4.1) onde, a cada passo, seleciona-se uma variável à qual será atribuído um valor de seu domínio. A ordem de escolha das variáveis e dos valores é totalmente flexível e influencia

[2]CSPs pertencem à classe de problemas NP-completos.

no tempo de resposta do programa. Ao se atribuírem valores às variáveis, o mecanismo de propagação se encarrega de reduzir os outros domínios. Caso alguma inconsistência seja detectada, a última atribuição de valor tem de ser desfeita (*backtracking*), dando lugar a uma outra alternativa. O processo todo se repete até que uma solução seja encontrada ou até que se prove a inexistência de soluções viáveis (Marriott & Stuckey (1998), Lever et al. (1995)).

Em certos casos, procura-se não somente uma solução viável qualquer, mas aquela que minimiza (ou maximiza) uma função objetivo. Uma forma de dar suporte ao conceito de otimização em Programação por Restrições é a seguinte. Sempre que uma solução é encontrada, calcula-se o seu custo c e impõe-se uma nova restrição ao sistema indicando que o custo da próxima solução deverá ser menor que c (em caso de minimização). Repete-se este processo até que não haja mais soluções viáveis. Nesse ponto, a última solução encontrada é uma solução de custo ótimo (Marriott & Stuckey (1998), Lever et al. (1995)).

Um grande desafio da programação por restrições é a criação de um modelo que represente adequadamente as condições reais de um problema. Um modelo ruim, por exemplo, pode gerar soluções errôneas ou não satisfatórias, e até mesmo não levar a uma solução. O modelo, portanto, será um fator que determinará o tempo de resposta do programa e a qualidade da solução gerada.

Parte III

O Trabalho Desenvolvido

Capítulo 5

ORCA: Resolvedor de Programação por Restrições

Em 2000, um consultor da área de Exploração e Produção (E&P) da Petrobras decidiu criar um projeto para automatizar o processo de escalonamento das atividades de desenvolvimento de poços de petróleo em alto mar e seus respectivos recursos (barcos e sondas), até então feito manualmente. A ferramenta também deveria otimizar a produção de petróleo.

Algumas análises foram conduzidas e o time de projeto desenvolveu um modelo de Programação por Restrições (cf., Marriott & Stuckey (1998)) usando as bibliotecas ILOG Solver e ILOG Scheduler (ILOG (1999*a*)). Essas bibliotecas permitem fácil modelagem, manutenção e entendimento do código.

Depois de quatro anos de desenvolvimento, alterações, correções e melhorias, a ferramenta desenvolvida, de nome ORCA ("Otimização de Recursos Críticos na Atividade de produção"), tornou-se operacional e bem sucedida.

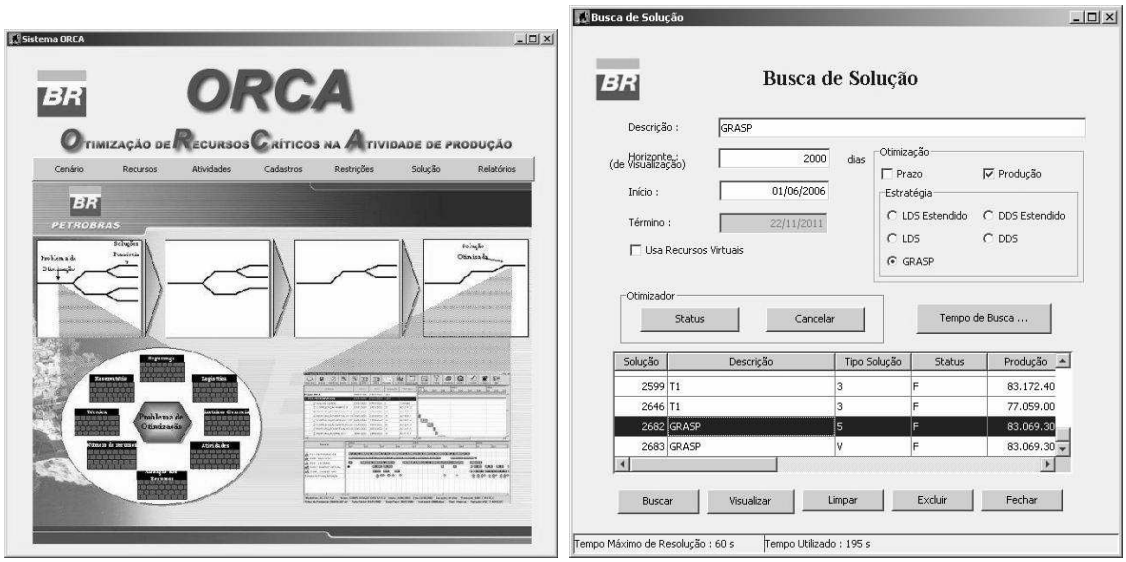
As próximas seções descrevem o modelo e os algoritmos utilizados no desenvolvimento do ORCA.

5.1 Modelo do ORCA

A modelagem do ORCA foi feita utilizando o ILOG Solver e o ILOG Scheduler, bibliotecas C++ de Programação por Restrições da ILOG [1]. Usando tal abordagem, a representação do problema é feita desassociada dos algoritmos usados para resolvê-lo, dessa forma oferecendo facilidade de desenvolvimento, entendimento e adaptação (ILOG (1999*b*)).

O modelo consiste de dois tipos de variáveis, ambos inteiros:

[1] http://www.ilog.com



(a) Tela Inicial (b) Tela de Busca de Solução

Figura 5.1: ORCA

- Um que representa a data de início de execução da atividade por um recurso. É caracterizado por um mínimo e máximo tempo de início.

- Um que representa qual recurso executará a atividade do poço. É caracterizado por um conjunto de possíveis recursos, dos quais um deve ser escolhido para realizar a respectiva atividade.

Todas as restrições descritas na Seção 2.2 foram impostas sobre as variáveis descritas acima através de métodos predefinidos das classes da ILOG. Por examplo, sendo a e b atividades representadas por objetos da classe ILOG *IlcActivity* e seja *P1* o modelo representado pela classe ILOG *IlcManager*, estabelecemos que a deve iniciar depois do fim de b através do seguinte código:

P1.add(a.startsAfterEnd(b));

O mecanismo de propagação das restrições, responsável pela redução de domínio das variáveis, fica a cargo das bibliotecas da ILOG. Entretanto, a simples propagação das restrições não é suficiente para se chegar a um valor único nas variáveis. Dessa forma, faz-se necessário utilizar de um algoritmo de busca para encontrar soluções.

5.1.1 Algoritmos do ORCA

As bibliotecas da ILOG provêm não apenas um algoritmo de busca genérico, mas também meios para se desenvolver um método de busca próprio.

No ILOG Solver, a implementação de algoritmos de busca é baseado na idéia de *goals*. *Goals* tornam possível a implementação de algoritmos em que a exata sequência de operações não é conhecida antecipadamente. Esse tipo de programação é freqüentemente conhecida como não-determinística, já que seus passos não estão determinados, ainda que a programação tenha um propósito ou objetivo (*goal*) em vista.

O mecanismo de busca explora o seguinte algoritmo: Enquanto existir alguma variável sem valor fixo,

1. escolha uma dessas variáveis;
2. atribua à variável um dos valores do seu domínio;
3. propague os efeitos dessa atribuição.

Entretanto, não se sabe qual valor no domínio é consistente com as restrições. Assim, se a atribuição levar à inconsistências, ela deve ser desfeita e outro valor deve ser tentado. Tal processo é conhecido como *backtracking*.

5.1.2 Função Objetivo

A Petrobras não quer apenas uma solução para o problema, mas uma solução que maximize a produção de petróleo. Para isso, o ILOG Solver oferece meios predefinidos para minimizar ou maximizar um critério. A técnica usada no Solver é o clássico *Branch & Bound* (Cormen et al. (2001)).

Para realizar a otimização, a implementação da biblioteca ILOG segue o seguinte algoritmo:

1. Busque uma solução da forma definida pelos *goals* inseridos no modelo.
2. Salve a solução encontrada.
3. Acrescente a restrição que estabelece que a próxima solução deve ser pelo menos uma unidade melhor que a solução encontrada anteriormente.
4. Repita o processo até que nenhuma outra solução seja encontrada ou até que o tempo de busca se esgote.

Assim, até o fim da busca, teremos a melhor solução encontrada, a com a maior produção de petróleo.

5.1.3 Técnicas de Busca

O ILOG Solver implementa vários procedimentos de busca de pesquisa operacional e inteligência artificial, tais como *best first search* (BFS), *limited discrepancy search* (LDS), *depth bounded discrepancy search* (DDS), e *interleaved depth first search* (IDFS). Esses procedimentos de busca mudam a ordem na qual os nós da árvore de busca são explorados.

A biblioteca implementa uma representação implícita dos estados de busca para armazenar vários estados a serem explorados. Essa representação envolve um caminho na árvore binária de busca. O caminho consiste de uma sequência de decisões — esquerda e direita — na árvore de busca. A busca vai de uma posição na árvore para outra através de uma combinação de *backtracking* e recomputação (Clocksin (1987)).

A busca do Solver trabalha assim: nós abertos (isto é, nós que podem ser explorados) são guardados numa pilha, e, a cada passo, o Solver escolhe o melhor nó aberto de acordo com uma função de avaliação e o explora. O Solver implementa funções de avaliação predefinadas para as seguintes técnicas de busca:

- *Depth First Search (DFS)* é o procedimento de busca padrão do ILOG Solver. DFS considera as arestas de saída de um vértice antes que os vizinhos daquele vértice. O maior problema do DFS é que ele não se recupera de más escolhas iniciais. Uma má decisão tomada no início pode levar a busca a percorrer inúmeros estados inconsistentes ou de baixa qualidade.

- *Best First Search (BFS)* é um algoritmo de busca que otimiza o *Breadth First Search*[2] via ordenação de todos os caminhos correntes de acordo com alguma heurística. A heurística tenta predizer quão perto o caminho está de alcançar uma solução. Caminhos que são tidos como mais próximos de uma solução são explorados primeiro (Nilsson (1971)). A implementação do Solver usa um parâmetro ϵ. Ao selecionar um nó aberto, o Solver determina o conjunto de nós abertos cujo custo é no máximo $1+\epsilon$ pior que o melhor nó aberto. Se o filho do nó corrente está nesse conjunto, o Solver explora esse filho. Se não, o Solver escolhe o melhor nó aberto.

- *Limited Discrepancy Search (LDS)* foi definido em Harvey & Ginsberg (1995). A discrepância de um nó de busca é definida como sua profundidade direita, isto é, o número de vezes que a busca escolheu a ramificação direita em um ponto de escolha (*choice point*) quando ia do nó raiz até o nó corrente. Dado um parâmetro k, a busca vai primeiro explorar nós com discrepância menor que k. Depois que essa exploração for completada, ele irá explorar nós com discrepância entre k e $2k$, e assim por diante. Dessa forma, essa busca corta a árvore de busca em tiras.

[2] *Breadth first search* é um algoritmo de busca em árvore que começa no nó raiz e explora todos os nós vizinhos, e só depois seus filhos.

- *Depth Bounded Discrepancy Search (DDS)* foi introduzido em Walsh (1997). Uma variação do LDS, o DDS assume que más escolhas são provavelmente tomadas mais próximo do topo da árvore de busca. Por isso, esse procedimento não conta o número de discrepâncias, mas a profundidades da última. O ILOG Solver implementa uma versão aprimorada do esquema em Walsh (1997); não se conta a profundidade da última discrepância, mas a da d-ésima última, onde d é um parâmetro da busca.

- *Interleaved Depth First Search (IDFS)* foi introduzido em Meseguer (1997). IDFS tenta imitar o comportamento de um número infinito de *threads* explorando a árvore de busca. O Solver implementa uma variação que limita a profundidade deste comportamento.

Todas esses procedimentos de busca foram testados em instâncias reais do EDP. O BFS se provou ineficiente para o problema. Em quase todas as instâncias, ele não gerou soluções dentro da primeira hora de execução. O DFS e o IDFS se mostraram bem melhores, mas os melhores resultados foram obtidos usando o LDS e o DDS. Nos testes, o LDS foi levemente superior ao DDS, além de ser mais flexível. Isso porque o DDS é eficiente se a heurística de busca é muito boa, *i.e.*, se ela faz más escolhas apenas no topo da árvore de busca (ILOG (1999*a*)), o que é algo difícil de se obter.

5.1.4 Goals para Escalonar as Atividades

Apresentamos agora os *goals* criados para guiar o processo de busca. Eles são executados na ordem em que são apresentados:

1. **Escolhendo os recursos:** Este *goal* seleciona para cada atividade o recurso disponível que menos foi utilizado até o momento.

2. **Tratando a restrição C7:** Caso haja sequências de poços, este *goal* atribui datas de início de execução para as atividades das sequências. Na ordenação das atividades, ele seleciona a variável disponível entre aquelas das sequências que ainda não foi atribuída e que gera o maior incremento na produção de petróleo. Dá-se preferência às atividades de perfuração em detrimento das de produção.

3. **Escalonando atividades de produção:** Este *goal* atribui datas de início de execução para as atividades de início de produção. Na ordenação das atividades, ele seleciona a variável disponível não atribuída que gera o maior incremento na produção de petróleo.

4. **Escalonando as outras atividades:** Este *goal* atribui a data de início de todas as atividades restantes. Na ordenação das atividades, ele seleciona a primeira ainda não atribuída.

5.1.5 Resultados do ORCA

O ORCA foi freqüentemente utilizado por engenheiros da Petrobras para definir bons escalonamentos no desenvolvimento dos poços de petróleo em alto mar. Foi também usado para analisar a necessidade de se adquirir ou alugar novos recursos. Esses usuários confirmaram que o ORCA gera soluções pelo menos tão boas quanto as manuais. E, além disso, demanda apenas um minuto de execução para gerar um bom escalonamento, quando manualmente se demorava de um a dois dias. Em uma de suas utilizações numa instância real, o ORCA mostrou ser melhor acrescentar um barco LSV à comprar uma terceira sonda. Com isso, o ORCA assegurou ganhos de US$15 milhões à Petrobras, e antecipou o início de produção dos poços em 26 dias.

Capítulo 6

O EDP sem Deslocamento de Recursos

Prólogo

O artigo a seguir trata o problema de escalonamento no desenvolvimento de poços de petróleo em alto mar (EDP), não sendo considerados deslocamentos de recursos.

Após uma descrição detalhada do problema e das restrições envolvidas, é apresentado um resolvedor GRASP que supera uma ferramenta de programação por restrições desenvolvida pela Petrobras.

Este artigo é uma versão reduzida do relatório técnico Pereira et al. (2005*b*) e foi apresentado na conferência *Fourth International Workshop on Efficient and Experimental Algorithms* (WEA2005, http://ru1.cti.gr/wea05/), que ocorreu na ilha de Santorini, Grécia, nos dias 10 a 13 de maio de 2005. O trabalho foi um dos 47 artigos regulares aceitos dentre 176 submetidos e foi publicado no volume 3503 da série *Lecture Notes in Computer Science*, que contém os anais da conferência (Pereira et al. (2005*a*)). Convidado, Romulo foi apresentar seu trabalho na conferência, tendo excelente receptividade pela comunidade presente. O artigo também está sendo considerado para o *Special Issue of the ACM Journal on Experimental Algorithmics* (JEA, http://www.jea.acm.org), dedicado ao evento.

O artigo está transcrito na sua íntegra, em inglês.

Comparative Experiments with GRASP and Constraint Programming for the Oil Well Drilling Problem

Romulo A. Pereira
romulo_a_pereira@yahoo.com.br

Arnaldo V. Moura
arnaldo@ic.unicamp.br

Cid C. de Souza
cid@ic.unicamp.br

Institute of Computing, University of Campinas

Abstract

Before promising locations become productive oil wells, it is often necessary to complete drilling activities at these locations. The scheduling of such activities must satisfy several conflicting constraints and attain a number of goals. Here, we describe a Greedy Randomized Adaptive Search Procedure (GRASP) for the scheduling of oil well drilling activities. The results are compared with those from a well accepted constraint programming implementation. Computational experience on real instances indicates that the GRASP implementation is competitive, outperforming the constraint programming implementation.

1 Introduction

Oil extracted from oceanic basins is an increasingly important fraction of the total world offer of petroleum and gas. Usually, diverse petroliferous basins are explored, each with hundreds of promising spots where productive oil wells could be located. However, before these places are turned into productive wells they must be developed, that is, a sequence of engineering activities must be completed at each promising spot, to render them ready for oil extraction. Oil derricks and ships are used to complete these activities. These resources are limited and expensive, either in acquisition or rent value, and must be used efficiently.

The oil *well drilling problem* (WDP) can be summarized thus: given a set of promising spots, the activities to be executed at each location, and the available resources, find a scheduling of the activities and resources, fulfilling several conflicting engineering and operational constraints, in such a way as to optimize some objective criteria. In this work, the specific WDP faced by Petrobras (a leading company in deep water oil extraction) is studied. This WDP imposes much more realistic constraints than other similar studies

(do Nascimento (2002)). The constraints are presented in detail, and an heuristic strategy is developed in order to maximize oil production within a given time horizon.

The next section describes the WDP. Section 3 discusses a GRASP implementation for the WDP. Section 4 presents our computational results obtained with this new algorithm and compares them to other results derived from a constraint programming implementation presently running at Petrobras. Finally, some concluding remarks are offered in the last section.

2 The Well Drilling Problem

After a well is drilled, the preparation for oil extraction develops in several stages. First, oil derricks place Wet Christmas Trees (or WCTs, structures where hydraulic valves are attached) at the mouth of the wells in order to avoid oil leakage. Later, boats connect pipelines between WCTs and manifolds. Manifolds are metallic structures installed by boats at the sea floor. Their use prevents the need for exclusive pipelines connecting each well to the surface, which would be prohibitively expensive. Once this stage is completed, oil extraction can begin. For that, Stationary Units of Production (SUPs) are anchored at specific locations in the surface, and boats interconnect manifolds to them. SUPs are used to process, and possibly store, the extracted products. Later, ships fetch the products from SUPs to land storage sites or other processing units. If the oil outflow is very high or a SUP does not have storage capacity, a petroliferous platform may be installed at the surface.

The constraints involved in the scheduling of oil development activities are:

C1. *Technological Precedence*: sets an order between pairs of activities. When considering the precedence between the start and finish of the activities in each pair, any of the four possibilities can be present.

C2. *Mark-Activity*: an activity must finish before or initiate after a fixed date, with or without lag time. This date is often related to some external event.

C3. *Baseline*: sets the start date of the activities.

C4. *Use of Resources*: to execute an activity, due to its intrinsic nature, a resource used must match some operational characteristics. For a boat, it must be verified if the on-board equipments can operate at the specified depth. For an oil derrick, its type and capabilities must be verified, as well as the maximum and minimum depth of operation and drilling.

C5. *Concurrence*: two activities at the same well, or executed by the same resource, cannot be simultaneous.

C6. *Unavailability*: resources may be unavailable for a period of time, either for maintenance reasons or due to contract expiration.

C7. *User Defined Sequences*: the user can specify a sequence for the drilling or for the "start production" activities of different wells. These sequences are specified by engineers in order to avoid loss of pressure in the oil field. If well A appears before well B in the sequence, then activity F_A of well A must finish before the start of activity S_B of well B. The activities F_A and S_B are either the activity of drilling or the activity of start production of their respective wells, according to the type of the sequence.

C8. *Surface Constraints*: represented by a polygonal security area defined around a well. When a well is inside the restricted area of another well, activities executed at them cannot be simultaneous. These constraints must be verified between pairs of mobile and pairs of mobile and anchored oil derricks.

C9. *Cluster Constraints*: an activity can be part of a cluster, which is a set of activities that must use the same resource.

C10. *Same Derrick*: it is desirable that the same oil derrick executes as much of the activities at a well as possible, in order to avoid unnecessary displacements.

The oil yield is calculated as follows. Each well has an associated outflow and an activity that marks the beginning of its production. When this last activity is concluded, the well is considered in production. The yield is obtained by multiplying the oil outflow by the period between that instant and the established time horizon. If the start production activity is set for after the time horizon, the corresponding yield is disregarded. The objective is to obtain a schedule of all activities, satisfying all constraints, while maximizing the oil yield.

Other goals to be attained by automating the schedule of the activities are:

1. *Faster solutions.* Human made solutions take many hours, even days, to be constructed. A faster method would permit the analysis of different scenarios for the same problem, for example, by adding or removing resources. Furthermore, modifications in already committed plans would not result in new hours, or days, spent in rescheduling.

2. *Better resource allocation.* With an automated scheduling, all highly skilled engineers responsible for the manual scheduling can receive other duties.

From the above description, it can be seen that the WDP is a difficult combinatorial problem. In fact, it is simple to devise a polynomial-time reduction to the classical Job Shop Scheduling problem, showing that the WDP is NP-hard.

The WDP treated here shows several differences from similar problems studied in the literature (do Nascimento (2002)). To tackle the same problem, a project team from Petrobras developed a Constraint Programming (cf., Marriott & Stuckey (1998)) model using ILOG's Solver and Scheduler (ILOG (1999)). After four years of development and testing, the tool, named ORCA (Portuguese acronym for "Optimization of Critical Resources in the Production Activity"), became operational and very successful. Nowadays, the ORCA solver is often used by engineers both to define a good schedule for the drilling activities and, also, to analyze the need for acquiring or renting new resources. They confirmed that ORCA generates better solutions than those made by humans. In one real instance, ORCA showed that buying a third oil derrick was unnecessary and that it was better to add a new LSV ship instead. As a result, Petrobras avoided a expenditure of US$ 15 million, while anticipating oil production by 26 days. Despite the good performance of ORCA, searching for even better solutions is still important, since a tenth of a percent of improvement in the oil production may represent millions of dollars in the company's revenue. The next sections show how we obtain such gains using GRASP.

3 A GRASP for the WDP

Our search for alternatives to compete with ORCA started with an implementation of Tabu Search (Glover & Laguna (1997)) for a simpler version of the WDP (do Nascimento (2002)). However, some issues proved to be particularly difficult to treat, especially the definition of an adequate neighborhood and ways to explore it. After some investigation, GRASP (Feo & Resende (1995)) seemed most appropriate for the WDP. Contrary to what occurs with other metaheuristics, such as tabu search or genetic algorithms, which use a large number of parameters in their implementations, the basic GRASP version requires the adjustment of fewer parameters. Despite its simplicity, GRASP is a well studied metaheuristic which has been successfully applied to a wide variety of optimization problems (cf. Festa & Resende (2001)). In particular, applications of GRASP to scheduling problems can be found in Aiex et al. (2003), Bard & Feo (1989), Feo & Bard (1989), Feo et al. (1995) and Binato et al. (2001).

The next paragraphs review some GRASP basics and describe our specific implementation designed to solve the WDP, named GRASP-WDP (GRASPW). The model and its algorithms are shown in the subsequent paragraphs.

GRASP Basics. In the GRASP methodology each iteration consists of two phases: *construction* and *local search* (Feo & Resende (1995)). Figure 1 illustrates a generic implementation of GRASP, in pseudo-code. The input includes parameters for setting the candidate list size (*ListSize*), the maximum number iterations (*MaxIter*), and the seed (*Seed*) for the

random number generator. The iterations are carried out in lines 2-6. Each iteration consists of the construction phase (line 3), the local search phase (line 4) and, if necessary, the incumbent solution update (line 5). In the construction phase, a feasible solution is built, updating the variable *Solution*. Then the local search algorithm seeks a better solution in the neighborhood of *Solution*, according to a given criterion, and updates *Solution*. This process of construction, search and update is executed *MaxIter* times.

1: procedure $GRASP(ListSize, MaxIter, Seed)$
2: **for** $k = 1$ a $MaxIter$ **do**
3: $Solution \leftarrow Construct\_Solution(ListSize, Seed)$;
4: $Solution \leftarrow Local\_Search(Solution)$;
5: $Update\_Solution(Solution, Best\_Solution\_Found)$;
6: **end for**
7: *return* $Best\_Solution\_Found$;
8: *end* $GRASP$

Figure 1: Pseudo-code of the GRASP Metaheuristic.

In the construction phase, a feasible solution is built one element at a time. Figure 2 illustrates a generic implementation of the construction phase, in pseudo-code. Input includes the candidate list size (*ListSize*) and the seed (*Seed*). The iterations are carried out in lines 2-8. At each iteration, the next element to be added is determined by adding all possible elements to a candidate list, ordered with respect to a greedy function that measures the, maybe myopic, benefit of selecting each element. This list is called the *Restricted Candidate List* (RCL). The adaptive component of the heuristic arises from the fact that the benefits associated with every element are updated at each iteration to reflect the changes brought on by the selection of the element in the previous iteration. The probabilistic component is present by the random choice of one of the best candidates in the RCL, but usually not the best one. This way of choosing elements allows for different solutions to be obtained at each iteration, while not necessarily jeopardizing the adaptive greedy component. The solutions generated by the construction phase are not guaranteed to be locally optimal. Hence, it is almost always beneficial to apply a local search to attempt to improve each constructed solution. The search phase is a standard deterministic local search algorithm that seeks to optimize the solution built in the construction phase.

The GRASPW Implementation. The GRASPW implementation was constructed using the C/C++ programming language. The heuristic uses two types of integer variables. One represents the beginning of execution of each activity in the corresponding well. These values range between a minimum and a maximum start time, with those values depending on the current partial solution being constructed. The second type of variables represents

```
1: procedure Construct_Solution(ListSize, Seed)
2: Solution ← 0;
3: Evaluate the incremental costs of the candidate elements;
4: while Solution is not a complete solution do
5:    Build the restricted candidate list, RCL(ListSize);
6:    Select an element s from the RCL at random;
7:    Solution ← Solution ∪ {s};
8:    Reevaluate the incremental costs;
9: end while
10: return Solution;
11: end Construct_Solution
```

Figure 2: Pseudo-code of the Construction Phase of GRASP.

which resource will execute each activity in its well. Their domains are characterized by a set of the possible resources, of whose one must be chosen to execute the activity. All the constraints described in Section 2 were enforced. Three constraints, namely, C2, C3 and C4, were set while reading the problem data, before the search begins. Note that, in these cases, all values needed to set the constraints are already defined. The other constraints were dealt with during the search for solutions, the variables involved being assigned single values.

The following adaptations were made to the procedure illustrated in Figure 1: (i) the search procedure was interrupted by a time limit instead of by the number of iterations; and (ii) During a complete run of the GRASPW heuristic, the value of *ListSize* can be monotonically incremented by a fixed amount when a predefined interval of time is reached with no improvement on the best solution. Doing so, the algorithm will explore larger regions of the search space. Alternatively, during a run of GRASPW, the value of *ListSize* can be monotonically decremented between iterations, thus focusing into a greedier heuristic. With this scheme we obtain a *dynamic sized RCL* in opposition to the original *static sized RCL*. Note that, as GRASP iterations are independent, one could think that there is no difference between increasing and decreasing the RCL size. However, as we do not know in advance the amount of time the algorithm will execute at each run or when the RCL size will be altered, we can not anticipate the result of a GRASPW run, when increasing or decreasing the RCL size.

As in the ORCA implementation, we seek solutions with the highest oil yield. To this end the construction phase illustrated in Figure 2 was modified thus:

1. The first time ever the construction phase is initiated, we use *ListSize* equal to one, when the algorithm behaves like a pure greedy heuristic. With few constraints obstructing the greedy heuristic, it tends to generate a good or even very good solution.

For example, in two out of twelve real instances, the best solution was found in the first pass of the construction phase.

2. Before line 3, we added a function called *isPressed*. It verifies if any activity of any well has a start time with a small domain and if only one resource can do it at that time. By a start time with a small domain we mean that its minimum and maximum values are very close, such that having only one resource able to execute it indicates that there is almost no flexibility to schedule the activity. If there are such activities, the function schedules their wells. Retarding the schedule of such activities could render the solution infeasible, as other activities may occupy the period of time where those activities would be scheduled. We schedule the wells, and not only the activities, in order to comply with constraint C10.

3. The candidates are defined by the production wells that are available (meaning that there are no wells yet not scheduled which must precede them), or the injection wells that have activities of production wells succeeding them. The activities of injection wells that do not have activities of production wells succeeding them are left to be scheduled after all others. Note that injection wells are not productive and therefore must not be scheduled before production wells, unless there are constraints forcing such a schedule.

4. The evaluation of incremental costs (line 3 of Figure 2) assesses how much oil a well can offer until the end of the time horizon. The RCL is built with those wells that offer the highest yields of oil. Actually, not only the oil offer is considered, but also the oil offer of the constrained successors of that well.

5. In the construction phase, the next element to be introduced in the solution is chosen uniformly from the candidates in the RCL (line 5 of Figure 2). However, any probability distribution can be used to bias the selection.We tried to bias the selection of the candidates proportionally to their oil offer.

6. To schedule the candidate well (line 7 of Figure 2), proceed as follows.

   As long as there are activities not yet scheduled in the well: (i) choose any activity available in the well, *i.e.*, one not yet scheduled and such that there is no other one not yet scheduled in the wells that must precede it; (ii) choose a resource for this activity that can execute it, and that can complete the activity the earliest; (iii) set the start time of the activity at the earliest possible time, *i.e.*, the maximum between the earliest time the resource is available to execute the activity and the minimum start time of the activity; and (iv) all activities that are constrained to succeed the chosen one must have their minimum start times updated to satisfy any constraints.

The scheduling of a well is done so as to satisfy all constraints, including the seven ones not yet enforced. In case of violations, and this can be tested after each activity is scheduled, the construction of this solution is aborted and a new one is started. Instead, we could backtrack a few steps, but this would slow down this phase, especially if the first steps were not appropriate.

7. After a well is scheduled, any activities that must succeed it has their minimum start time updated to satisfy any constraints. If that is not possible, the construction of this solution is also aborted.

For the search phase, an appropriate neighborhood was defined, so as to permit explorations quickly leading to better solutions. The 2-exchange local search algorithm based on the disjunctive graph model of Roy & Sussmann (1964) was used. The same neighborhood was used in Binato et al. (2001) for a Job Shop Scheduling problem. In order to apply the 2-exchange local search to the WDP, we swap two elements in the scheduling. For example, if in resource X the scheduling was ... $\rightarrow X1 \rightarrow A \rightarrow X2 \rightarrow$ ... and in resource Y it was ... $\rightarrow Y1 \rightarrow B \rightarrow Y2 \rightarrow$ ..., where $\rightarrow$ represents a conjunctive arc, the result of the swap would be a schedule like ... $\rightarrow X1 \rightarrow B \rightarrow X2 \rightarrow$ ... and ... $\rightarrow Y1 \rightarrow A \rightarrow Y2 \rightarrow$ ..., in resources X and Y, respectively. Since the execution time of elements A and B can be different, all activities after them may have their start times updated.

We need to decide, of course, what an element stands for. Some options are: (i) *An activity*: with very small granularity, giving rise to huge neighborhoods (do Nascimento (2002)), and, worse, moving an activity to another position would possibly force us to move also its predecessors and successors in the same well, because of constraints C10; or (ii) *A well*: with higher granularity, but since the sequence of activities in a well may be splited in the present schedule due to constraints C1, some problems now being that moving all activities takes time to verify all constraints, and exchanging the whole well may not be possible even though exchanging only part of it could be; or finally (iii) *Part of a well*: (that is, a maximal set of activities of the same well scheduled consecutively in the same resource) with medium granularity and already satisfying constraint C10.

In our implementation we chose the last possibility, where the local search algorithm exchanges all pairs of parts of two wells, no matter on what resource they have been scheduled. That neighborhood is of size $O(n^2)$, where n is the number of parts of wells. For practical instances, this is one order of magnitude smaller than the neighborhood that uses activities as the moving elements.

To fully specify the local search phase we need a rule that defines how the neighborhood is searched and which solution replaces the current one. This rule is called the *pivoting rule* (Yannakakis (1997)), and examples of it are the *first improvement rule* (FIR) and the *best improvement rule* (BIR). In the first case, the algorithm moves to a neighboring solution

as soon as it finds a better solution; in the second case, all neighbors are checked and the best one is chosen. In either case, the worst case running time of each iteration is bounded by $O(n^2)$, where n is the number of elements in the neighborhood. In the next section we present a comparison between these two alternatives.

4 Computational Results

In this section, computational results for the GRASPW implementation are given. They are also compared with results obtained with the ORCA implementation over the same real instances. All tests were run on a platform equipped with a Sun SPARC Ultra 60 processor running a Solaris 9 operating system at 450 MHz and with 1024 MB of RAM. Both GRASPW and ORCA were allowed to run for 1800 seconds on each instance.

Typical Instances. Twenty one real instances provided by Petrobras were used in our tests. Table 1 summarizes the dataset. Columns with the same numerical data refer to distinct instances that differ in the number of other constraints, like C9. The first part of that table displays the instances were no C7 constraints were found. In order to reduce the amount of time spent in testing, in some experiments we used only 7 of these instances, eliminating instances that differed only in the number of clusters (see constraint C9, in Section 2). The lower part shows the nine instances were C7 constraints were present. The ORCA implementation had difficulties to handle these contraints.

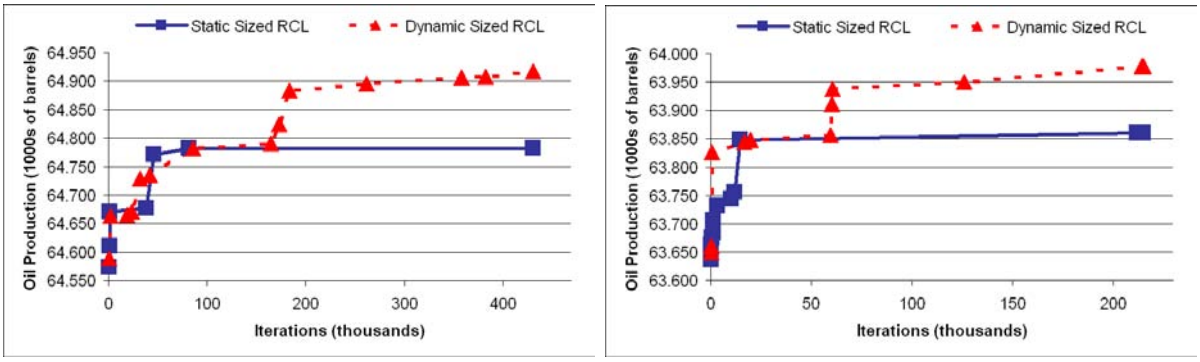
Instance	1	2	3	4	5	6	7	8	9	10	11	12
# wells	29	22	29	29	17	22	22	29	29	22	29	22
# activities	98	107	98	98	111	107	128	98	98	107	98	107
# boats	1	1	2	1	1	2	1	1	1	2	1	2
# derriks	3	2	3	3	2	2	3	3	3	2	3	2
# C7 constr.	0	0	0	0	0	0	0	0	0	0	0	0

Instance	13	14	15	16	17	18	19	20	21
# wells	22	22	22	22	22	22	22	22	29
# activities	107	107	107	107	107	107	107	107	98
# boats	2	2	2	1	2	2	2	2	1
# derriks	2	2	2	2	2	2	2	2	3
# C7 constr.	1	1	1	1	1	1	1	1	2

Table 1: Test instances.

Setting GRASPW Parameters. In Section 3 we presented the idea of a *dynamic sized RCL*. There are at least two ways we could exploit this idea: we may decrease the number

of candidates through time, using a greedier heuristic; or we may increase the number of candidates through time, in order to drive away from a local optimum into new regions of the search tree. In the first case, the initial RCL size is set to $\max(13, w)$, w being the number of wells, and is decreased by one every 300 seconds without improvement. In the second case, we start with $\max(5, w)$ for the initial RCL size and increase it by one every 300 seconds without improvement. The first approach did not yield good results when applied to the WDP, generating the same or worse solutions than those found by GRASPW with a static sized RCL. However, the second approach proved promising. Figure 3.a shows the algorithm with dynamic sized RCL generating better solutions after 150 thousand iterations, when the RCL is increased. The same happens in Figure 3.b after 50 thousand iterations, when another real instance is tested. Amongst twelve scenarios tested, four had better solutions with the dynamic sized RCL, totalizing an increase of around 261 thousand barrels of oil. In the other eight scenarios, the same solutions were found.



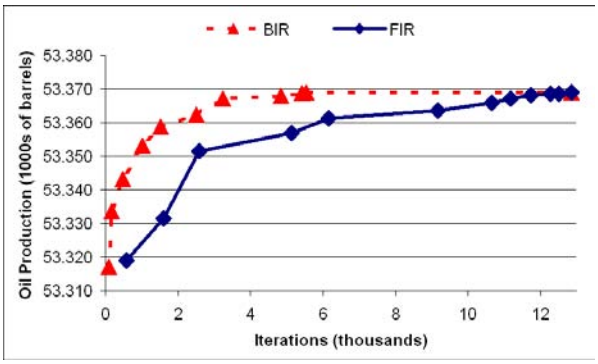
(a) Instance 1 (b) Instance 2

Figure 3: Static Sized RCL x Dynamic Sized RCL.

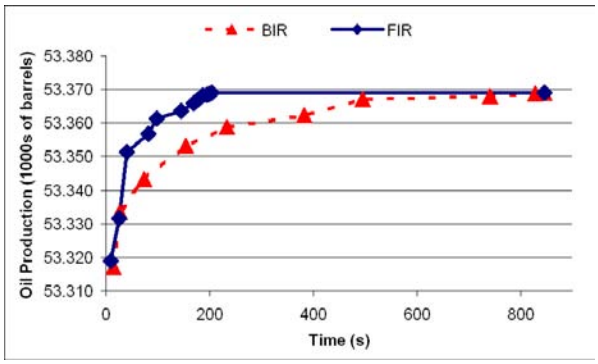
Another technique tested was to bias the selection towards some particular candidates, those with the highest oil yield, but this strategy did not produce good results. Amongst twelve instances tested, three had slightly worse solutions with such a bias function, totaling a decreasing of 40 thousand barrels of oil. Worse, with the bias function in place, the algorithm takes much more time to find the same solution than when no bias is used. Summing up all the differences in time for the twelve scenarios, with the bias function the algorithm took 3431 more seconds to reach the same results, an average increase of 72%.

We also considered two options for searching the neighborhood and selecting a new neighbor: the *first improvement rule* (FIR) and the *best improvement rule* (BIR). We tested both of them on seven instances. The BIR heuristics proved to be the best one when finding solutions whose production was equal to a predefined target value and with the

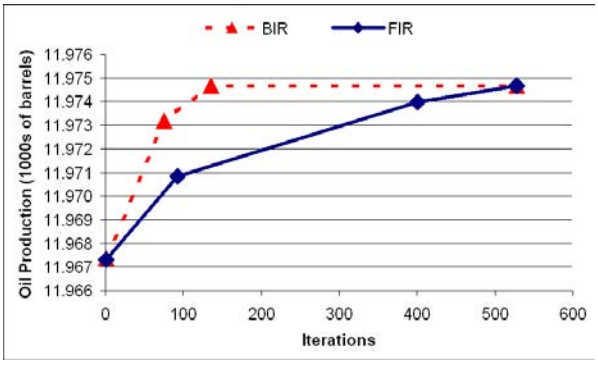
least number of iterations (see Figure 4 (a) and (c)). On average, to find a solution with a predefined production, the BIR strategy used about 60% of the number of iterations of the FIR strategy. On the other hand, the FIR strategy was faster in most instances (see Figure 4 (b) and (d)). On average, to find a solution with a predefined production, the FIR strategy used 66% of the time used by the BIR strategy. That is because, on average, a FIR iteration was almost seven times faster than a BIR iteration. Since, to users, running time was deemed important, the FIR strategy was found to better suit this problem.



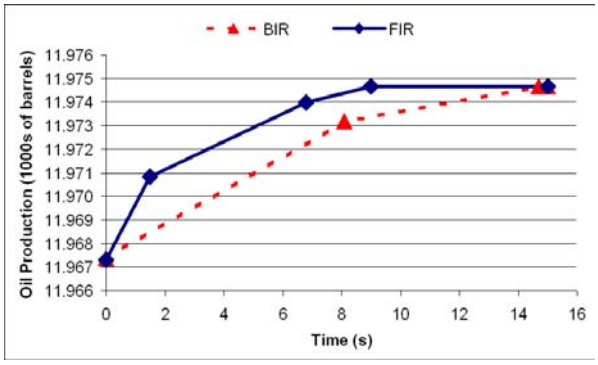
(a) Instance 1 - BIR



(b) Instance 1 - FIR



(c) Instance 2 - BIR

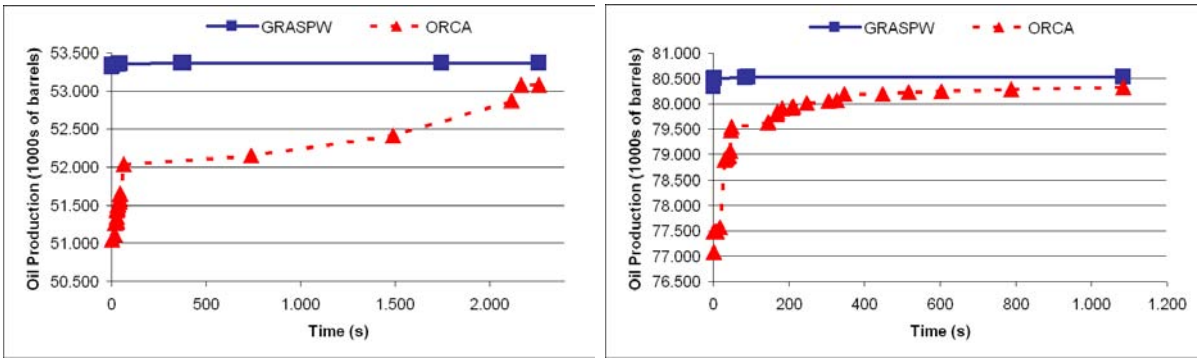


(d) Instance 2 - FIR

Figure 4: BIR × FIR.

The ORCA and the GRASPW Implementations. Our GRASPW implementation found better solutions than the ORCA implementation on all twelve but one of the real instances tested and, in that one, it found the same solution. GRASPW achieved 0.14% more oil production on average which, despite being a small percentage, means an increase of almost one million barrels of oil, in total. Perhaps, the highest gain with GRASPW was in running time. It found solutions with the same production of those generated by ORCA,

on average, in only 36.3% of the time used by ORCA on the same instances. Figure 5.a shows that the best GRASPW solution has a production 290 thousand barrels of oil higher than the best ORCA solution. Furthermore, GRASPW found a solution with the same oil production of the best ORCA solution within the first second, while ORCA found it only after 2200 seconds. Similarly, Figure 5.b shows that the best GRASPW solution has a production of almost 200 thousand barrels of oil higher than the best ORCA solution. Again, GRASPW found a solution with the same oil yield as the best ORCA solution within the first second, while ORCA found it only after 1000 seconds.



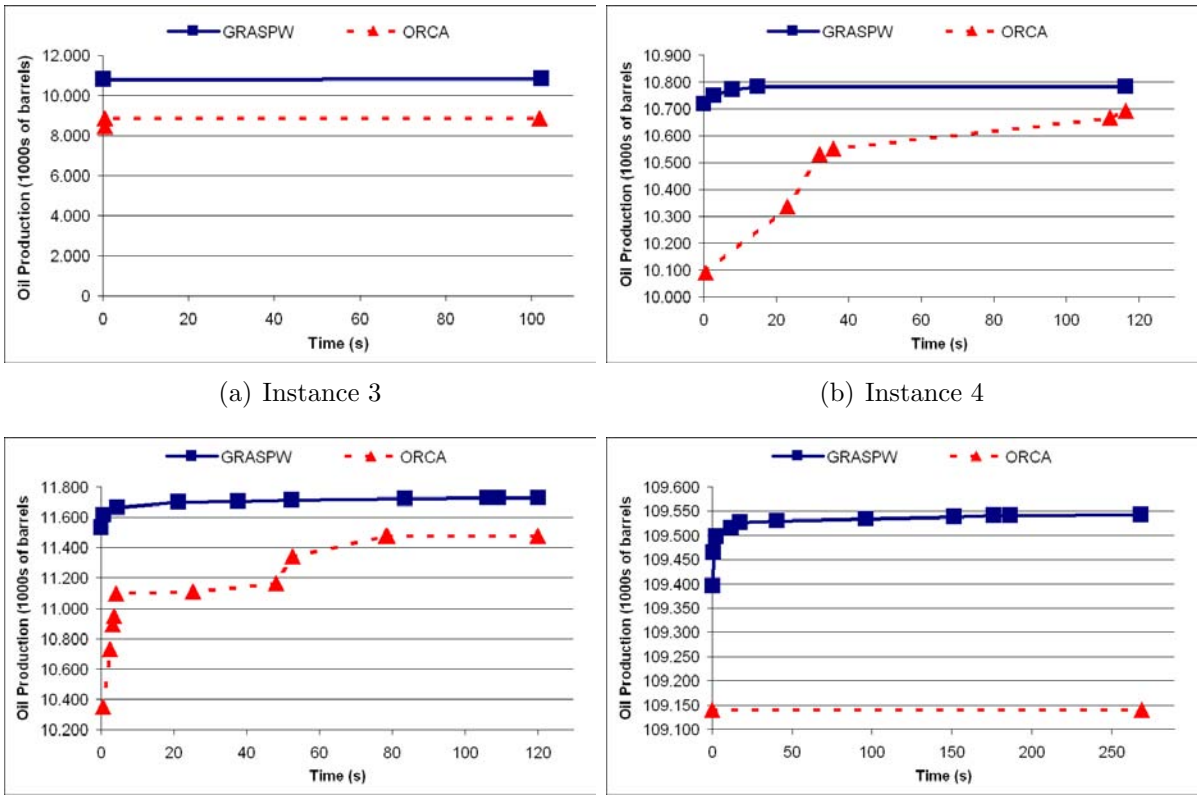
(a) Instance 1 (b) Instance 2

Figure 5: ORCA x GRASPW.

In all the nine tested scenarios, where the constraints C7 were present, GRASPW found better solutions than ORCA. In one of them, ORCA could not find any solution, while GRASPW found one with about 26 million barrels of oil production. Comparing the other eight scenarios, GRASPW achieved 5.3% more oil production, on average, which means an increase of around 4.6 million barrels of oil, in total. In Figure 6 we present the results over four such instances. In all of them GRASPW was more effective than ORCA.

5 Conclusions

Scheduling activities efficiently is of paramount importance to the industry, in general. Petrobras, a leading company in deep water exploration of oil, presented us the WDP, a scheduling problem related to oil well drilling. Here we constrast two approaches to the WDP: the constraint programming tool ORCA and a GRASP implementation, dubbed GRASPW. Computational experiments were carried out on several real instances. We conclude that GRASPW greatly outperforms ORCA. Not only it generates solutions with



(a) Instance 3

(b) Instance 4

(c) Instance 5

(d) Instance 6

Figure 6: ORCA x GRASPW.

higher oil production, but often it outputs solutions with the same oil production as ORCA in much less time. We also note that ORCA already produced better results than the manual solutions.

It is worth mentioning that ORCA is built over the ILOG Constraint Programming suite, a set of highly expensive and sophisticated libraries with years of development. Using GRASPW, which was entirely programmed from the ground up, these costs could be averted. In opposition, the ILOG suite favors easiness of development, of maintenance and of understanding of the source code.

Acknowledgements The third author was supported by grants 307773/2004-3 from CNPq and 03/09925-5 from FAPESP.

6 References

Aiex, R. M., Binato, S. and Resende, M. G. C., 2003, 'Parallel GRASP with path-relinking for job shop scheduling', *Parallel Computing* **29**(4), 393–430.

Bard, J. F. and Feo, T. A., 1989, 'Operations sequencing in discrete parts manufacturing', *Management Science* **35**, 249–255.

Binato, S., Hery, W. J., Loewenstern, D. and Resende, M. G. C., 2001, A GRASP for job shop scheduling, *in* P. Hansen and C. C. Ribeiro, eds, 'Essays and surveys on metaheuristics', Kluwer Academic Publishers, pp. 59–79.

do Nascimento, J. M., 2002, Hybrid computational tools for the optimization of the production of petroleum in deep waters, Master's thesis, Institute of Computing, University of Campinas.

Feo, T. A. and Bard, J. F., 1989, 'Flight scheduling and maintenance base planning', *Management Science* **35**, 1415–1432.

Feo, T. A., Bard, J. and Holland, S., 1995, 'Facility-wide planning and scheduling of printed wiring board assembly', *Operations Research* **43**, 219–230.

Feo, T. A. and Resende, M. G. C., 1995, 'Greedy randomized adaptative search procedures', *Journal of Global Optimization* **6**, 109–133.

Festa, P. and Resende, M. G. C., 2001, GRASP: An annotated bibliography, Technical report, AT&T Labs.

Glover, F. and Laguna, M., 1997, *Tabu Search*, Kluwer Academic Publishers, Norwell, Massachusetts.

ILOG, 1999, *ILOG Solver 4.4 Reference Manual*, ILOG.

Marriott, K. and Stuckey, P. J., 1998, *Programming with Constraints: An introduction*, MIT Press, Cambridge, Massachusetts.

Roy, B. and Sussmann, B., 1964, Les problèmes d'ordonnancement avec contraintes disjonctives, *in* 'Note DS No 9 bis', SEMA, Paris.

Yannakakis, M., 1997, Computational complexity, *in* E. H. L. Aarts and J. K. Lenstra, eds, 'Local Search in Combinatorial Optimization', John Wiley & Sons, Chichester, pp. 19–55.

Epílogo

Todos os testes realizados tiveram um limite de tempo de execução de 1800 segundos, conforme já informado no artigo. Entretanto, engenheiros da Petrobras ao usar um resolvedor para o problema fixam um limite de tempo de apenas 120 segundos. Isso ocorre porque esses engenheiros realizam diversas análises sobre uma mesma instância, ou seja, a testam retirando/adicionando/trocando restrições e recursos, avaliando por exemplo a necessidade ou benefícios de se adquirir novas sondas ou barcos. Para poder executar um bom número de análises é preferível usar um limite de tempo baixo, tendo em vista que o tempo desses engenheiros é limitado. É por isso que comparamos não só a solução final dos resolvedores, como também as curvas de produção gerada das soluções pelo tempo, e preferimos resolvedores que convergem rapidamente para boas soluções.

Já quanto ao tempo em que se dá a alteração do tamanho da RCL, o fixamos em 1/6 do tempo total, ou 300 segundos sem melhoras na melhor solução, conforme já informado no artigo. Fixar um tempo não é o ideal, pois ao se migrar para plataformas de *hardware* superior, esse tempo poderia se tornar excessivo. Assim, o que poderia ser feito é fixar essa alteração pelo número de iterações GRASP proporcional ao limite de tempo fornecido. Tal não foi feito pois para se comparar de forma mais justa as técnicas BIR e FIR, queríamos um fator igual para alteração no tamanho da RCL. Se utilizássemos o número de iterações, um resolvedor com BIR, em que as iterações são mais lentas que um com FIR, teria sua RCL alterada bem depois, prejudicando-o quando comparado a um resolvedor com FIR. Ao se disponibilizar a ferramenta GRASP para a Petrobras, a alteração do tamanho da RCL deverá ser feita após um número de iterações sem melhorias na melhor solução, sendo esse número proporcional ao limite de tempo fornecido.

Um ponto importante não mencionado no artigo é que, como o GRASP faz uso de randomização, cada instância é testada cinco vezes, e os resultados reportados são sempre uma média dos testes. Poder-se-ia fazer uma análise do desvio padrão das produções das melhores soluções geradas pelo GRASP, verificando formalmente sua robustez, mas tal não se mostrou necessário. Isso porque em todos os testes realizados com um mesmo resolvedor em uma mesma instância, a diferença entre as produções de soluções diferentes nunca superou os 0.5%. Dessa forma, um mesmo resolvedor GRASP mostra uma variação muito pequena entre a qualidade de suas soluções para uma mesma instância.

Outro ponto que vale a pena ressaltar é que a vizinhança utilizada no GRASP também poderia ser utilizada numa Busca Tabu. A Juliana em seu trabalho (do Nascimento 2002) teve dificuldades na definição de uma vizinhança pois usou uma vizinhança de larga escala, em que desenvolveu uma técnica híbrida entre programação por restrições e Busca Tabu. A escolha da utilização da metaheurística GRASP nessa tese se deveu principalmente a sua menor quantidade de parâmetros a serem ajustados.

Apesar da ferramenta ORCA ter se apresentada robusta e eficiente, gerando ganhos expressivos à Petrobras, nosso trabalho conseguiu gerar uma aplicação GRASP que produziu resultados ainda melhores que aqueles obtidos pelo ORCA. Diante do fato, obtivemos apoio da Petrobras para atacar uma versão estendida do EDP, agora considerando os deslocamentos de recursos, como é mostrado no capítulo a seguir.

Capítulo 7

O EDP com Deslocamento de Recursos

Prólogo

Com o sucesso obtido com o GRASP no EDP, a Petrobras se comprometeu a nos fornecer informações para tratar também o deslocamento dos recursos, o que aproximaria ainda mais os escalonamentos gerados com os realmente executados na prática. Dessa forma, surgiu o EDPDR, uma extensão importante do EDP.

O artigo a seguir trata o problema de escalonamento no desenvolvimento de poços de petróleo em alto mar com deslocamento de recursos (EDPDR). O texto está em inglês, transcrevendo na íntegra o artigo submetido ao *Journal of Heuristics*.

Após uma descrição detalhada do problema e das restrições envolvidas, é apresentado um resolvedor GRASP que supera uma ferramenta de programação por restrições desenvolvida pela Petrobras. Como o problema se tornou bem mais complicado, o que prejudicou a performance da ferramenta ORCA e a qualidade de suas soluções, incorporamos técnicas mais avançadas ao GRASP para aumentar ainda mais os ganhos obtidos em relação ao ORCA.

O estudo do EDPDR resultou em três publicações:

- Relatório Técnico: Pereira et al. (2005*c*), emitido pelo Instituto de Computação da Unicamp.

- Prêmio Petrobras de Tecnologia: O artigo, traduzido para o português, foi o terceiro colocado dentre 25 inscritos na categoria *Produção*, recebendo premiação em dinheiro e uma bolsa de doutorado. Para mais detalhes, vide o site:
  http://www2.petrobras.com.br/tecnologia2/port/pptecnologia.asp.

- Journal of Heuristics: O artigo foi submetido e espera o retorno dos avaliadores. Vide: http://www.kluweronline.com/issn/1381-1231.

GRASP Strategies for Scheduling Activities at Oil Wells with Resource Displacement

Romulo A. Pereira
romulo_a_pereira@yahoo.com.br

Arnaldo V. Moura
arnaldo@ic.unicamp.br

Cid C. de Souza
cid@ic.unicamp.br

Institute of Computing, University of Campinas

Abstract

Before promising locations at petroliferous basins become productive oil wells, it is necessary to complete development activities at these locations. The scheduling of such activities must satisfy several conflicting constraints and attain a number of goals. Moreover, resource displacements between wells are also important. We describe a Greedy Randomized Adaptive Search Procedure (GRASP) for the scheduling of oil well development activities with resource displacement. The results are compared with schedules produced by a well accepted constraint programming implementation. Computational experience on real instances indicates that the GRASP implementation is competitive, outperforming the constraint programming implementation.

Today, oil and gas are fossil fuels of wide use in our society. They are important ingredients in the processes of making plastics, dyes, kerosene, gasoline, gas and many other products. A significant amount of these fossil fuels is extracted from oceanic basins, *e.g.*, from the offshore Marlin basin in Rio de Janeiro, Brazil. Petrobras is a company with recognized expertise in oil exploration in deep sea waters, being also one of the twenty biggest oil companies in the world. Usually, Petrobras explores diverse petroliferous basins, each with hundreds of promising spots where productive oil wells could be located. However, before these places are turned into productive wells they must be developed, that is, a sequence of engineering activities must be completed at each promising spot, to render them ready for oil extraction. Oil derricks and ships are used to complete these activities. These resources have to move from one spot to another and such displacements must be considered when sequencing the activities. Furthermore, such resources are limited and expensive, either in acquisition or rent value, and must be used efficiently.

The oil *well development with resource displacement problem* (WDRDP) can be summarized thus: given a set of promising spots, the activities to be executed at each location,

and the available resources, find a scheduling of the activities and resources, fulfilling several conflicting engineering and operational constraints, including the displacements of the resources, in such a way as to optimize some objective criteria. In this work, the specific WDRDP faced by Petrobras is studied.

This WDRDP imposes much more realistic constraints than other similar studies (do Nascimento (2002), Pereira et al. (2005)). Actually, Pereira et al. (2005) is an early version of this paper, which did not consider the displacement of the resources. Here, we will take this special characteristic into account. In addition, in this paper, we discuss other advanced techniques not explored in Pereira et al. (2005).

In the sequel, the constraints are presented in detail and our GRASP heuristic strategies are developed, maximizing oil production within a given time horizon. The GRASP implementations were built and tested over several real instances provided by Petrobras. The best implementation indicated a gain of almost 16 million barrels of oil, or more than US$ 830 millions, summing up over the average solutions of all the instances, when compared to solutions generated by the constraint programming scheduler in use by the company. As another advantage, our methods rely on non commercial libraries, whereas the constraint programming implementation relied on expensive third party constraint programming solvers.

The next section describes the WDRDP. Section 2 discusses GRASP implementations for the WDRDP. Section 3 presents our computational results and compares them to other results derived from the constraint programming implementation presently running at Petrobras. Finally, some concluding remarks are offered in the last section.

1 Scheduling the Development of Oil Wells

When a spot is considered a promising oil well, oil derricks are sent there to accomplish the due *drilling* operations. After a well is drilled, the preparation for oil extraction develops in several stages.

First, in the stage called *completion*, oil derricks place Wet Christmas Trees (or WCTs, structures where hydraulic valves are attached) at the mouth of the wells in order to avoid oil leakage. Later, boats connect pipelines between WCTs and manifolds, this stage being called *interconnection*. Manifolds are metallic structures installed by boats at the sea floor. Their use prevents the need for exclusive pipelines connecting each well to the surface, which would be prohibitively expensive. Once this stage is completed, oil extraction can begin. For that, Stationary Units of Production (SUPs) are anchored at specific locations in the surface and boats interconnect manifolds to them. SUPs are used to process, and possibly store, the extracted products. Later, ships fetch the products from SUPs to land storage sites or other processing units. If the oil outflow is very high or a SUP does not

have storage capacity, a petroliferous platform may be installed at the surface.

There are two types of wells that may be developed: (i) *productive wells*, which are those who have an oil yield; and (ii) *injection wells*, where only maintenance activities are executed and oil is not extracted from them.

1.1 Oil Yield and Objectives

The oil yield is calculated as follows. Each well has an associated outflow and an activity that marks the beginning of its production. When this last activity is concluded, the well is considered in production. The yield is obtained by multiplying the oil outflow by the period between that instant and the established time horizon of production. If the start production activity is set for a time after the horizon, the corresponding yield is disregarded. The objective is to obtain a scheduling of all activities, satisfying all constraints, while maximizing the oil yield.

Other goals to be attained by automating the scheduling of the activities are:

1. *Faster solutions*. Human made solutions take many hours, even days, to be constructed. A faster method would permit the analysis of different scenarios for the same problem, for example, by adding or removing resources. Furthermore, modifications in already committed plans would not result in new hours, or days, spent in rescheduling.

2. *Better resource allocation*. With an automated scheduling, all highly skilled engineers responsible for the manual scheduling can receive other duties.

3. *Savings*. The automated schedule will, usually, result in a better use of boats and derricks, thereby saving considerable operational resources.

1.2 The WDRDP Constraints

The constraints involved in the scheduling of oil well development activities are:

C1. *Technological Precedence*: sets an order between pairs of activities. When considering the precedence between the start and finish of the activities in each pair, any of the four possible combinations can be present.

C2. *Mark-Activity*: an activity must finish before or initiate after a fixed date, or *mark*, with or without lag time. This date is often related to some external event, *e.g.*, the installation of a petroliferous platform.

C3. *Baseline*: sets the start date of the activities.

C4. *Use of Resources*: to execute an activity, due to its intrinsic nature, a resource used must match some operational characteristics. For a boat, it must be verified if the on-board equipments can operate at the specified depth. For an oil derrick, its type and capabilities must be verified, as well as its maximum and minimum depth of operation and drilling.

C5. *Concurrence*: two activities at the same well, or executed by the same resource, can not be simultaneous.

C6. *Unavailability*: resources may be unavailable for a period of time, either for maintenance reasons or due to contract expiration.

C7. *User Defined Sequences*: the user can specify a sequence for the drilling activities or for the "start production" activities of different wells, depending to the type of this sequence. These sequences are specified by engineers in order to avoid loss of pressure in the oil field. If well A appears before well B in the sequence, then the activity of well A must terminate before the start of activity of well B can be scheduled.

C8. *Surface Constraints*: represented by a polygonal security area defined around a well. When the center of a well is inside the restricted area of another well, activities executed at both wells cannot be simultaneous. These constraints must be verified between pairs of mobile and pairs of mobile and anchored oil derricks.

C9. *Cluster Constraints*: an activity may be part of a cluster, which is a set of activities that must use the same resource.

C10. *Oil Derrick Displacements*: when an oil derrick moves between two wells, a set-up time[1] will be considered. Therefore, unnecessary displacements must be avoided, for example, by making the same oil derrick execute as much activities at a well as possible. For more details, see Subsection 1.4.

From the above description, it can be seen that the WDRDP is a difficult combinatorial optimization problem. In fact, it is simple to devise a polynomial-time reduction from the classical *Job Shop Scheduling Problem (JSP)* to the WDRDP, thus showing that the WDRDP is NP-hard. The WDRDP treated here shows several differences from similar problems studied in the literature (do Nascimento (2002), Pereira et al. (2005)).

[1]Time to unanchor, move and anchor in a new place.

1.3 The Constraint Programming Solution

To tackle the same problem, a project team from Petrobras developed a Constraint Programming (cf., Marriott & Stuckey (1998)) model using ILOG's Solver and Scheduler[2] (ILOG (1999*b*)). After four years of development and testing, the tool, named ORCA[3], became operational and very successful. Nowadays, the ORCA solver is often used by engineers both to define a good scheduling for the drilling activities and, also, to analyze the need for acquiring or renting new resources. They confirmed that ORCA generates better solutions than those made by humans. In one real instance, ORCA showed that buying a third oil derrick was unnecessary and that it was better to add a new LSV ship instead. As a result, an expenditure of US$ 15 million was avoided, while anticipating oil production by 26 days. Despite the good performance of ORCA, searching for even better solutions is still important, since a tenth of a percent improvement in the oil production may represent an increase of millions of dollars in the company's revenue.

1.4 The Resource Displacement

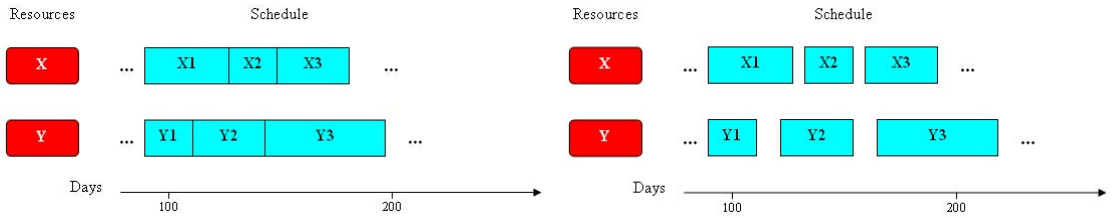
As already mentioned, here we will deal with a new facet of the problem, the resource displacement. This constraint forces a period of "inactivity" for the resource, so that it can unanchor, travel between two wells and anchor in the new spot. We call this period the set-up time. It is set between activities of different wells which are scheduled consecutively. In Figure 1.a we see a typical schedule with no resource displacement. Each square in the "Schedule" area represents a different well. A square in dark gray represents a group of activities of the same well which are executed by the resource, the light gray square, which is in the same horizontal line. In Figure 1.b, we present a schedule with resource displacement. Note that between each well there is a void period of time, which is the set-up time.

The set-up time is a constant for each resource. It is calculated by engineers, based on the average speed of the resource and the average distance between spots in the field, plus the average time to anchor and unanchor. As engineers noted when scheduling the activities at wells of an oil basin, the average distance between these wells does not differ much from the average distance traveled by the resources. The weather, which influences the speed of the resource, was also considered in the average speed.

However, different resources may have different set-up times. Consider, for example, resources X and Y in Figure 1.b. Note that the set-up time of Y is twice longer than the set-up time of X. This may alter the selection of which resource will execute activities at a well. For example, if both resources X and Y are available at the same instant and

[2]Registered trade marks of ILOG Inc.

[3]Portuguese acronym for "Optimization of Critical Resources in the Production Activity".



(a) No Displacement (b) With Displacement

Figure 1: Schedules without and with resource displacement.

can execute the next well to be scheduled, it does matter which resource we choose. This would not be the case if both resources had the same set-up times. Thus, a good selection of which resource will execute the activity is even more important when resource displacement is considered.

The constraint programming solver had many difficulties when treating the resource displacement. A first approach used created virtual activities to represent the set-up time. But, since as it is not known in advance if an activity from another well will be scheduled consecutively, it would be necessary to create virtual activities dynamically, during the search for solutions. Of course they would have to be destroyed when backtracking. This process was quite complex and slowed down the search considerably, and so this approach was abandoned. Instead, an *if-then constraint* was set, imposing that if the previous activity executed by the resource was from another well, then the current activity was shifted by the corresponding set-up time. Of course, as this constraint needed to hold between each pair of activities, the number of constraints added to the problem was $n * (n-1)/2$, where n is the number of activities. This increase in the number of constraints also slowed down the execution of ORCA, and it did not generate very good solutions for many instances.

2 GRASP Strategies for the WDRDP

Our search for alternatives to compete with ORCA started with an implementation of a tabu search strategy (Glover & Laguna (1997)) for a simpler version of the WDRDP (do Nascimento (2002)). However, some issues proved to be particularly difficult to treat, especially the definition of an adequate neighborhood and ways to explore it. After some investigation, the GRASP (Feo & Resende (1995)) method seemed to be a more appropriate approach for the WDRDP. Contrary to what occurs with other meta-heuristics, such as tabu search or genetic algorithms which use a large number of parameters in their implementations, the

basic GRASP version requires the adjustment of fewer parameters. Despite its simplicity, GRASP is a well studied meta-heuristic which has been successfully applied to a wide variety of optimization problems (cf. Festa & Resende (2002)). In particular, applications of GRASP to scheduling problems can be found in Bard & Feo (1989), Feo & Bard (1989), Feo et al. (1995), Bard et al. (1996) and Binato et al. (2002). It is worth mentioning that GRASP proved itself very valuable in dealing with the well development problem without resource displacement (Pereira et al. (2005)).

The next paragraphs review some GRASP basics and describe our specific implementation designed to solve the WDRDP, named GRASP-WDRDP (or GRASPW, for short). In the sequel, we present some advanced GRASP techniques that were used to improve our solver. The model and its algorithms are also shown in detail in the subsequent paragraphs.

2.1 Greedy Randomized Adaptive Search Procedure (GRASP)

In the GRASP methodology each iteration consists of two phases: *construction* and *local search* (Feo & Resende (1995)). Figure 2 illustrates a generic implementation of GRASP, in pseudo-code. The input includes parameters for setting the candidate list size ($ListSize$), the maximum number of iterations ($MaxIter$), and the seed ($Seed$) for the random number generator. The iterations are carried out in lines 2-6. Each iteration consists of the construction phase (line 3), the local search phase (line 4) and, if necessary, the incumbent solution update (line 5). In the construction phase, a feasible solution is built, updating the variable $Solution$. Then the local search algorithm seeks a better solution in the neighborhood of $Solution$, according to a given criterion, and updates $Solution$. This process of construction, search and update is executed $MaxIter$ times.

```
1: procedure GRASP(ListSize, MaxIter, Seed)
2: for k = 1 to MaxIter do
3:    Solution ← Construct_Solution(ListSize, Seed);
4:    Solution ← Local_Search(Solution);
5:    Update_Solution(Solution, Best_Solution_Found);
6: end for
7: return Best_Solution_Found;
8: end GRASP
```

Figure 2: Pseudo-code of the GRASP Meta-heuristic.

In the construction phase, a feasible solution is built one element at a time. Figure 3 illustrates a generic implementation of the construction phase, in pseudo-code. Input includes the candidate list size ($ListSize$) and the seed ($Seed$). The iterations are carried out in lines 2-8. At each iteration, the next element is selected from all possible elements

added to the candidate list. These elements are ordered with respect to a greedy function that measures the, maybe myopic, benefit of selecting each element. This list is called the *Restricted Candidate List* (RCL). The adaptive component of the heuristic arises from the fact that the benefits associated with every element are updated at each iteration to reflect the changes brought on by the selection of the candidate in the previous iteration. The probabilistic component is present by the random choice of one of the best candidates in the RCL, but usually not the best one. This way of choosing elements allows for different solutions to be obtained at each iteration, while not necessarily jeopardizing the adaptive greedy component.

The solutions generated by the construction phase are not guaranteed to be locally optimal. Hence, it is almost always beneficial to apply a local search procedure to improve the constructed solution. The search phase is a standard deterministic local search algorithm that seeks to optimize the solution built in the construction phase.

1: procedure $Construct\_Solution(ListSize, Seed)$
2: $Solution \leftarrow 0$;
3: *Evaluate the incremental costs of the candidate elements*;
4: **while** *Solution is not a complete solution* **do**
5: *Build the restricted candidate list*, $RCL(ListSize)$;
6: *Select an element s from the RCL at random*;
7: $Solution \leftarrow Solution \cup \{s\}$;
8: *Reevaluate the incremental costs*;
9: **end while**
10: *return Solution*;
11: *end Construct_Solution*

Figure 3: Pseudo-code of the Construction Phase of GRASP.

2.2 GRASP Advanced Techniques

As the problem increased in complexity, compared to the problem with no resource displacement studied in Pereira et al. (2005), we decided to consider some improvements and alternative techniques to be introduced in the basic GRASP procedure.

Bias Function: In the construction procedure of the basic GRASP, the next element to be introduced in the solution is chosen at random from the candidates in the RCL. The elements of the RCL are assigned equal probabilities of being chosen. However, any probability distribution can be used to bias the selection toward some particular candidates. Bresina (1996) proposed a construction mechanism based on the *rank* $r(\alpha)$ assigned to each candidate element α, according to its value, measured by the

greedy function, $v(\alpha)$. However, as the sorting of elements to obtain the *rank* has time complexity $O(n \log n)$, where n is the number of elements in the RCL, and since the sorting must be repeated at each choice of an element, this process could slow down the implementation. Thus, we decided to use directly the value $v(\alpha)$ of the candidates to create the following bias functions:

- uniform: $bias(\alpha) = 1$
- linear: $bias(\alpha) = v(\alpha)$
- log: $bias(\alpha) = \ln v(\alpha)$
- exponential: $bias(\alpha) = e^{v(\alpha)}$
- quadratic: $bias(\alpha) = v(\alpha)^2$
- square root: $bias(\alpha) = \sqrt{v(\alpha)}$

Once the value of the bias function is evaluated for all elements of the RCL, the probability of the candidate α being chosen is:

$$\frac{bias(\alpha)}{\sum_{\alpha' \in RCL} bias(\alpha')}$$

Proximate Optimality Principle (POP): This technique is based in the idea that "good solutions at one level are likely to be found 'close to' good solutions at an adjacent level" (Glover & Laguna (1997)). Fleurent & Glover (1999) provided a GRASP interpretation of this principle. They suggested that imperfections introduced during steps of the GRASP construction phase can be ironed out by applying local search during (and not only at the end of) the GRASP construction phase. Due to efficiency considerations, a practical use of the POP in a GRASP implementation would be to apply a local search during a few points in the construction phase, and not at the end of each construction iteration. Binato et al. (2002), when dealing with a JSP, applied a local search when 40% and 80% of the construction moves have been taken, as well as at the end of the construction phase. Due to the relation between the JSP and the WDRDP (see Section 1.2), and after some promising tests, the same values were used in our GRASP solver.

One possible shortcoming of the basic GRASP method is the independence of its iterations, *i.e.*, the fact that it does not learn from the history of solutions found in previous iterations. This is so because the standard algorithm discards information about any solution encountered that does not improve the incumbent solution. Information gathered from good solutions can be used to implement memory-based procedures. We show some of these strategies below.

Intensification: Fleurent & Glover (1999) observed that the standard GRASP does not use long term memory and proposed a scheme to use this kind of memory in the heuristic. Long term memory is one of the pillars of tabu search. The idea of the approach is to maintain a set of elite solutions. To be included in this set, a solution must either be better than all elite solutions according to some objective function, or must be better than the worst solution of the set, while being sufficiently different from all elite solutions. In our problem, a solution is considered sufficiently different from another if the number of activities that have different start dates in both, or the number of different resources allocated to both is higher than the number of activities divided by the number of resources. As the number of resources is usually smaller than five, two solutions will be sufficiently different if at least 20% of their activities have a different start date or resource allocated to both.

We show next how the elite solutions will bias the selection of the candidates in the GRASP construction phase. To each candidate α we evaluate the intensity function $Int(\alpha)$ as follows:

$$Int(\alpha) = \frac{\sum_{e_i \in S} Prod(e_i)}{Max_{e_j \in E} Prod(e_j)}.$$

In this function, S is the set of elite solutions in which the element α has the same start date and resource allocated when compared to the solution being built; $Prod()$ is the value of oil yield associated with a solution, E is the set of elite solutions, and Max returns the maximum value in the indicated set.

Let $iter$ be the amount of iterations executed and k a parameter, we define the bias function using the intensity function as follows:

$$biasInt(\alpha) = bias(\alpha) + \frac{Int(\alpha) \cdot iter}{k}$$

The fraction $iter/k$ is used to give emphasis to the intensity function as the number of iterations increases, and thus the quality of the elite solutions in which the intensity function is based also possibly increases.

Path-Relinking (P-R): This technique was originally proposed in Glover (1996) as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search methods (Glover (2000), Glover & Laguna (1997), Glover et al. (2000)). Starting from one or more elite solutions, paths in the solution space leading towards other elite solutions are generated and explored in the search for better solutions. This is accomplished by selecting moves that introduce, in the

moving solution, attributes contained in the target solution. Path-relinking may be viewed as a strategy that seeks to incorporate attributes of high quality solutions, by favoring these attributes in the selected moves.

The use of path-relinking within a GRASP procedure was first proposed in Glover et al. (2000). It was followed by several extensions, improvements and successful applications (Aiex et al. (2000), Canuto et al. (2001), Resende & Ribeiro (2001)). Two basic strategies are:

- path-relinking is applied as a post-optimization step to all pairs of elite solutions.
- path-relinking is applied as an intensification strategy to each local optimum obtained after the local search phase.

According to Resende & Ribeiro (2002) the second strategy is more promising. Another consideration, also from Resende & Ribeiro (2002), is that exploring the two trajectories, in both directions, between two solutions results in small gains and demands twice the time. Thus, it is usually more appropriate to explore only one path, the one from the best solution to the other, as better solutions are more often found near the neighborhood of the best solutions (Resende & Ribeiro (2002)).

Selective Local Search (SLS): Another approach that can be used within the standard GRASP procedure is to apply local search only in those solutions sufficiently different from the elite solutions, or that have a good quality, *i.e.*, better than the worst of the elite solutions. In other words, we try to apply local search only where there are higher chances of obtaining better solutions, improving the efficiency of the algorithm.

2.3 The New GRASP Solver Implementation: GRASPW

The GRASPW implementation was constructed using the C/C++ programming language. Our model uses two types of integer variables. One represents the beginning of execution of each activity in the corresponding well. These values range between a minimum and a maximum start time, depending on the current partial solution being constructed. The second type of variables represents which resource will execute each activity in a well. Their domains are characterized by a set of the possible resources, of whose one must be chosen to execute the corresponding activity.

All the constraints described in Section 1 were enforced. Three constraints, namely, C2, C3 e C4, were set while reading the problem data, before the search begins. Note that, in these cases, all values needed to set the constraints are already defined. The other constraints were dealt with during the search for solutions, the variables involved being assigned single values.

Constraint C10, that deals with resource displacement and which was responsible for a loss of performance of the ORCA search algorithm, was treated here in a simple way. As we do not use a mechanism of constraint propagation, we displace the current activity by at least the amount of the set-up time, if the previous activity scheduled in the same resource is from a different well.

Of course, we could also have imposed constraint C10 during the search in ORCA instead of writing such constraints directly in the model. However this was not done for two reasons:

- First, this goes against the fundamental design principle in constraint programming, which dissociates the representation of the problem from strategies for its resolution (ILOG (1999*c*), ILOG (1999*a*)). This approach is known as *declarative programming*. In other words, this separation of the *model* from the *search* has a number of practical implications: it shortens development time, decreases maintenance problems and heightens adaptability of the application. Furthermore, such separation makes it easier to experiment with different strategies and different algorithms without redesigning the model, and even allows for the addition of further constraints to the problem without having to rewrite the search method.

- Besides generating solutions to the WDRDP, ORCA has also the functionality of validating the model. If the data of the problem generates inconsistencies, ORCA will inform the user that there is no feasible solution to the problem. In order to validate the import data, all the constraints must be imposed in the model, including the C10 constraints. Our GRASP solver does not have this functionality.

2.4 The Construction Phase

The following two adaptations were made to the procedure illustrated in Figure 2:

- The search procedure was interrupted by a time limit instead of by the number of iterations; and

- During a complete run of the GRASPW meta-heuristic, the value of $ListSize$ can be monotonically incremented by a fixed amount when a predefined interval of time is reached with no improvement on the best solution. This allows the algorithm to explore larger regions of the search space. Alternatively, during a run of GRASPW, the value of $ListSize$ can be monotonically decremented between iterations, thus focusing into a greedier heuristic. With this scheme we obtain a *dynamic sized RCL*, in opposition to the original *static sized RCL*. Note that, as GRASP iterations are independent, one could imagine that there is no difference between increasing and

decreasing the RCL size. However, as we do not know in advance the amount of time the algorithm will execute in each run, or when the RCL size will be altered, we can not anticipate the result of a GRASPW run, when increasing or decreasing the RCL size.

As in the ORCA implementation, we seek solutions with the highest oil yield. To this end, the construction phase illustrated in Figure 3 was modified thus:

1. The first time the construction phase is initiated, we use *ListSize* equals to one, when the algorithm behaves like a greedy heuristic. With few constraints obstructing the greedy heuristic, it tends to generate good or even very good solutions. For example, in six out of seventeen real instances, the best solution was found in the first pass of the construction phase.

2. The candidates are defined by the production wells that are available (meaning that there are no wells yet not scheduled which must precede them), or the injection wells that have activities of production wells succeeding them. The activities of injection wells that do not have activities of production wells succeeding them are left to be scheduled after all others. Note that injection wells are not productive and therefore must not be scheduled before production wells, unless there are technological constraints forcing such a schedule. We schedule the wells and not single activities because otherwise the solution would have a huge number of displacements of resources between wells, which would affect the oil production negatively.

3. The evaluation of incremental costs (line 3 of Figure 3) assesses how much oil a well can offer until the end of the time horizon. The RCL is built with those wells that offer the highest yields of oil. Actually, not only the oil offer is considered, but also the oil offer of the constrained successors of a well.

4. In the construction phase, the next element to be introduced in the solution is chosen uniformly from the candidates in the RCL (line 5 of Figure 3). However, any probability distribution can be used to bias the selection (see Section 2.2). We tried some bias functions in the selection of the candidates, as will be discussed later.

5. To schedule the candidate well (line 7 of Figure 3), the routine is as follows.

   As long as there are activities not yet scheduled in the well:

   (a) choose any activity available in the well, *i.e.*, one not yet scheduled and such that there is no other activity not yet scheduled in the wells that must precede it;

(b) choose a resource for this activity that can execute it, and that can complete the activity the earliest;

(c) set the start time of the activity at the earliest possible time, *i.e.*, the maximum between the earliest time the resource is available to execute the activity (considering the constraints, including the set-up time) and the minimum start time of the activity; and

(d) all activities that are constrained to succeed the chosen one must have their minimum start times updated so that all constraints are satisfied.

The scheduling of a well is done so as to satisfy all constraints, including the seven ones not yet enforced while reading the problem data. In case of violations, and this can be tested after each activity is scheduled, the construction of this solution is aborted and a new one is started. We could, instead, backtrack a few steps, but this would slow down this phase, especially if the first steps were not appropriate.

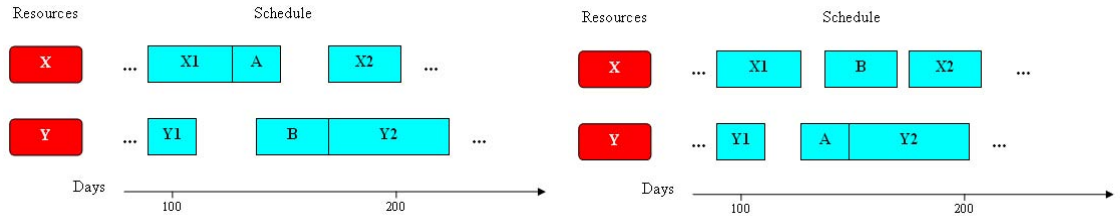
6. After a well is scheduled, any activities that must succeed it have their minimum start times updated to satisfy all constraints. If that is not possible, the construction of this solution is also aborted.

2.5 The Local Search Phase

As we have no guaranties that the solution found in the construction phase is locally optimum, a local search is used to improve the solution. For the search phase, an appropriate neighborhood was defined, so as to permit explorations that quickly lead to better solutions. The 2-exchange local search algorithm based on the disjunctive graph model of Roy & Sussmann (1964) was used. The same neighborhood was used in Binato et al. (2002) for a Job Shop Scheduling problem. In order to apply the 2-exchange local search to the WDRDP, we swap two elements in the scheduling. For example, consider the schedule presented in Figure 4.a. The elements (light gray squares) are executed by the resource (dark gray squares) which is depicted in the same horizontal line. The size of the squares represents the execution time of the corresponding elements. We swap elements A and B, which results in the schedule shown in Figure 4.b. Since the execution time of elements A and B can be different, all activities after them may have their start times updated.

We need to decide, of course, what an element stands for. Some options are:

1. *An activity.* Very small granularity, giving rise to huge neighborhoods (do Nascimento (2002)) and, worse, moving an activity to another position would possibly force its predecessors and successors in the same well to be moved as well, in order to avoid the displacements that may result from constraints of type C10;



(a) Before (b) After

Figure 4: 2-exchange swap.

2. *A well.* With higher granularity. But since the sequence of activities in a well may be splitted in the present schedule due to constraints of type C1, moving all activities takes time to verify all constraints, and exchanging the whole well may not be possible even though exchanging only part of it could be;

3. *Part of a well.* That means a maximal set of activities of the same well scheduled consecutively in the same resource. With medium granularity, it avoids the displacements that may result from constraints of type C10.

In our implementation, we chose the last alternative, where the local search algorithm exchanges all pairs of parts of wells, no matter on what resource they have been scheduled. That neighborhood is of size $O(n^2)$, where n is the number of parts of wells. For practical instances, this is one order of magnitude smaller than the neighborhood that uses activities as the moving elements.

To fully specify the local search phase we need a rule that defines how the neighborhood is searched and which solution replaces the current one. This rule is called the *pivoting rule* (Yannakakis (1997)), and examples of it are the *first improvement rule* (FIR) and the *best improvement rule* (BIR). In the first case, the algorithm moves to a neighboring solution as soon as it finds a better solution; in the second case, all neighbors are checked and the best one is chosen. In either case, the worst case running time of each iteration is bounded by $O(n^2)$, where n is the number of elements in the neighborhood. In the next section we present a comparison between these two alternatives.

Besides the improvements proposed above to the standard GRASP procedure, all the advanced techniques presented in Section 2.2 were implemented and tested.

3 Computational Results

In this section, computational results for the GRASPW implementation are discussed. They are also compared with results obtained with the ORCA implementation over the same real instances. All tests were run on a platform equipped with a Sun SPARC Ultra 60 processor, running a Solaris 9 operating system at 450 MHz and with 1024 MB of RAM. Both the GRASPW and the ORCA implementations were allowed to run for 1800 seconds on each instance.

3.1 Typical Instances

Twenty two real instances provided by Petrobras were used in our tests. Table 1 summarizes the dataset. Columns with the same numerical data, like columns 8 and 9, refer to distinct instances that differ in the number of other constraints not shown in the table, like C9. The first part of that table displays the instances were no C7 constraints were found. In order to reduce the amount of time spent in testing, in some experiments we used only 7 of these instances, eliminating instances that differed only by a few constraints. The lower part of Table 1 shows the ten instances were C7 constraints were present. It is worth mentioning that, as GRASP makes use of randomization, each instance was tested five times, and the results being reported always reflect the average of the tests.

The horizon of production ranges from a thousand to three thousand days. Note that, after every well has been scheduled, the oil yield is the same between all solutions until the horizon of production. Thus, any gains in production, when comparing two solutions, happen before all wells are scheduled. If we considered the full horizon to compute yields, these gains, in percent, would be smaller. Therefore, for each instance we use as the horizon of production the end date of the last activity of the well scheduled the latest, among all solutions of the solvers.

3.2 Setting GRASPW Parameters

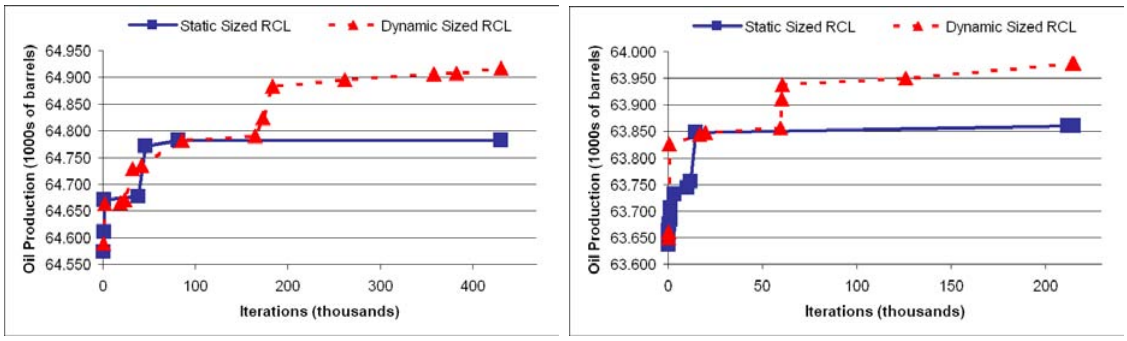
In Section 2 we presented the idea of a *dynamic sized RCL*. There are at least two ways we could exploit this idea: we may decrease monotonically the number of candidates using a greedier heuristic; or we may increase monotonically the number of candidates in order to drive away from a local optimum into new regions of the search tree. In the first case, the initial RCL size is set to $\max(13, w)$, w being the number of wells, and is decreased by one every 300 seconds without improvement. In the second case, we start with $\max(5, w)$ for the initial RCL size and increase it by one every 300 seconds without improvement. The first approach did not yield good results when applied to the WDRDP, generating the same or worse solutions than those found by GRASPW with a static sized RCL. However,

Instance	1	2	3	4	5	6	7	8	9	10	11	12
# wells	29	22	29	29	17	22	22	29	29	22	29	22
# activities	98	107	98	98	111	107	128	98	98	107	98	107
# boats	1	1	2	1	1	2	1	1	1	2	1	2
# derricks	3	2	3	3	2	2	3	3	3	2	3	2
# C7	0	0	0	0	0	0	0	0	0	0	0	0

Instance	13	14	15	16	17	18	19	20	21	22
# wells	22	22	22	22	22	22	22	22	29	65
# activities	107	107	107	107	107	107	107	107	98	338
# boats	2	2	2	1	2	2	2	2	1	1
# derricks	2	2	2	2	2	2	2	2	3	2
# C7	1	1	1	1	1	1	1	1	2	2

Table 1: Tested Instances.

the second approach proved promising. Figure 5.a shows the algorithm with dynamic sized RCL generating better solutions after 150 thousand iterations, when the RCL is increased. The same happens in Figure 5.b after 50 thousand iterations, when another real instance is tested. Amongst twelve scenarios tested, four had better solutions with the dynamic sized RCL, summing up an increase of around 261 thousand barrels of oil. In the other eight scenarios, the same solutions were found.

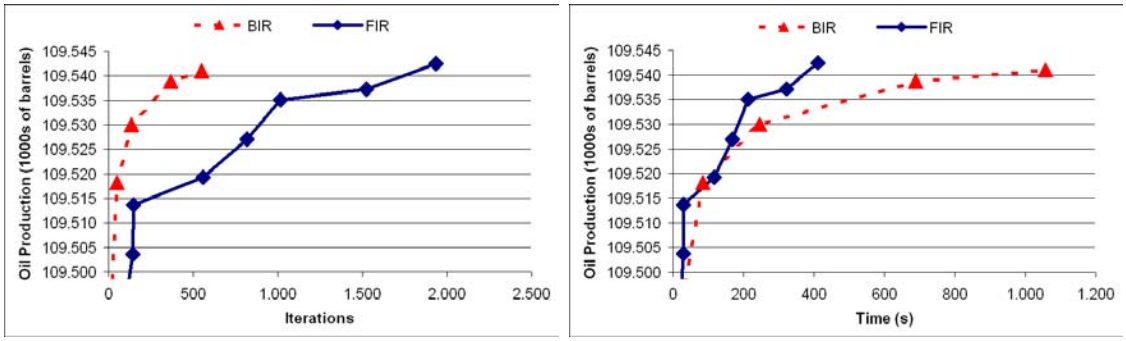


(a) Example 1 (b) Example 2

Figure 5: Static Sized RCL Solver x Dynamic Sized RCL Solver.

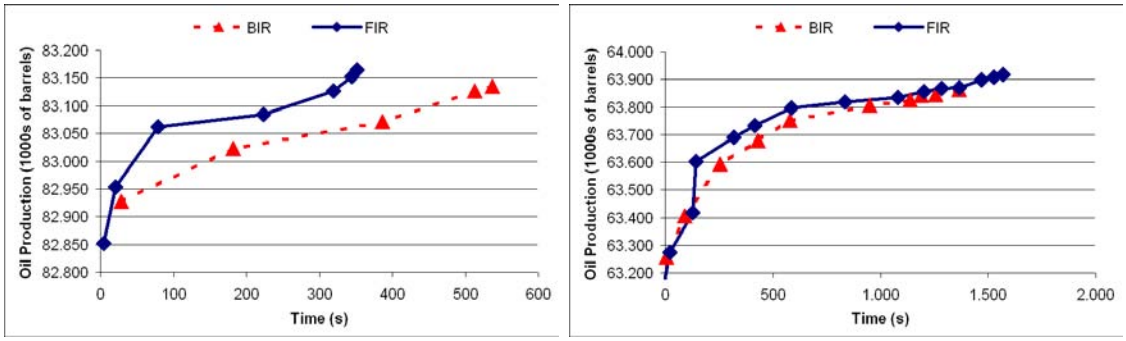
We also considered two options for searching the neighborhood and selecting a new neighbor: the *first improvement rule* (FIR) and the *best improvement rule* (BIR) strategy. Tests were executed and FIR has proved to be better. The BIR heuristics found solutions whose production was equal to a predefined target value with the least number of iterations

(see Figure 6.a). On average, to find a solution with a predefined production, the BIR strategy used about 60% of the number of iterations of the FIR strategy. On the other hand, the FIR strategy was faster in most instances (see Figure 6.b). On the average, to find a solution with a predefined production, the FIR strategy used 66% of the time used by the BIR strategy. That is because, on the average, a FIR iteration was almost seven times faster than a BIR iteration. Note that, to users, running time is deemed important. Note also that the FIR solver built solutions with a slightly higher oil production, as can be seen in Figures 6 and 7. Summing up all instances, the gain was almost 80 thousand barrels of oil. We concluded that the FIR strategy was the one that better suited this problem.



(a) (b)

Figure 6: BIR x FIR: Example 1

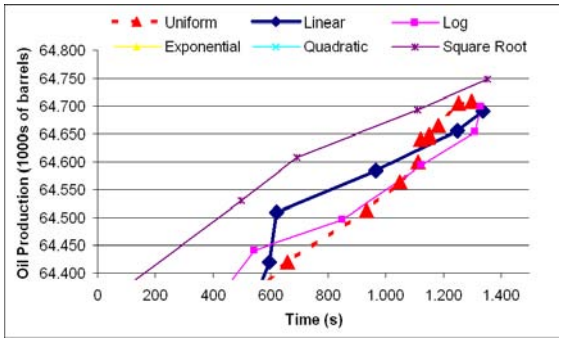


(a) Example 2 (b) Example 3

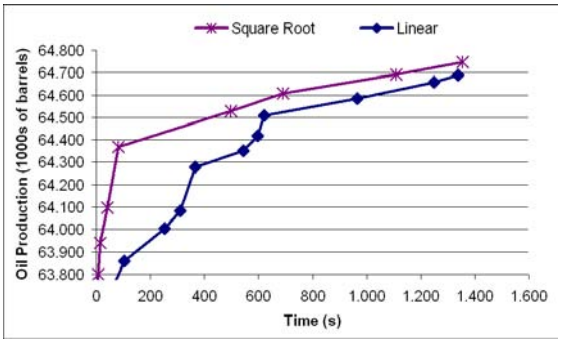
Figure 7: BIR x FIR.

Another technique tested was to bias the selection towards some particular candidates,

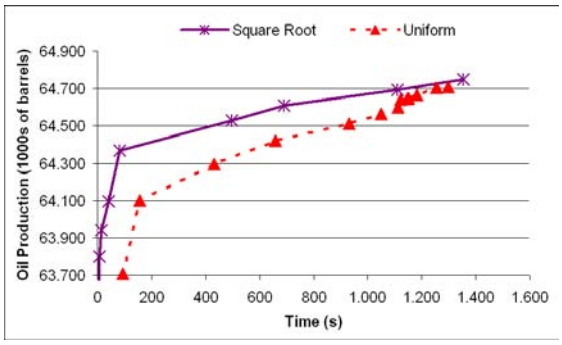
those with the highest oil yield. Five probability distributions, besides the uniform distribution, were considered, as presented in Section 2.2. Comparative experiments showed that the exponential and quadratic bias functions generated much worse solutions, with production of more than 2 millions barrels of oil smaller than the other approaches, summing up over all instances. Among the other bias functions, the function based on square root proved to be the best one for this problem, as can be seen in Figures 8.a and 9.a. For a better view over this two examples, we zoom comparing square root to others bias functions on Figures 8.(b,c,d) and 9.(b,c,d).



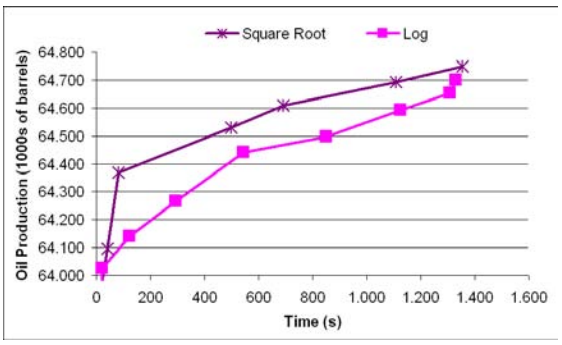
(a) Example 1



(b) Example 1: Square root X Linear



(c) Example 1: Square root X Uniform

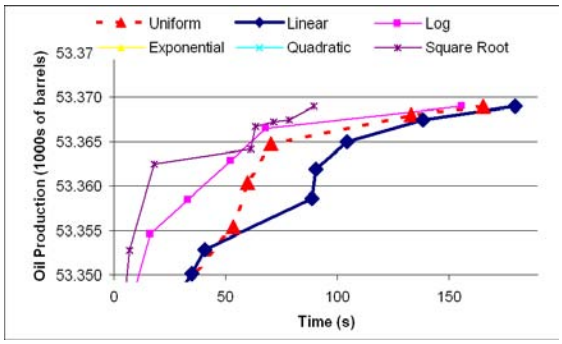


(d) Example 1: Square root X Log

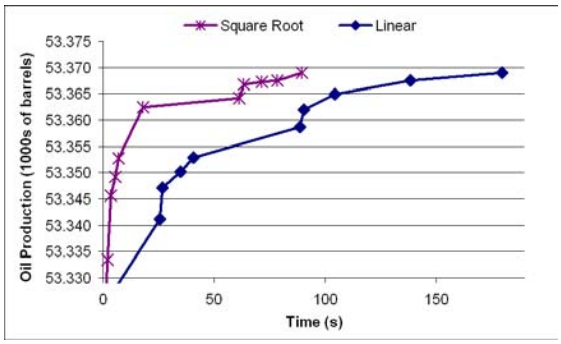
Figure 8: Bias Functions, I.

3.3 The GRASPW Implementations

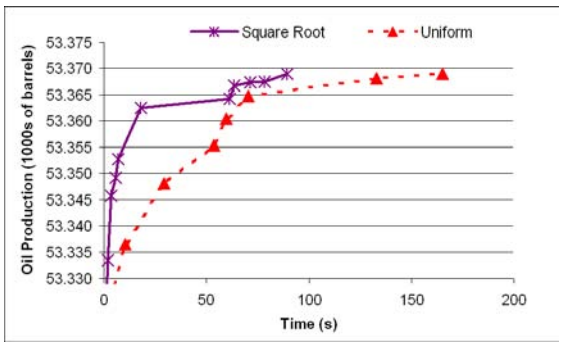
In Section 2.2, we presented some advanced techniques that can be included in the basic GRASP procedure. Many combinations of these were tested, but not all possible combinations. With 6 distinct techniques (five advanced plus the pivoting rule), all possible



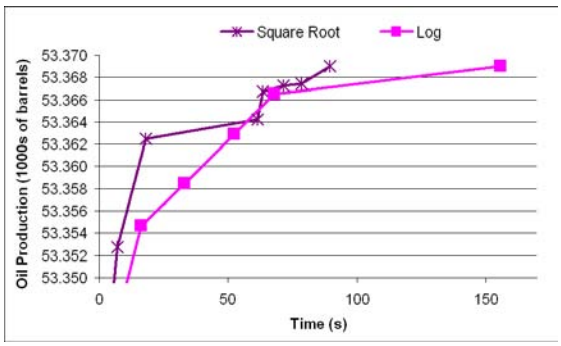
(a) Example 2



(b) Example 2: Square root X Linear



(c) Example 2: Square root X Uniform



(d) Example 2: Square root X Log

Figure 9: Bias Functions, II

combinations would number 64 distinct solvers. It was simply too time consuming to execute broad tests with each and every one of these solvers. We choose the best combinations based on faster tests, selecting a pool of 27 solvers. To simplify the visualization and data analysis, we show only 14 solvers, besides the ORCA solver. See Table 2.

As there are many solvers and in many tests there were ties between them, we elaborated five criteria to analyze which would be the best solver. We list these criteria below:

1. Sum of the oil production of the best solutions for all instances.

2. Sum of the oil production of the average solutions for all instances.

3. Average of the rank obtained sorting decreasingly the solver solutions of each instance using the *Ranking1 Rule* (R1), explained below. The sorting was according to the average oil production and average time of execution to find the best solution. We ranked all solvers.

Techniques	C.P.	GRASP	POP	Bias	Int	P-R	SLS	BIR
ORCA	•							
G1		•						
G2		•		•				
G3		•		•	•			
G4		•				•		
G5		•					•	
G6		•						•
G7		•	•					
G8		•	•				•	
G9		•	•					•
G10		•	•	•				
G11		•	•	•			•	
G12		•	•	•			•	•
G13		•	•	•	•		•	
G14		•	•	•	•	•	•	

C.P.	Constraint Programming
GRASP	Greedy Randomized Adaptive Search Procedure
POP	Proximate Optimality Principle
Bias	Bias Function (Square Root)
Int	Intensification
P-R	Path-Relinking
SLS	Selective Local Search
BIR	BIR

Table 2: Solvers

4. Average of the rank obtained sorting decreasingly the solver solutions of each instance using the *Ranking2 Rule* (R2), explained below. The sorting was according to the average oil production and average time of execution to find the best solution. We ranked all solvers.

5. Number of instances for which the solver obtained the best solution known to that instance.

Ranking1 Rule is a rank where, if s solvers are drawn in rank r, the next solvers would come in rank $r + 1$. *Ranking2 Rule*, in the same example, would put the next solvers in rank $r + s$.

Table 3 presents the values of the criteria achieved by the 15 solvers created for the WDRDP. Values in boldface are the best results among the solvers for each criteria. Note that G14 stands out being the best solver according to three criteria, followed closely by G7 which was the best in two criteria. However we can not say yet that G14 is the best solver, because if it behaves poorly in the other two criteria, another solver could prove

to be more appropriate for the WDRDP. According to these values, we elaborated Table 4, where we show the rank of the solvers in each criterion. Note that there are significant variations in the rank of the solvers when using different criteria.

Criterion	1	2	3	4	5
ORCA	331.043.827	331.043.827	9,588235	23,47059	2
G1	354.428.768	346.325.637	7,682353	10,47059	7
G2	354.421.217	346.255.908	8,035294	11,11765	6
G3	354.528.275	346.318.220	7,470588	10,23529	6
G4	354.379.849	346.255.508	8	11,41176	6
G5	354.428.768	346.342.803	6,823529	9,176471	7
G6	354.432.875	346.377.716	4,623529	8,235294	7
G7	354.838.320	346.807.529	**2,941176**	**4,352941**	8
G8	354.838.320	346.744.587	4,352941	5,764706	8
G9	354.838.320	346.801.864	3,470588	5,294118	8
G10	354.929.689	346.782.341	4,411765	6,588235	7
G11	354.929.689	346.762.940	4,294118	6,411765	**9**
G12	354.929.689	346.762.060	4,588235	6,941176	7
G13	**354.932.391**	346.765.955	4,058824	6	7
G14	354.888.020	**346.814.967**	**2,941176**	4,764706	**9**

Table 3: Values obtained by the Solvers in the Comparative Criteria

In order to make a decision about which would be the best solver, we again ranked the solvers according to the average rank they obtained in each criterion. We made this rank using both R1 and R2 type rules, as can be seen in Table 5. Using this table, we obtained the rank of the best solvers, presented in Table 6. We conclude that G14 is actually the best GRASP solver that we developed for the WDRDP.

This solver makes use of many advanced techniques for GRASP heuristics, like POP, bias function based on square root, intensification, *Path-Relinking* and SLS (see Section 2.2). Our second best solver for the problem was G7, which makes use only of the POP technique. As can be seen in Table 3, and by the average rank in Table 5, G7 is almost as good as G14. Hence, the other advanced techniques used in solver G14, besides POP, give only marginal gains to G14. POP, on the other hand, improves drastically the performance of a solver for this problem. This can be seen comparing G1 (basic GRASP) to G7 (basic GRASP with POP) in the same tables mentioned above, or comparing the group of solvers G1 until G6 (solvers without POP) to the group of G7 and the other solvers, that is, the group of solvers that implemented POP. When POP is applied, a great leap in quality can be seen. Note, for example, that G1 is just the eleventh best solver, while G7 is the second best.

Note also that G7 is better than G10, G11, G12 and G13, solvers which combine POP and other advanced techniques. This happened because these advanced techniques required

Criterion	1	2	3	4	5
	G13	G14	G7-G14	G7	G11-G14
	G10-G11-G12	G7	G9	G14	G7-G8-G9
	G14	G9	G13	G9	G1-G5-G6-G10-G12-G13
	G7-G8-G9	G10	G11	G8	G2-G3-G4
	G3	G13	G8	G13	ORCA
	G6	G11	G10	G11	
	G1-G5	G12	G12	G10	
	G2	G8	G6	G12	
	G4	G6	G5	G6	
	ORCA	G5	G3	G5	
		G1	G1	G3	
		G3	G4	G1	
		G2	G2	G2	
		G4	ORCA	G4	
		ORCA		ORCA	

Table 4: Rank of the Solvers

longer processing times from the solvers, and the gains were not sufficient to justify their use. Solver G7 could only be overcame when all the advanced techniques were combined in G14, and even so it was just slightly overcame.

With the above facts in mind, we concluded that POP is the best advanced technique to solve the WDRDP.

3.4 Comparative Results

As can be seen in Section 3.3, the G14 solver was considered the best GRASPW implementation. Note that the constraint programming solver, ORCA, was much worse than all the GRASPW solvers, even when compared to the standard GRASP procedure, G1.

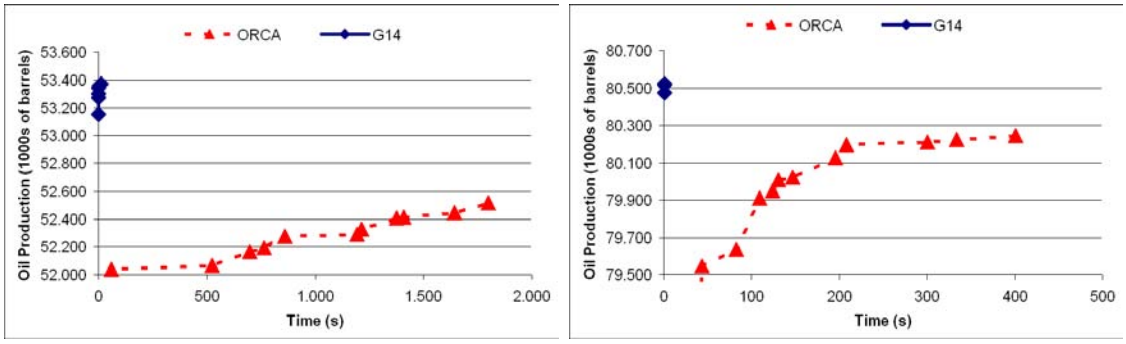
Compared to ORCA, the G14 solver generated gains of more than 16 millions barrels of oil, summing up the average solutions of all instances. This means an increase of 4,5% in the oil production. Note that, with the oil barrel price around US$ 52, the use of the G14 solver, instead of the ORCA solver, would yield gains of almost US$ 832 million. Besides, even generating better solutions, G14 takes, to find its best solution, only 95% of the time ORCA takes to find its best solution. And to find better solutions than the ORCA's best one, G14 takes only 45% of the time taken by ORCA. Among the 17 instances tested, G14 generates, within the first second of execution, better solutions than ORCA's best solution, in 14 instances. Two examples of that can be seen in Figure 10. In Figure 11, we show two instances where G14 does not generate, within the first second of execution, solutions better that the ORCA best solution.

Analysis of Criteria	R1	R2	Average R1-R2
ORCA	11,8	15	13,4
G1	8,8	10,4	9,6
G2	10,2	13	11,6
G3	8,4	11	9,7
G4	10,6	13,4	12
G5	7,8	9,4	8,6
G6	7	8,6	7,8
G7	2	2,6	2,3
G8	4,6	5,4	5
G9	2,8	3,6	3,2
G10	4,4	5,2	4,8
G11	3,8	4	3,9
G12	5,4	6,2	5,8
G13	3,4	4,2	3,8
G14	**1,6**	**2**	**1,8**

Table 5: Analysis of the Rank of the Solvers

However, there is an instance where ORCA was competitive compared to G14, as can be seen in Figure 12. Here, ORCA's best solution was found in 5 seconds and G14 only generated a better solution than that after 1134 seconds. Nevertheless, after 1522 seconds, G14 generated a solution with an oil production almost 63 thousand barrels superior than ORCA's best solution.

We conclude that our GRASPW implementation is much more efficient and generate better solutions than the ORCA solver.



(a) Example 1 (b) Example 2

Figure 10: ORCA × G14, I

Solver
G14
G7
G9
G13
G11
G10
G8
G12
G6
G5
G1
G3
G2
G4
ORCA

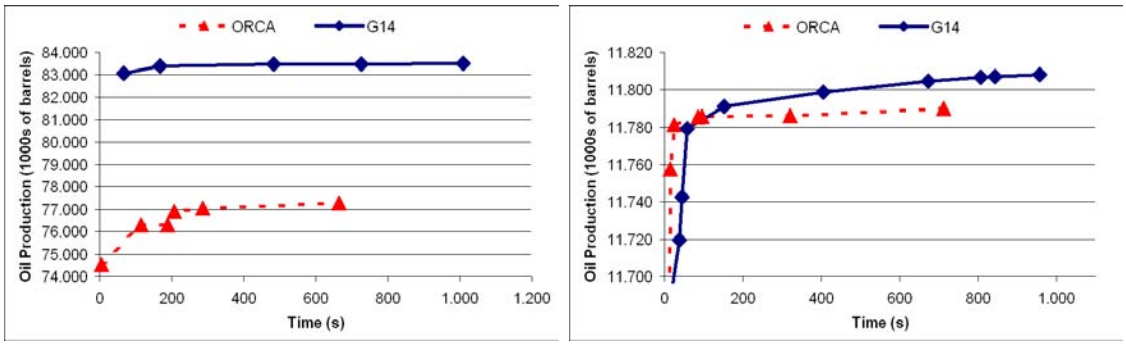
Table 6: Rank of the Best Solvers

4 Conclusions

Scheduling activities efficiently is of paramount importance to the industry, in general. Petrobras, a leading company in deep water oil exploration, presented us the WDRDP, a scheduling problem related to oil well development. Here we constrast two approaches to the WDRDP: the constraint programming tool ORCA and a GRASP implementation, dubbed GRASPW.

Computational experiments were carried out on several real instances. We conclude that GRASPW greatly outperforms ORCA. Not only it generates solutions with higher oil production, but often it outputs solutions with the same oil production as ORCA, but in much less time. We recall that ORCA already produced better results than the manual solutions.

It is worth mentioning that ORCA is built using the ILOG Constraint Programming suite, a set of expensive and sophisticated libraries with years of development. Using GRASPW, which was entirely programmed from the ground up, these costs could be averted. In opposition, the ILOG suite favors easiness of development, of maintenance and of understanding of the source code.



(a) Example 3 (b) Example 4

Figure 11: ORCA × G14, II

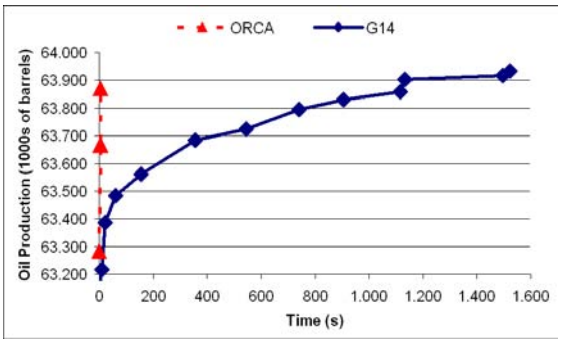


Figure 12: ORCA×G14: Example 5

5 References

Aiex, R. M., Resende, M. G. C. and Toraldo, G., 2000, GRASP with path-relinking for the three-index assignment problem, Technical report, AT&T Labs.

Bard, J. F. and Feo, T. A., 1989, 'Operations sequencing in discrete parts manufacturing', *Management Science* **35**, 249–255.

Bard, J., Feo, T. and Holland, S., 1996, 'A GRASP for scheduling printed wiring board assembly', *I.I.E. Transactions* **28**, 155–165.

Binato, S., Hery, W. J., Loewenstern, D. and Resende, M. G. C., 2001, A GRASP for job shop scheduling, *in* P. Hansen and C. C. Ribeiro, eds, 'Essays and surveys on metaheuristics', Kluwer Academic Publishers, pp. 59–79.

Bresina, J. L., 1996, Heuristic-biased stochastic sampling, *in* 'Proceedings of the Thirteenth National Conference on Artificial Intelligence', pp. 271–278.

Canuto, S. A., Resende, M. G. C. and Ribeiro, C. C., 2001, 'Local search with perturbations for the prize-collecting steiner tree problem in graphs', *Networks* **38**, 50–58.

do Nascimento, J. M., 2002, Hybrid computational tools for the optimization of the production of petroleum in deep waters, Master's thesis, Institute of Computing, University of Campinas.

Feo, T. A. and Bard, J. F., 1989, 'Flight scheduling and maintenance base planning', *Management Science* **35**, 1415–1432.

Feo, T. A., Bard, J. and Holland, S., 1995, 'Facility-wide planning and scheduling of printed wiring board assembly', *Operations Research* **43**, 219–230.

Feo, T. A. and Resende, M. G. C., 1995, 'Greedy randomized adaptative search procedures', *Journal of Global Optimization* **6**, 109–133.

Festa, P. and Resende, M. G. C., 2001, GRASP: An annotated bibliography, Technical report, AT&T Labs.

Fleurent, C. and Glover, F., 1999, 'Improved constructive multistart strategies for the

quadratic assignment problem using adaptive memory', *INFORMS Journal on Computing* **11**, 198–204.

Glover, F., 1996, Tabu search and adaptive memory programming — advances, applications and challenges, *in* R. S. Barr, R. V. Helgason and J. L. Kennington, eds, 'Interfaces in Computer Science and Operations Research', Kluwer, pp. 1–75.

Glover, F., 2000, Multi-start and strategic oscillation methods — principles to exploit adaptive memory, *in* M. Laguna and J. L. Gonzáles-Velarde, eds, 'Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research', Kluwer, pp. 1–24.

Glover, F., Laguma, M. and Martí, R., 2000, Fundamentals of scatter search and path relinking, *in* M. Laguna and J. L. Gonzáles-Velarde, eds, 'Control and Cybernetics', Vol. 39, Kluwer, pp. 653–684.

Glover, F. and Laguna, M., 1997, *Tabu Search*, Kluwer Academic Publishers, Norwell, Massachusetts.

ILOG, 1999*a*, *ILOG Solver 4.4 Getting Started*, ILOG.
ILOG, 1999, *ILOG Solver 4.4 Reference Manual*, ILOG.

ILOG , 1999*c*, *ILOG Solver 4.4 User's Manual*, ILOG.

Marriott, K. and Stuckey, P. J., 1998, *Programming with Constraints: An introduction*, MIT Press, Cambridge, Massachusetts.

Pereira, R. A., Moura, A. V. and de Souza, C. C., 2005, Comparative experiments with GRASP and constraint programming for the oil well drilling problem, *in* S. E. Nikoletseas, ed., 'Experimental and Efficient Algorithms - WEA 2005, LNCS 3503', Springer,
pp. 328–340.

Resende, M. G. C. and Ribeiro, C. C., 2001, A GRASP with path-relinking for private virtual circuit routing, Technical report, AT&T Labs.

Resende, M. G. C. and Ribeiro, C. C., 2002, 'Greedy randomized adaptive search procedure', *AT&T Labs Research Technical Report TD53RSJY*.

Roy, B. and Sussmann, B., 1964, Les problèmes d'ordonnancement avec contraintes disjonctives, *in* 'Note DS No 9 bis', SEMA, Paris.

Yannakakis, M., 1997, Computational complexity, *in* E. H. L. Aarts and J. K. Lenstra, eds, 'Local Search in Combinatorial Optimization', John Wiley & Sons, Chichester, pp. 19–55.

Epílogo

Foi explicado no artigo que um dos motivos para a não imposição das restrições de deslocamento de recursos diretamente na busca é o princípio fundamental da programação por restrições, que defende a dissociação entre modelo e busca. Assim, o ORCA consome um tempo de execução considerável na propagação das restrições de deslocamento, que são de ordem $O(n^2)$, e portanto perde eficiência. Ainda assim, o ORCA preferiu seguir o princípio da programação por restrições. Isso porque, por exemplo, no ORCA atual se fosse decidido que as restrições de deslocamento não são mais necessárias, apenas o modelo seria alterado. Já se essas restrições estivessem definidas na busca, toda ela teria de ser remodelada. Ou seja, nesse último caso, uma alteração de modelagem implicaria numa alteração no algoritmo de busca. Isso dificulta a manutenção do código, pois para alterar a modelagem um indivíduo deve conhecer não só o modelo, como também a busca.

Assim como no primeiro artigo, optamos por não fazer uma análise do desvio padrão das produções das melhores soluções geradas pelo GRASP e, portanto, de sua robustez. Isso porque em todos os testes realizados com um mesmo resolvedor em uma mesma instância, a diferença entre as produções de soluções diferentes nunca superou os 0.5%. Dessa forma, um mesmo resolvedor GRASP mostra uma variação muito pequena entre a qualidade de suas soluções para uma mesma instância.

No artigo, foi explicado duas regras, dentre outras, para a escolha dos melhores resolvedores GRASP, a saber as regras *Ranking1 Rule (R1)* e *Ranking2 Rule (R2)*. Para esclarecer melhor essas regras, suponha um exemplo em que os resolvedores X e Y geram as melhores soluções, ambos com uma média idêntica na produção de suas soluções. Assim, ambos estão na posição 1, tanto em R1 como em R2, já que são os melhores. O resolvedor Z, por sua vez, vem logo após os resolvedores anteriores na qualidade das soluções geradas. Quanto a R1, Z está na posição 2, pois gera as soluções de segunda melhor qualidade. Já quanto a R2, Z está na posição 3, pois é o terceiro melhor resolvedor. Essas duas regras foram utilizadas por representarem visualizações diferentes e importantes para a escolha do melhor resolvedor.

Outro fato a ser mencionado aqui é que, diferentemente de outras publicações, tais como Aiex et al. (2001), Resende & Ribeiro (2001) e Aiex et al. (2003), o *path-relinking* gerou poucos ganhos em nosso GRASP. Como explicado no artigo, a estratégia do *path-relinking* consiste em explorar caminhos que levam de uma solução à outra. Esse "caminho", por sua vez, pode ser definido de várias formas. Em nosso trabalho, optamos por levar um poço à uma posição na ordem de execução de um recurso igual àquela em que ocorre na solução sendo comparada. Ou seja, ao caminharmos da solução $E2$ para a $E1$, tomamos o poço $P1$ que foi o primeiro executado pelo barco $B1$ em $E1$ e o fazemos o primeiro executado pelo mesmo $B1$ em $E2$. Isso é feito via *2-exchange* (Roy & Sussmann (1964)), ou seja, colocamos

$P1$ em primeiro e colocamos o que antes era primeiro onde estava $P1$ no escalonamento. O processo é repetido para todos os recursos e todas as posições de execuções nos recursos. Dessa forma $E2$ convergirá para uma solução igual ou semelhante à $E1$.

Entretanto, há um problema intrínseco nessa abordagem. Devido às miríades de restrições do problema, poços podem ter sido particionados no escalonamento tanto na solução $E2$ como na $E1$, mas não necessariamente os mesmos poços e da mesma forma. Assim, para aproximarmos $E2$ de $E1$, precisaríamos unir as atividades dos poços particionados em $E2$ que não se particionaram em $E1$, e separar as atividades dos poços unidos de $E2$ que estão particionadas em $E1$. Além disso, um poço pode estar particionado em $E1$ e em $E2$, mas de formas diferentes. Então, aproximar duas soluções quando há particionamentos torna-se um problema razoável de codificação e, muito pior, demanda um tempo computacional significativo, prejudicando a performance do algoritmo. Ou seja, se implementada essa aproximação, o *path-relinking* iria tornar as iterações do GRASP bem mais lentas. Dessa forma, optamos por aproximar as soluções apenas com os poços que não estão particionados em ambas as soluções, limitando porém a aproximação entre elas. Logo, o *path-relinking* implementado é incompleto.

Outra questão é que nas publicações já citadas opta-se por sempre escolher o melhor movimento na vizinhança de uma solução em direção à solução guia. Nesta tese, não escolhemos necessariamente o melhor movimento, isso porque decidir qual é o melhor movimento poderia demandar um tempo computacional excessivo. Entretanto, escolhemos movimentos que tendem a ter maior impacto na produção final, e que portanto tendem a ser os melhores, já que nossos movimentos se concentram nos primeros poços escalonados diferentemente nas soluções. Uma opção a ser testada futuramente é sempre escolher o melhor movimento, mas aplicar o *path-relinking* apenas às soluções que superem um certo critério de qualidade, assim evitando um consumo excessivo de tempo computacional. Para acelerar ainda mais o algoritmo, pode-se também truncar o caminho entre as soluções.

Outro idéia seria selecionar a solução elite a ser combinada com a solução GRASP aleatoriamente, mas com probabilidades proporcionais ao número de elementos diferentes entre essas soluções. Isso tende a gerar caminhos mais longos entre as duas soluções, levando a mais chances de encontrar soluções melhores (Resende & Ribeiro (2003)).

Ainda outra alternativa é aplicar a busca local em soluções encontradas no caminho do *path-relinking*. A aplicação da busca local deve se dar de tempos em tempos, de acordo com algum parâmetro de freqüência ou critério de qualidade, para evitar perda de eficiência do algoritmo.

Assim, melhorias no esquema de *path-relinking* podem ser tentadas em trabalhos futuros.

Parte IV

Considerações Finais

Capítulo 8

Conclusões

O GRASP é uma metaheurística bem estudada e aplicada em diversos problemas NP-difícieis. Entretanto, a Petrobras enfrenta um particular tipo de problema de escalonamento para o qual não existe ferramenta comercial pronta, nem algoritmos eficientes sob estudo. O GRASP nunca foi testado nesse problema. Assim, pioneiramente, aplicamos o GRASP ao problema enfrentado pela Petrobras para lidar com escalonamento de recursos em alto mar. A ferramenta desenvolvida não só automatiza o processo, antes manual, como otimiza esse processo e supera uma ferramenta desenvolvida pela empresa para o mesmo fim. O problema é, provavelmente, também enfrentado por outras empresas petrolíferas ao redor do globo, tornando o inovador estudo do problema ainda mais abrangente.

Nosso estudo não só aumentou os conhecimentos sobre a metaheurística GRASP, testando-a sob um novo prisma, como criou melhores formas de tratar o problema de escalonamento de recursos em alto mar. Para a Petrobras, os ganhos são de cerca de 1 milhão de barris de petróleo por escalonamento, quando comparada a uma ferramenta já superior ao processo manual. Com o barril de petróleo a US$58[1], os ganhos são de cerca de US$58 milhões, por escalonamento. A Petrobras faz em média 2 ou 3 escalonamentos por ano.

Outro ponto importante é que, apesar de o GRASP fazer uso de randomização, em todos os testes realizados com um mesmo resolvedor em uma mesma instância, a diferença entre as produções de soluções diferentes nunca superou os 0.5%. Dessa forma, os resolvedores GRASP se mostraram robustos. Isso é relevante para os engenheiros da Petrobras, pois o GRASP dificilmente seria aceito se suas execuções variassem entre soluções boas, médias, ruins e situações de não solução.

No meio acadêmico, o trabalho foi bem recebido, e um artigo inicial descrevendo a abordagem adotada foi aprovado no *4th International Workshop on Efficient and Experimental Algorithms* (WEA 2005 - http://ru1.cti.gr/wea05/). Convidado, Romulo foi apresentar o trabalho no congresso, realizado na Ilha de Santorini, Grécia. O artigo também foi publi-

[1]Cotação de 16/11/2005.

cado no volume 3503[2] da série *Lecture Notes in Computer Science*, da editora Springer. Esse artigo inicial, por ter participado do WEA2005, poderá vir a ser publicado numa edição especial do *ACM Journal on Experimental Algorithmics* (www.jea.acm.org) dedicado ao evento. Um segundo artigo foi premiado no Prêmio Petrobras de Tecnologia como terceiro colocado na categoria Produção[3] e foi submetido ao *Journal of Heuristics*[4], uma importante publicação da área de otimização combinatória.

Devido ao sucesso da ferramenta desenvolvida, ela já foi implantada na Petrobras e já está em uso em processos na Bacia de Campos e Bacia do Espírito Santo.

Ao mesmo tempo em que procurou estudar o problema de escalonamento no desenvolvimento de poços de petróleo em alto mar enfrentado pela Petrobras, essa dissertação teve como objetivo adicional comparar os resultados de uma implementação específica de programação por restrições e uma implementação específica da metaheurística GRASP. Experimentos computacionais foram conduzidos em instâncias reais do problema. Concluímos que o GRASP desenvolvido gera soluções com maior produção petrolífera que aquelas geradas pelo resolvedor ORCA, que usa programação por restrições. Tal se deu tanto no EDP, quanto no EDPDR, quando se adotou técnicas mais avançadas do GRASP. Os resultados também provam que o GRASP supera o resolvedor ORCA quando gera soluções de mesma produção em menor tempo de execução.

É válido mencionar que o resolvedor ORCA foi construído com a suíte de Programação por Restrições da ILOG, que consiste de sofisticadas bibliotecas com mais de uma década de desenvolvimento[5]. Além disso, essa suíte é de preço elevado, tanto no valor de aquisição, quanto no valor de manutenção da licença. Dessa forma, usando o GRASP desenvolvido, a Petrobras pode evitar esses custos. Note, entretanto, que a suíte da ILOG favorece a facilidade de desenvolvimento, de manutenção e de entendimento do código fonte. Note também que os resultados gerados pelo ORCA superam resultados manuais, *i.e.*, soluções desenvolvidas por engenheiros.

De uma maneira geral, acredita-se que os objetivos dessa dissertação foram alcançados satisfatoriamente. Instâncias reais do problema original foram resolvidas em tempos de computação aceitáveis. Além disso, foi possível conhecer mais a fundo todas as técnicas utilizadas e as dificuldades intrínsecas do problema tratado. Desse modo, contribuiu-se positivamente para o sucesso de futuros empreendimentos nessa área.

[2]http://www.springeronline.com/sgw/cda/frontpage/0,11855,4-149-22-48266374-0,00.html
[3]Vide: http://www2.petrobras.com.br/tecnologia2/port/pptecnologia.asp
[4]http://www.kluweronline.com/issn/1381-1231
[5]Vide http://www.ilog.com

Capítulo 9

Trabalhos Futuros

O trabalho desenvolvido nesta dissertação pode ser estendido de diversas maneiras:

- Diferentemente de outras publicações, tais como Aiex et al. (2001), Resende & Ribeiro (2001) e Aiex et al. (2003), o *path-relinking* gerou poucos ganhos em nosso GRASP. Poder-se-ia tentar adaptar essa técnica de uma melhor forma para o EDP, usando por exemplo as abordagens citadas no epílogo do Capítulo 7.

- A Petrobras vem incorporando *clusters* de processadores ao seu parque computacional. Dessa forma, seria interessante adaptar a ferramenta GRASP aqui desenvolvida para atuar paralelamente, utilizando o paralelismo dos *clusters*.

- Testar técnicas de "perturbações nos custos" (*cost perturbations*) no GRASP, como em Canuto et al. (2001), Ribeiro et al. (2002) e Resende & Ribeiro (2002).

- Testar técnicas híbridas, como GRASP com busca tabu (Ribeiro et al. (2002), Colomé & Serra (1998)) ou GRASP com algoritmos genéticos (Lourenço et al. (1998)).

- Testar outras técnicas, como busca tabu e algoritmos genéticos para o EDP, e compará-las ao GRASP.

Bibliografia

Aiex, R. M., Binato, S. & Resende, M. G. C. (2003), 'Parallel GRASP with path-relinking for job shop scheduling', *Parallel Computing* **29**(4), 393–430.

Aiex, R. M., Resende, M. G. C. & Toraldo, G. (2001), GRASP with path-relinking for the three-index assignment problem, Technical report, AT&T Labs.

Binato, S. & Oliveira, G. (2002), 'A reactive GRASP for transmission network expansion planning', *Essays And Surveys In Metaheuristics - Kluwer Academic Publishers* .

Canuto, S. A., Resende, M. G. C. & Ribeiro, C. C. (2001), 'Local search with perturbations for the prize-collecting steiner tree problem in graphs', *Networks* **38**, 50–58.

Clocksin, W. (1987), 'Principles of the delphi parallel inference machine', *Computer Journal* **30**(5), 386–392.

Colomé, R. & Serra, D. (1998), 'Consumer choice in competitive location models: Formulations and heuristics', *Economics Working Papers* (290).

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2001), *Introduction to Algorithms*, 2nd edn, MIT Press.

do Nascimento, J. M. (2002), Hybrid computational tools for the optimization of the production of petroleum in deep waters, Master's thesis, Institute of Computing, University of Campinas.

Faria, H., Binato, S., Resende, M. & Falcão, D. M. (2005), 'Power transmission network design by a greedy randomized adaptive path relinking approach', *IEEE Transactions on Power Systems* **20**(1), 43–49.

Festa, P. & Resende, M. (2002), 'GRASP: An annotated bibliography', *Essays And Surveys In Metaheuristics - Kluwer Academic Publishers* .

Fortuna, V. J., Moura, A. V. & de Souza, C. C. (2002), Relatório 2 de iniciação científica, Relatório FAPESP, Instituto de Computação, Unicamp.

Garey, M. R. & Johnson, D. S. (1979), *Computers and intractability: A guide to the theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, California.

Harvey, W. D. & Ginsberg, M. L. (1995), 'Limited discrepancy search', *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* **1**, 607–613.

Hasle, G., Haut, R., Johansen, B. & Ølberg, T. (1997), Well activity scheduling – an application of constraint reasoning, *in* 'Proceedings of PACT'97 (The 1997 International Conference on Parallel Architectures and Compilation Techniques)', San Francisco, Callifornia.

ILOG (1999*a*), *ILOG Solver 4.4 Reference Manual*, ILOG.

ILOG (1999*b*), *ILOG Solver 4.4 User's Manual*, ILOG.

Jaffar, J. & Lassez, J. (1987), Constraint logic programming, *in* '14^{th} ACM Symposium on Principles of Programming Languages', Munique, Alemanha, pp. 111–119.

Le Pape, C. (1994), 'Implementation of resource constraints in ILOG SCHEDULE: a library for the development of constraint-based scheduling systems', *Intelligent Systems Engineering* **3**(2), 55–66.

Lever, J., Wallace, M. & Richards, B. (1995), 'Constraint logic programming for scheduling and planning', *BT Technology Journal* **13**(1), 73–80.

Lourenço, H. R., Paixão, J. & Portugal, R. (1998), Metaheuristics for the bus-driver scheduling problem, *in* 'Economics Working Papers', Vol. 304, Department of Economics and Business, Universitat Pompeu Fabra.

Marriott, K. & Stuckey, P. J. (1998), *Programming with Constraints: An introduction*, MIT Press, Cambridge, Massachusetts.

Meseguer, P. (1997), 'Interleaved depth-first search', *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* **2**, 1382–1387.

Moura, A. V. & de Souza, C. C. (2000), 'Reuniões com Engenheiros da Petrobras, estacionados na Bacia Petrolífera de Campos — notas não publicadas'.

Nilsson, N. J. (1971), *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill.

Pereira, R. A., Moura, A. V. & de Souza, C. C. (2002), Estudo e desenvolvimento de ferramentas para sequenciamento de atividades no desenvolvimento de poços de petróleo: Uso de programação por restrições, Relatório FAPESP, Instituto de Computação, Unicamp.

Pereira, R. A., Moura, A. V. & de Souza, C. C. (2005*a*), Comparative experiments with GRASP and constraint programming for the oil well drilling problem, *in* S. E. Nikoletseas, ed., 'Experimental and Efficient Algorithms - WEA 2005, LNCS 3503', Springer, pp. 328–340.

Pereira, R. A., Moura, A. V. & de Souza, C. C. (2005*b*), Constraint programming and grasp approaches to schedule oil well drillings, Relatório técnico IC-05-001, http://www.ic.unicamp.br/reltec-ftp/2005/titles.html, Instituto de Computação, Unicamp.

Pereira, R. A., Moura, A. V. & de Souza, C. C. (2005*c*), Grasp strategies for scheduling activities at oil wells with resource displacement, Relatório técnico IC-05-028, http://www.ic.unicamp.br/reltec-ftp/2005/titles.html, Instituto de Computação, Unicamp.

Resende, M. G. C. & Ribeiro, C. C. (2001), A GRASP with path-relinking for private virtual circuit routing, Technical report, AT&T Labs.

Resende, M. G. C. & Ribeiro, C. C. (2002), 'Greedy randomized adaptive search procedure', *AT&T Labs Research Technical Report TD53RSJY* .

Resende, M. G. C. & Ribeiro, C. C. (2003), GRASP and path-relinking: Recent advances and applications, Technical report td-5tu726, AT&T Labs Research.

Ribeiro, C. C., Uchoa, E. & Werneck, R. F. (2002), 'A hybrid GRASP with perturbations for the steiner problem in graphs', *INFORMS Journal on Computing* **14**, 228–246.

Ribeiro, M. (2002), 'A GRASP for job shop scheduling', *Essays And Surveys In Metaheuristics* pp. 59–79.

Roy, B. & Sussmann, B. (1964), Les problèmes d'ordonnancement avec contraintes disjonctives, *in* 'Note DS No 9 bis', SEMA, Paris.

Souza, M., Duhamel, C. & Ribeiro, C. (2004), 'A GRASP heuristic for the capacitated minimum spanning tree problem using a memory-based local search strategy', *Applied Optimization* pp. 627–657.

Srinivasan, A., Ramakrishnan, K. & Kumaran, K. (2000), 'Optimal design of signaling networks for internet telephony', *Lucent Technologies, Labs Innovations* .

Walsh, T. (1997), 'Depth-bounded discrepancy search', *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* **2**, 1388–1393.

Yunes, T. H., Moura, A. V. & de Souza, C. C. (2000*a*), A hybrid approach for solving large scale crew scheduling problems, *in* 'Lecture Notes in Computer Science, vol. 1753', Boston, MA, EUA, pp. 293–307. Anais do *Second International Workshop on Practical Aspects of Declarative Languages (PADL'00)*.

Yunes, T. H., Moura, A. V. & de Souza, C. C. (2000*b*), Solving very large crew scheduling problems to optimality, *in* '14^{th} ACM Symposium on Applied Computing (SAC'00)', Como, Itália, pp. 446–451.