METAHEURÍSTICAS PARA O PROBLEMA QUADRÁTICO DE ALOCAÇÃO EM LINGUAGEM DE PROGRAMAÇÃO GO

Daniel de Mazza Cerqueira Mendes

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Produção, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Produção.

Orientadora: Laura Silvia Bahiense da Silva Leite

Rio de Janeiro

Novembro de 2015

METAHEURÍSTICAS PARA O PROBLEMA QUADRÁTICO DE ALOCAÇÃO EM LINGUAGEM DE PROGRAMAÇÃO GO

Daniel de Mazza Cerqueira Mendes

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE PRODUÇÃO.

Examinada por:

Prof. Laura Silvia Bahiense da Silva Leite, D.Sc.

Prof. Ricardo Cordeiro de Farias, D.Sc.

Prof. Samuel Jurkiewicz, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
NOVEMBRO DE 2015

Mendes, Daniel de Mazza Cerqueira

Metaheurísticas para o Problema Quadrático de Alocação em Linguagem de Programação Go / Daniel de Mazza Cerqueira Mendes. – Rio de Janeiro: UFRJ / COPPE, 2015.

XIII, 101 p.: il.; 29, 7cm.

Orientadora: Laura Silvia Bahiense da Silva Leite

Dissertação (mestrado) – UFRJ / COPPE / Programa de Engenharia de Produção, 2015.

Referências Bibliográficas: p. 85 – 86.

1. Problema Quadrático de Alocação. 2. Otimização Combinatória. 3. Go. I. Leite, Laura Silvia Bahiense da Silva. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Produção. III. Título.

À Estrella D'Alva Benaion Bohadana que sempre me incentivou na vida, na jornada acadêmica e na curiosidade do mundo que pode vir a ser.
*(*in memoriam*).*

Aos meus pais, Carlos e Izabel, que me educaram a enveredar o mundo que está pelo caminho que é.

E a todos – mesmo os anônimos – cuja existência testemunha em favor de uma humanidade que vale a pena; que são a razão de ser da pesquisa científica: são o sentido e por vezes o próprio caminho de realização do que é bom dentre tudo que poderia ser.

Agradecimentos

Gostaria de agradecer primeiramente a Dra. Laura Bahiense pela obstinada paciência com que soube conduzir minha personalidade complexa por conselhos certeiros para um fim produtivo. Agradeço também ao Ph.D. Francisco Antonio Doria, Dr. Samuel Jurkiewicz e Dr. Paulo Oswaldo Boaventura Netto que tão gentilmente acolheram, incentivaram e contribuíram para o esforço de meu trabalho. Agradeço também aos amigos a Rennan Chagas e Manuel Doria com os quais quase cotidianamente debati sobre o que vinha pesquisando e desenvolvendo, de modo que a presença dos mesmos tenha influenciado positivamente o tom do presente trabalho. E, especialmente, quero agradecer ao meu amor, Marianne Ferruzzi Tine, que esteve ao meu lado me fortalecendo durante todo o mestrado entendendo a dedicação que o mesmo requereu. E agradeço ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo suporte financeiro que viabilizou esse trabalho.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

METAHEURÍSTICAS PARA O PROBLEMA QUADRÁTICO DE ALOCAÇÃO EM LINGUAGEM DE PROGRAMAÇÃO GO

Daniel de Mazza Cerqueira Mendes

Novembro/2015

Orientadora: Laura Silvia Bahiense da Silva Leite

Programa: Engenharia de Produção

Apresentamos uma implementação em linguagem de programação Go de heurísticas e metaheurísticas para o Problema Quadrático de Alocação (PQA), bem como a proposta da metaheurística Íris que formulamos a partir do estudo do sucesso e dos problemas de outras. Propomos alternativas para o comportamento de algumas heurísticas tradicionais utilizando-se de parcial parcial força-bruta com o algoritmo Steinhaus-Johnson-Trotter (*Plain Changes*) que também compõe, articulado com mapeamento a números de base fatorial (fatorádicos), a proposta de Íris.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

METAHEURISTICS FOR THE QUADRATIC ASSIGNMENT PROBLEM IN GO PROGRAMMING LANGUAGE

Daniel de Mazza Cerqueira Mendes

November/2015

Advisor: Laura Silvia Bahiense da Silva Leite

Department: Production Engineering

We present in this paper an implementation of metaheuristics for the Quadratic Assignment Problem (PQA) in programming language Go, as well as a proposal of *Íris* metaheuristics, formulated from the analysis of other metaheuristics' success and problems. We also propose alternatives for some traditional heuristics behavior using partial brute-force with Steinhaus-Johnson-Trotter (*Plain Changes*) algorithm that also compose, through a mapping to factorial number system (factoradic), the *Íris* proposal.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

No contexto da Pesquisa Operacional – e nas demandas reais de mercado – a relação entre qualidade de solução e recursos empregados para se buscar tal solução é um fator relevante. Problemas de Otimização Combinatória como o Problema Quadrático de Alocação (PQA), *Quadratic Assignment Problem (QAP)* da classe de complexidade **NP**-Difícil demandam quantidade de tempo de processamento e/ou memória que crescem assintoticamente em função exponencial ao tamanho do problema para os mais eficientes algoritmos atualmente conhecidos.

Definição Básica

O PQA é o problema de assinalar/designar facilidades a localidades pré-definidas de modo a minimizar o custo total de transporte de materiais entre as facilidades, dados os fluxos pré-definidos entre elas.

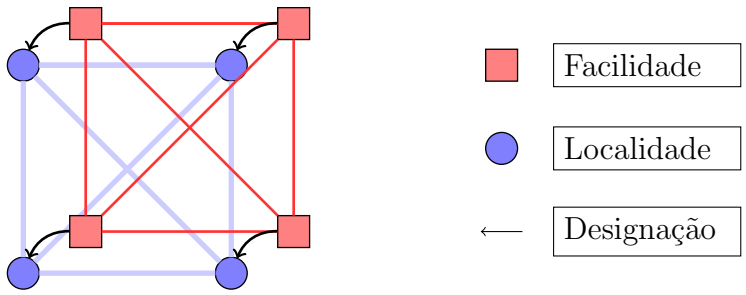


Figura 1.1: Ilustração de PQA (mapeamento de facilides a localidades).

Formulação Básica

Sendo a_{ij} o fluxo entre as facilidades i e j e b_{kl} a distância das localidades k e l, enquanto c_{ij} representa a parte linear, o custo fixo para alocar a facilidade i na posição j. O problema foi introduzidopor Koopmans e Beckmann (1957) [1]:

$$min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} a_{ij}b_{kl} \cdot x_{ik}x_{jl} + \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}, \tag{1.1}$$

s.a:

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n, \tag{1.2}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n, \tag{1.3}$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, n. \tag{1.4}$$

A condições (1.2), (1.3) e (1.4) restringem os valores $x_{i,j}$ da matriz X_n a valores binários e de modo que não haja mais de um valor 1 na mesma linha ou coluna. A maioria dos autores dispensa a parte linear $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$ em razão de sua simples resolução. Em relaxações do problema, a restrição de integralidade (1.4) é substituída permitindo valores reais no intervalo $[0, 1]$ e são adicionadas restrições para que a solução relaxada se viabilize gradativamente como veremos brevemente em Formulações (2.2).

$$0 \leq x_{ij} \leq 1 \qquad i, j = 1, \ldots, n \tag{1.5}$$

Utilizando-se da notação de permutação, seja π um mapeamento de facilidades a localidades (que pode ser abreviado para a notação de uma só linha).

$$\pi = \begin{pmatrix} 1 & 2 & \cdots & n \\ \pi(1) & \pi(2) & \cdots & \pi(n) \end{pmatrix} = (\pi(1), \pi(2), \ldots, \pi(n))$$

Desse modo PQA pode ser formulado com $\pi \in \Pi_n$ por

$$min \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}b_{\pi(i)\pi(j)} \tag{1.6}$$

uma vez que a permutação de índices de matrizes incorpora as restrições (1.2), (1.3) e (1.4) implicitamente.

Instâncias completas da formulação Koopmans-Beckman[1] correspondem a dois grafos orientados valorados incluindo o valor do laço para as relações de fluxo e distância. A correspondência entre a representação por grafo e matriz das relações de fluxo de uma instância completa de PQA é representada a seguir (figura 1.2). As referências em vermelho remetem aos valores que mudaram de posição na matriz em se aplicando a permutação π.

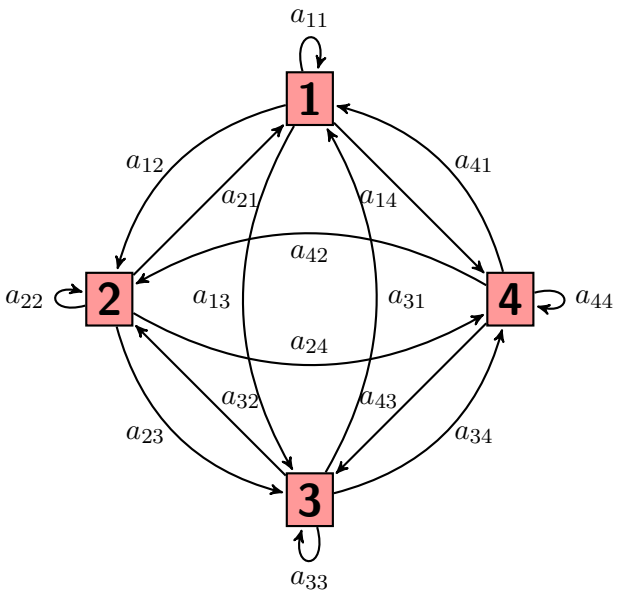
Diferentemente do Problema de Alocação Linear, para o qual o algoritmo Húngaro [2] encontra solução em tempo polinomial, para a versão quadrática do problema não se conhece um método tão eficiente. Atualmente instâncias de tamanho $n \simeq 30$ já requerem supercomputadores para que se possa provar a otimalidade de uma solução [3]. Em vista da necessidade prática de encontrar boas soluções para o referido problema sem contudo poder resolvê-lo à otimalidade para instâncias de tamanho razoavelmente grande (quiçá de proporções aplicáveis no mundo real) lançamos mão de heurísticas e metaheurísticas em favor de uma aproximação rápida.

Se por um lado a aproximação da solução ótima de um PQA pode requerer investimento exponencial de recursos computacionais, por outro uma pequena diferença entre soluções aproximadas pode definir o vencedor pela disputa de *market share*. Noutras palavras, pode ocorrer de o impacto dessa diferença também ser exponencial, de modo que a razão entre investimento e retorno mantenha-se razoavelmente proporcional para uma determinada faixa de investimento.

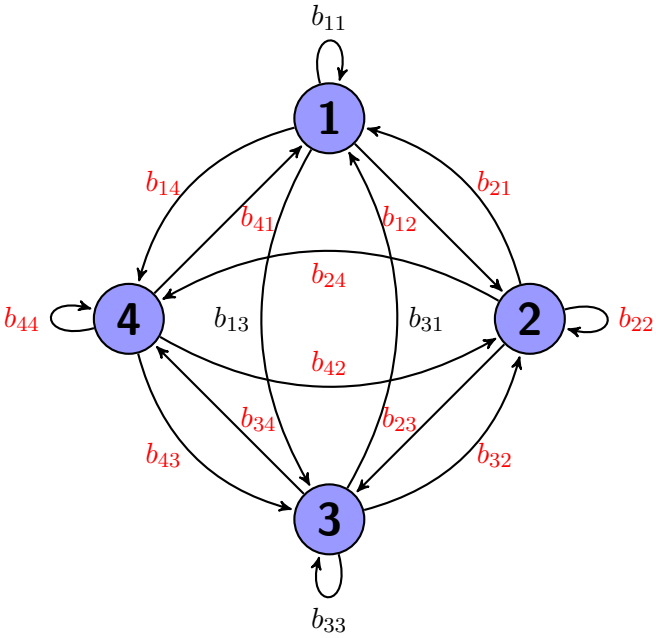
1.1 Motivação

O PQA é um dos mais difíceis e aplicáveis problemas de otimização combinatória [4]. Em razão de a eficiência ser um fator crítico para um solucionador de PQA, exploramos a implementação em uma nova e promissora linguagem de programação, Go[1], em vez de empregarmos programas já desenvolvidos. A possibilidade de encontrar em Go uma melhor ferramenta para desenvolver heurísticas complexas nos motivou a experimentar traduzir para ela algumas das metaheurísticas da literatura

[1] `https://golang.org`



$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$



$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & \mathbf{4} & 3 & \mathbf{2} \end{pmatrix}$$

$$B_\pi = \begin{bmatrix} b_{11} & b_{14} & b_{13} & b_{12} \\ b_{41} & b_{44} & b_{43} & b_{42} \\ b_{31} & b_{34} & b_{33} & b_{32} \\ b_{21} & b_{24} & b_{23} & b_{22} \end{bmatrix}$$

$$Custo = \sum_{i=1}^{4} \sum_{j=1}^{4} a_{ij} b_{\pi(i)\pi(j)} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{14} + a_{13}b_{13} + a_{14}b_{12} + \\ a_{21}b_{41} + a_{22}b_{44} + a_{23}b_{43} + a_{24}b_{42} + \\ a_{31}b_{31} + a_{32}b_{34} + a_{33}b_{33} + a_{34}b_{32} + \\ a_{41}b_{21} + a_{42}b_{24} + a_{43}b_{23} + a_{44}b_{22} \end{pmatrix}$$

Figura 1.2: Ilustração da correspondência grafo-matriz.

de PQA propondo mudanças por razão de idiomaticidade, ou seja, de exploração dos recursos primitivos da linguagem pelos quais um determinado algoritmo pode ser mais eficiente quando nela desenvolvido.

1.2 Objetivo

Temos por objetivo analisar o funcionamento - tanto independente quanto articulado - de metaheurísticas para o PQA em vista da eficiência e idiomaticidade de nossa implementação em linguagem de programação Go. Apresentamos também algumas hibridizações de heurísticas com algoritmos de enumeração permutatória (geração de permutações) a fim de melhorar suas funcionalidades em alguns casos, bem como a formulação de uma proposta de metaheurística baseada em hibridizações de heurísticas com um algoritmo permutatório e seu mapeamento para base numérica fatorádica, a qual nomeamos por Íris em função de seu peculiar caminho hamiltoniano sobre as $n!$ permutações quando representado em disposição circular de vértices.

1.3 Metodologia

Com o intuito de aprimorar o funcionamento de metaheurísticas para PQA revisamos algumas das mais importantes da literatura e as implementamos em linguagem Go a fim de comparar suas performances e analisar – a partir de testes empíricos – quais de suas características as tornam melhores que as demais sob algum aspecto. Ressaltamos veementemente que **os testes feitos não se pretendem completos nem "justos"** sob a enorme diversidade de aspectos que poderiam ser levados em conta para mensurá-los. Nossa intenção na exploração das metaheurísticas da literatura foi apenas a de verificar superficialmente o impacto de alterações de seus parâmetros e procedimentos internos. Assim, ainda que tenhamos apresentado uma tabela comparativa dos resultados e tempos das metaheurísticas para a bateria de testes de QAPLib, seu caráter é tão somente ilustrativo.

A feitura da presente dissertação procurou analisar a correspondência entre a noção intuitiva que a literatura apresenta sobre o comportamento das heurísticas com seus resultados em nossos testes empíricos e em alguns casos decidimos des-

continuar a exploração de algumas das possibilidades. Por exemplo, a busca local de primeira-melhoria se mostrou inexoravelmente melhor que a baseada em melhor-melhoria em quase todos os casos testados. Assim, salvo quando mencionarmos o contrário, a busca local empregada será a de primeira-melhoria. A metaheurística Simulated Annealing apresentou performance tão inferior às outras que abandonamos testes com a mesma já de início.

Muito do que pretendíamos em princípio teve de ser ponderado/reduzido – por exemplo a exploração algorítmica do problema na forma de grafo incompleto em *software*, o que otimiza o processo de busca e cálculo de custo e delta em instâncias esparsas – pois já havia complexidade suficiente e optamos por privilegiar o estudo consistente dos tópicos mais urgentes. Outros tópicos foram descontinuados por não apresentarem resultado satisfatório em testes preliminares como por exemplo a utilização de outras distribuições randômicas além da uniforme na construção gulosa de GRASP, o que não anula o valor de experiências futuras nesse sentido. Nosso principal critério de filtragem de experiências para publicação foi a de razoável estabilidade na bateria de testes do QAPLib. Os métodos que apresentaram melhora para apenas uma parte dos problemas foram cortados ainda que pudessem apontar otimizações específicas, pois suas análises expandiriam indefinidamente o tema, fugindo da proposta geral desse trabalho. Ocorre também que o teste completo do QAPLib é dispendioso e se optássemos por publicar experiências pouco testadas, arriscaríamos assumir conclusões ainda menos seguras, o que preferimos evitar.

Em suma, nossa metodologia consite no estudo e implementação de alguns dos metodos heurísticos de solução de PQA da literatura, bem como testes exploratórios com mensuração não-conclusiva do impacto relativo de variações internas do funcionamento das mesmas quanto à performance. Como resultado das noções depreendidas do referido estudo, decidimos por esboçar a produção de uma metaheurística que procurasse lidar complementarmente com algumas questões não contempladas pelas anteriores, principalmente o equilíbrio entre "variedade" e "profundidade" de busca no espaço de solução, sem contudo suprimir a necessidade daquelas.

1.4 Organização da Dissertação

Dos tópicos relevantes sobre o PQA, escolhemos o subconjunto que deve configurar uma apresentação o mais possível flúida e concisa.

No segundo capítulo, *Revisão Bibliográfica* apresenta sucintamente conceitos, formulações, heurísticas e metaheurísticas conforme a literatura adicionando comentários oriundos de nossas verificações empíricas. Mencionamos também as características da linguagem de programação Go que nos incentivaram a optar pela mesma para realização dos testes. Ao final do capítulo apresentamos uma breve explanação de algoritmos de geração de permutação e suas características exploráveis para nossos propósitos.

No terceiro capítulo, *Metodo Proposto* é apresentado enfocando a complementariedade das metaheurísticas GRASP, Busca Tabu e FANT e a definição da proposta de Íris.

No quarto capítulo, *Resultados e Discussões*, apresentamos os dados que fundamentam as conclusões apresentadas adiante.

No quinto capítulo, *Conclusões* apresentamos, além das conclusões, as ressaltas sobre a validade do que se pôde concluir a partir dos testes e também comentamos sobre o que se pretendia fazer, o que foi feito e o que se poderá fazer no futuro com relação ao presente trabalho.

No primeiro apêndice, *Apêndice A* estão as tabelas com os testes referidos ao longo da dissertação.

Capítulo 2

Revisão Bibliográfica

> Clear is better than clever.
>
> Rob Pike [5]

Revisamos a literatura de PQA abordando algoritmos exatos, mistos, heurísticas e metaheurísticas, mas privilegiando o enfoque em metaheurísticas com detalhamento suficiente para auxiliar novas implementações das mesmas.

2.1 Conceitos e Notações

O esquema de codificação (serialização) da resposta de um problema deve ser o menos prolixo possivel sem tornar-se ambíguo tanto por questão de conveniência quanto para evitar que se subverta o cálculo de complexidade computacional de um método de solução, uma vez que o mesmo é mensurado pela relação entre quantidade de passos até a solução e o "tamanho" da resposta para uma instância [6]. Problemas de otimização combinatória relacionados ao mapeamento entre elementos de dois distintos conjuntos assumem variações quanto ao modo com que apresentam uma sequencia de números identificadores como resposta, geralmente em listas simples ou aninhadas com ou sem ordenação. As listas ordenadas constumam ser utilizadas para suprimir o primeiro identificador de um par mapeado quando o mesmo pode ser inferido pela posição na lista, que é o caso da representação de uma permutação em apenas um vetor.

Uma permutação (lista ordenada de índices sem repetição, ex: $[b, c, a, d]$) é suficiente para representar a solução de um PQA, enquanto um Problema Generalizado de

Alocação é representado por uma lista de adjacência ou equivalente (lista ordenada de listas não ordenadas de elementos sem repetição, ex: $[\{b\}, \{c, a\}, \{d\}]$). Para um problema de agendamento de tarefas a múltiplos trabalhadores ou máquinas se pode utilizar uma estrutura semelhante à de PGA porém com ordenação interna dos grupos (lista ordenada de listas ordenadas de elementos sem repetição, ex: $[[b], [c, a], [d]]$). Cada uma dessas serializações podem ser traduzidas para representações computacionais diferentes a depender da conveniência para os métodos que as utilizem. Em nosso trabalho daremos enfoque à permutação.

Permutações

Uma permutação ϕ pode ser escrita de duas formas equivalentes,

$$\phi \ = \ (1, \ 4, \ 3, \ 2) \ = \ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 3 & 2 \end{pmatrix} \ ,$$

sendo que a primeira omite os valores que podem ser inferidos a partir do índice. Toda permutação corresponde a uma e apenas uma matriz binária. Essa matriz tem valor 1 na posição x_{ij} sendo i a posição correspondente do índice no vetor permutação e j o valor que há nessa posição. Noutras palavras, o vetor permutação preenche com unidades a matriz binária pela posição das colunas. Seja $e(j)$ o vetor coluna com 1 na posição j e zero nas demais.

$$\phi \ = \ (1, \ 4, \ 3, \ 2) \ \Leftrightarrow \ X_\phi \ = \ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \ = \ \begin{bmatrix} e(1) \\ e(4) \\ e(3) \\ e(2) \end{bmatrix}$$

Há também a forma cíclica de representar a permutação (que diferenciaremos da primeira pela supressão das vírgulas e/ou presença de mais de um par de parênteses).

$$\phi \ = \ (2 \ 4)(1)(3)$$

Sendo que ambas equivalem.

$$\phi \ = \ (2 \ 4)(1)(3) \ = \ (1, \ 4, \ 3, \ 2)$$

Na forma cíclica, para cada grupo de índices com mais de um índice eles sofrem uma rotação de modo que cada índice é permutado com o próximo e o último com o primeiro (em se comparando com a notação de um vetor). Por exemplo, as seguintes permutações são idênticas:

$$(1)(2\ 3\ 4\ 5)(6) \ = \ (1,\ 3,\ 4,\ 5,\ 2,\ 6)\ .$$

A forma cíclica é mais interessante em se trabalhando com problemas de caminho ou circuito num grafo, por exemplo no Problema do Caixeiro Viajante (*TSP, Travelling Salesman Problem*). Para o PQA a notação na forma de um vetor é a mais utilizada na literatura e é a que também empregaremos. Permutação cíclica também possui representação matricial biunívoca.

$$(1)(2\ 3\ 4\ 5)(6) \ = \ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Ressaltamos que há uma diferença importante entre a representação tradicional de vetores e matrizes na forma didática e na forma com que essas estruturas são tratadas computacionalmente[1]. Quase sempre no desenvolvimento de *software* os índices começam por 0 em vez de 1, assim um vetor é indexado no intervalo $[0, n-1]$, bem como os índices $i, j, k...$ de de matrizes n-dimensionais. Além disso, quanto à disposição em memória, as matrizes n-dimensionais são alocadas em sequencia como um só vetor e os índices são recalculados como múltiplos de intervalos para se definir o endereço da célula a que se quer referir. Essa diferença é relevante em se considerando o modo eficiente de iterar por seus valores. Assim, haverá diferença no tratamento dos índices entre a representação em fórmulas, pseudo-código e código-fonte.

[1]Nas arquiteturas de microcomputadores e linguagens de programação mais populares, não como padrão universal ou necessário.

2.2 Formulações

Koopmans e Beckmann em "*Assignment problems and the location of economic activities*" (1957) [1] apresentaram o problema quadrático de alocação, no qual procurava-se otimizar a interação das atividades econômicas em consideração à proximidade entre as mesmas. Assumiu-se que "a suposição de que o benefício de uma atividade econômica numa atividade não depende do uso de outras localidades é um tanto inadequada para as complexidades das decisões de localização".

Considerando n facilidades e n localidades e a matriz $[a_{ki}]$ na qual o elemento a_{ki} representa o rendimento da operação da facilidade k na localidade i, independente da localização de outras facilidades (linear). Seja a matriz $[b_{kl}]$ de números não-negativos, com $b_{kl},\ k \neq l;\ k,l = 1,\ldots,n$ representando o fluxo do transporte de mercadorias da facilidade k para l e outra matriz $[c_{ij}]$ com $i \neq j$ representando o custo de transporte da localidade i para j e $[p_{ki}]$ uma matriz de permutação.

$$max \sum_{k,i} a_{ki}p_{ki} - \sum_{k,l}\sum_{i,j} b_{kl}p_{ki}c_{ij}p_{lj} \tag{2.1}$$

sujeito a:

$$p \in P_n \tag{2.2}$$

pressupondo:

$$b_{kk} = c_{kk} = 0 \qquad k = 1,\ldots,n, \tag{2.3}$$

$$c_{ij} \leq c_{ik} + c_{kj} \qquad i,j,k = 1,\ldots,n, \tag{2.4}$$

$$b_{kl} \geq 0 \qquad k \neq l; k,l = 1,\ldots,n, \tag{2.5}$$

$$c_{ij} > 0 \qquad i \neq j; i,j = 1,\ldots,n. \tag{2.6}$$

Posteriormente a formulação foi tomada de modo semelhante como problema de minimização (2.7) do componente quadrático (e parte linear com sinal contrário) e alguma variação nas restrições (2.3) e (2.4). A seguir apresentaremos alguns tipos de formulação de PQA presentes na literatura.

$$min \sum_{k,l} \sum_{i,j} b_{kl} p_{ki} c_{ij} p_{lj} - \sum_{k,i} a_{ki} p_{ki} \tag{2.7}$$

Formulação por Programação Inteira (PLI)

Referente à formulação de Koopmans e Beckman [1] tal como foi tomada pela literatura.

Defn. Dados:

F_{n^2} :	$[f_{ij}]$	Matriz de fluxo	(entre facilidades i e j)
D_{n^2} :	$[d_{kl}]$	Matriz de distância	(entre localidades k e l)
B_{n^2} :	$[b_{ij}]$	Matriz de custo	(componente linear)

Var. Decisão:

X_{n^2} :	$[x_{ij}]$	Matriz de decisão	(permutação)

$$min \quad \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} f_{ij} d_{kl} x_{ik} x_{jl} \; + \; \sum_{i,j}^{n} b_{ij} x_{ij} \tag{2.8}$$

$$s.a. \quad \sum_{i=1}^{n} x_{ij} = 1 \qquad 1 \leq j \leq n \; , \tag{2.9}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad 1 \leq i \leq n \; , \tag{2.10}$$

$$x_{ij} \in \{0, 1\} \qquad 1 \leq i, j \leq n \; . \tag{2.11}$$

As restrições (2.9), (2.10) e (2.11) podem ser simplificadas como

$$x_{ij} \setminus X_{n \times n} \in \Pi_n \tag{2.12}$$

pois referem-se às condições em que a matriz X é uma matriz permutação de n elementos.

Lawler propôs uma versão mais abrangente (1963)[7] da qual a versão Koopmans-Beckman é um caso especial.

$$min \sum_{i,j=1}^{n} \sum_{k,p=1}^{n} c_{ijkp} x_{ij} x_{kp} \tag{2.13}$$

s.a. (2.9), (2.10), (2.11)

Formulação por Programação Inteira Mista (PLIM)

Formulação apresentada por Frieze & Yadegar (1983)[8]. A linearização do PQA é quase sempre inconveniente [4], mas possibilitou a determinação de limitantes inferiores.

Defn. Dados:

C_{n^4} :	$[c_{ijkp}]$	Matriz de custo	(componente quadrático)
B_{n^2} :	$[b_{ij}]$	Matriz de custo	(componente linear)

Var. Decisão:

X_{n^2} :	$[x_{ij}]$	Matriz de decisão	(permutação)

$$min \quad \sum_{i,j=1}^{n} \sum_{k,p=1}^{n} c_{ijkp} x_{ij} x_{kp} \; + \; \sum_{i,j}^{n} b_{ij} x_{ij} \tag{2.14}$$

$$s.a. \quad \sum_{i=1}^{n} x_{ij} = 1 \qquad 1 \leq j \leq n \; , \tag{2.15}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad 1 \leq i \leq n \; , \tag{2.16}$$

$$\sum_{i=1}^{n} y_{ijkp} = x_{kp} \qquad 1 \leq j, k, p \leq n \; , \tag{2.17}$$

$$\sum_{j=1}^{n} y_{ijkp} = x_{kp} \qquad 1 \leq i, k, p \leq n \; , \tag{2.18}$$

$$\sum_{i=1}^{n} y_{ijij} = x_{kp} \qquad 1 \leq i, j, p \leq n \; , \tag{2.19}$$

$$\sum_{j=1}^{n} y_{ijij} = x_{kp} \qquad 1 \leq i, j, k \leq n \; , \tag{2.20}$$

$$y_{ijij} = x_{ij} \qquad 1 \leq i, j \leq n \; , \tag{2.21}$$

$$x_{ij} \in \{0, 1\} \qquad 1 \leq i, j \leq n \; , \tag{2.22}$$

$$0 \leq y_{ijkl} \leq 1 \qquad 1 \leq i, j, k, l \leq n \; . \tag{2.23}$$

A formulação utiliza-se de n^4 variáveis reais, n^2 variáveis booleanas e $n^4 + 4n^3 + n^2 + 2n$ restrições.

Formulação por Permutação

Defn. Dados:			
$F_{n\times n}$:	$[f_{ij}]$	Matriz de fluxo	
$D_{n\times n}$:	$[d_{\pi(i)\pi(j)}]$	Matriz de distância	(com índices permutados)
S_n :		Conj. de permutações	(de n elementos)
Var. Decisão:			
π :		Permutação	

$$min_{\pi\in S_n} \quad \sum_{i,j=1}^{n} f_{ij} d_{\pi(i)\pi(j)} \tag{2.24}$$

Formulação Traço

Defn. Dados:		
$F_{n\times n}$:	Matriz de fluxo	
$D_{n\times n}$:	Matriz de distância	
Π :	Conjunto de permutações	
Var. Decisão:		
$X_{n\times n}$:	Matriz de decisão	(permutação)

$$min_{X\in\Pi} \quad tr(F.X.D^t.X^t) \tag{2.25}$$

Essa formulação explora com recursos de ágebra linear as propriedades da função traço[2] de uma matriz para determinar limitante inferiores.

A mesma pode ser simplificada em problemas cuja matriz de distância é simétrica.

$$min_{X\in\Pi} \quad tr(F.X.D.X^t) \qquad sse: \ D = D^t \tag{2.26}$$

[2]Soma dos elementos da diagonal principal da matriz.

Formulação Compacta

Um PQA na forma Koopmans-Beckmann pode ser formulado numa forma mais compacta se definirmos o produto interno entre duas matrizes quadradas como

$$\langle A, B\rangle := \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}b_{ij} \ . \tag{2.27}$$

Assim, a forma completa envolvendo a parte linear pode ser alternativamente definida como se segue – independentemente de simetria da matriz de distância.

Defn. Dados:		
$F_{n\times n}$:	Matriz de fluxo	
$D_{n\times n}$:	Matriz de distância	
$B_{n\times n}$:	Matriz de distância	(componente linear)
X_n :	Conjunto de permutações	
Var. Decisão:		
$X_{n\times n}$:	Matriz de decisão	(permutação)

$$min \quad \langle F, XDX^T\rangle + \langle B, X\rangle \ , \tag{2.28}$$

$$s.a. \quad X \in \mathbf{X}_n \ . \tag{2.29}$$

2.3 Complexidade e Dificuldades

A publicação por Kuhn (1955) [2] do Algoritmo Húngaro para o Problema Linear de Alocação finalmente possibilitou resolver instâncias do mundo real impraticáveis para quaisquer computadores e algoritmos até então existentes [9]. Sua publicação foi um revolucionário episódio do bem-conhecido e recorrente fato de que algoritmos simples podem incorporar propriedades matemáticas (prescindindo de explicitá-las por completo) e se comportar, ainda assim, de modo a preservá-las, viabilizando a solução de problemas outrora irrealizáveis pela dispendiosidade dos procedimentos requeridos. Noutras palavras, *podemos definir algoritmos que "imitem" outros* quanto à entrada e saída de dados, mas que sejam internamente muito mais simples[3]

Muito embora o problema linear tenha sido sanado – e se haver concebido daí um fundamento da Otimização Combinatória [9] – uma série de outros problemas se mostraram irredutíveis polinomialmente ao mesmo método (ou quaisquer outros métodos polinomiais), incorrendo na classificação de problemas por sua dificuldade relativa. A *Teoria de Análise de Algoritmos* estuda análise de complexidade de algoritmos, enquanto a *Teoria da Complexidade Computacional* estuda a classificação de problemas com base na complexidade dos algoritmos que os resolvam.

O PQA é classificado por complexidade computacional como NP-Difícil (*NP-Hard*), o que significa que não se conhece cientificamente algum algoritmo capaz de encontrar a solução ótima global para as instâncias do problema de modo que a quantidade de passos para se encontrar tal solução cresça assintoticamente em relação ao tamanho do problema, de acordo com uma fórmula que possa ser representada na forma de um polinômio.

O problema de se provar que uma heurística é melhor dentre outras para se resolver um problema NP-Difícil pode vir a ser também muito difícil. A fim de que o empreendimento de dissertar sobre metaheurísticas para o problema não se torne tão ou mais complicado que o próprio problema, acedemos à proposta limitada/simplificada de apresentar testes empíricos e conjecturas ligadas aos mesmos.

[3]Entretanto nem todo tipo de algoritmo é facilmente simplificável. Algoritmos redutíveis a autômatos finitos são os mais bem-estabelecidos quanto às simplificações, mas são também os mais limitados.

Escalada de Complexidade

A complexidade computacional de um problema não aumenta apenas pela quantidade de dados envolvidos ou do número de soluções possíveis em relação ao tamanho da instância, mas substancialmente pelo potencial de exploração de inferências que ajudem a podar[4] significativamente o espaço de solução durante a resolução do problema. A seguir, apresentamos problemas com o mesmo número de soluções para instâncias de mesmo tamanho, mas de complexidade bastante diferente.

Problema de Ordenação (Ordn. não-crescente):

$$min \sum_{i=1}^{n} c_{\pi(i)} * i \tag{2.30}$$

Problema de Alocação Linear (PAL):

$$min \sum_{i=1}^{n} c_{i\pi(i)} \tag{2.31}$$

Problema Quadrático de Alocação (PQA):

$$min \sum_{i=1}^{n} c_{ij\pi(i)\pi(j)} \tag{2.32}$$

Problema	Tam.	Qt.Sol.	Compl.	Classe	Ind.lin.	Ind.par.
Ordn.	n	$n!$	$O(n\ log\ n)$	Linear	sim	sim
PAL	n	$n!$	$O(n^3)$	Polinomial	sim	**não**
PQA	n	$n!$	$O(n!)$	Exponencial	**não**	**não**

Tabela 2.1: Comparação de complexidade de problemas de mesmo tamanho.

A independência do valor no teste par-a-par (Ind.par.) – a não interferência da ordem de outros pares na validade da ordem de cada par de elementos no vetor solução – é o que torna a ordenação simples muito eficiente, o que não ocorre no PAL nem no PQA. A independência linear (Ind.lin.) – a não interferência do custo

[4]Desprezar uma partição do espaço de solução por algum critério de interesse de modo a não mais buscar naquele intervalo.

de alocação de um elemento no custo de outro – é a característica que possibilita a aplicação do algoritmo húngaro à alocação linear, o que não ocorre no PQA.

2.4 A Linguagem de Programação Go

A emergência de uma nova linguagem de programação, Go, desenvolvida e publicada em código-aberto pela Google Inc, de uso irrestrito e gratuito, nos forneceu ferramentas interessantes para lidar com heurísticas não só por sua disponibilidade, mas também por características especiais da linguagem que a torna versátil e eficiente.

Na última década, assistimos ao desenvolvimento de uma enorme diversidade de linguagens de programação, as quais procuraram resolver as demandas computacionais já identificadas até então, bem como os novos problemas que só se mostraram urgentes por desdobramentos históricos do uso das linguagens anteriores. Desse modo, ocorreu um processo de "seleção científica e pragmática" que permitiu amadurecer os conceitos sobre como precisamos que sejam as linguagens de programação de propósito geral.

Por considerarmos que a definição concreta de algoritmos e heurísticas em uma linguagem de programação é bastante relevante para se avaliar o desemprenho e qualidade dos mesmos, abordaremos superficialmente a própria linguagem Go em seus aspectos relevantes para nossa proposta, bem como apresentaremos parte do código-fonte no presente documento (enquanto a totalidade do código-fonte ficará disponível para apreciação em formato adequado).

Tratamos por 'linguagem de programação' uma linguagem com algum nível de regularidade [5] pela qual se pode definir para um computador como ele deve executar um ou mais procedimentos incluindo algoritmos complexos a partir de suas funções primitivas, ou uma abstração em alto-nível delas, fornecida pela dada linguagem.

Atualmente, as mais difundidas linguagens de programação são classificadas, quanto à sintaxe, como 'linguagens livre de contexto', i.é, a validade sintática de suas sentenças não depende das sentenças das quais elas fazem parte (ou quaisquer componentes linguísticos exteriores a ela), mas apenas da validade de suas sentenças internas (e/ou disposição dos morfemas)[6].

[5] Conforme a hierarquia de linguagens de Chomsky[10].

[6] O conjunto dessas linguagens é idêntico ao conjunto de linguagens aceitas por um autômato

2.5 Exemplos de Aplicação do PQA

2.5.1 Layout de um Hospital

A funcionalidade da arquitetura de um hospital influencia sobremaneira na qualidade do serviço prestado e, consequentemente na saúde dos clientes podendo em algumas situações vir a ser o fator crítico de perda ou não de vidas. Suponhamos um projeto de construcão de hospital que, dentre as muitas características passíveis de otimização, tenhamos de nos ocupar apenas da disposição dos serviços em áreas pré-definidas.

A partir de uma pré-programação que levou em conta a quantidade de tráfego entre as áreas e a importância do tipo de tráfego, gerou-se uma matriz $F_{n\times n}$ com os coeficientes que representam os dois fatores de maneira abstrata enquanto "urgência" de tráfego de uma área para outra, incluindo todas as áreas do hospital. Assim, na matrix F, f_{ij} representa essa urgência em se poder transitar do serviço i para o j, (p.ex, do elevador para a sala-de-parto) e f_{ji} da direção oposta. Dispomos também de uma matriz $D_{n\times n}$, na qual d_{ij} representa a distância da localidade i para a j.

Vale salientar que não se pode presumir que qualquer uma das matrizes seja simétrica, pois mesmo a matriz de distância pode referir-se a caminhos não retilíneos, diferentes para ida e para volta. Evidentemente, é melhor que designemos/assinalemos os serviços às localidades de modo a minimizar as distâncias entre os serviços cujo trânsito tem maior coeficiente de urgência. Assim, pretendemos encontrar uma permutação das pré-definidas localidades em relação aos serviços que minimize o somatório dos produtos urgência-distância ($f_{ij} \times d_{p(i)p(j)}$) entre todos os serviços vinculados às localidades conforme a permutação/solução.

Numa aplicação real, uma boa solução poderia ser aprimorada pela utilização de um software de simulação de eventos discretos, cuja análise estatística poderia apontar conflitos de funcionalidade não depurados pela modelagem empregada até então.

---

com pilha/lista (*pushdown automata*).

2.5.2 Isomorfirsmo de Grafos

A verificação de que dois grafos (não valorados) são isomorfos pode ser formulada como um PQA em se traduzindo ambos os grafos para uma versão valorada de cada na qual o valor do arco é 1 quando existe um arco entre o par de vértices e 0 quando não existe. Os grafos são isomorfos se, e somente se a solução ótima do PQA de maximização tem valor idêntico ao da cardinalidade do conjunto de arcos de ambos os grafos.

2.5.3 Problema do Caixeiro Viajante

Um Problema de Caixeiro Viajante (*Travelling Salesman Problem, TSP*) pode ser formulado como um PQA utilizando-se uma matriz de distâncias entre cidades e uma matriz de permutação cíclica. A solução ótima para esse PQA é a soluçcão ótima para o Problema do Caixeiro Viajante interpretando-se a permutação na forma cíclica, como uma sequência de cidades a serem visitadas conforme a ordem dos índices da solução.

2.6 Heurísticas

Distinção entre Algoritmos, Heurísticas e Metaheurísticas:

Algoritmos

Algoritmos são fórmulas procedurais – baseadas em sequencias de passos – que, partindo de uma dada situação, transformam-na em outra de modo previsível. Algoritmos podem conter, e em geral contém, passos condicionais, i.é. passos cuja realização dependem do resultado da verificação de uma característica da situação[7].

Heurísticas

Heurísticas são um tipo de algoritmo com características especiais, a saber, a possível presença de variáveis pseudo-randômicas e, essencialmente, um proceder que reflete um saber intuitivo sobre o conteúdo de sua aplicação, que entretanto não assegura um resultado, mas que se supõe realizar de modo satisfatório alguma tarefa.

[7] Tecnicamente 'estado'.

Entendemos que o melhor emprego de heurísticas esteja ligado à presunção implícita de conclusões indutivamente fortes (ou probabilisticamente aceitáveis), mesmo quando não dedutivamente válidas, decidindo com alguma arbitrariedade sobre o que é impassível de certeza – dadas as informações até então disponíveis e a dipendiosidade de tempo e memória para obtenção das que faltam.

Em consonância com a literatura de otimização, trataremos por "algoritmo" ou "algoritmo exato" o tipo de algoritmo que se define em oposição à heurística, ainda que estritamente essa se refira a um subconjunto daquele. Algoritmos estão, portanto, associados a soluções por procedimentos determinísticos que almejam senão o ótimo global. Heurísticas, entretando, estão associadas à busca por "ótimos locais" (soluções sub-ótimas) e sua qualidade é ligada à relação entre qualidade de soluções e o tempo consumido para encontrá-las.

Metaheurísticas

Metaheurísticas são composições robustas que geralmente incluem mais de uma heurística. O termo 'hiperheurística' é menos empregado e se refere a uma composição de metaheurísticas.

2.6.1 Construção "Gulosa"

Um algoritmo 'guloso' é caracterizado pela tomada de decisão localmente ótima para cada etapa de uma solução heurística com a esperança de chegar ao ótimo global ou se aproximar dele. Desse modo, a construção gulosa da solução inicial do PQA se trata do gradativo acréscimo de um elemento (índice de vértice) no vetor permutação até que sua cardinalidade se torne idêntica ao tamanho da instância.

Ilustrando-se o conceito por outra aplicação, no chamado "problema da mochila inteira" – que consiste em preenchê-la o máximo possível com objetos que variam em peso e valor em vista de maximizar o valor total do que pode ser integralmente inserido na mochila – a construção gulosa poderia se dar privilegiando o valor ou quanto ao espaço disponível. Assim, uma construção gulosa quanto ao valor dos objetos seria construída colocando-se dentro da mochila primeiramente os objetos de maior valor até seu limite de capacidade. No caso de uma abordagem gulosa quanto à capacidade, se poderia procurar combinações dos objetos disponíveis que

minimizassem a sobra da capacidade. Ambas as abordagens desse exemplo procuram gerar soluções que resolvem uma parte do problema, porém desprezando momentaneamente outra.

No caso do PQA, uma possível abordagem gulosa seria escolher aleatoriamente um par de vértices e agregar gradativamente os demais vértices escolhendo qual dos disponíveis, em cada iteração da construção, mais contribuiria para minimizar a função-objetivo. Esse procedimento, porém, rapidamente levaria à degradação da variedade de novas soluções gulosas geradas (o que impediria o progresso na busca por boas soluções nas gerações adiante). Por essa razão, metaheurísticas como GRASP adicionam um componente de aleatoriedade/randomismo que em lugar de escolher a melhor opção, lista as possibilidades em ordem não-crescente de interesse e, dentro de uma parte inicial dessa lista, escolhe aleatoriamente alguma das possibilidades.

2.6.2 Busca Local

A busca local é um processo de melhoria gradativa de uma solução intimamente ligado à noção de 'estrutura de vizinhança'[8]. Definindo-se uma ou mais estruturas de vizinhança entre as soluções de um problema, é possível fazer pequenas alterações na solução – transitando-se condicionalmente para uma solução vizinha a cada passo – visando sua melhoria quanto à função-objetivo. A estrutura de vizinhança do PQA é tradicionalmente definida em função da troca de pares de índices do vetor solução. Assim, o número de soluções vizinhas a qualquer solução de instância tamanho n é ${}^{n}C_{2}$.

A busca local apresenta algumas variações dentro das metaheurísticas, mas principalmente se pode classificar entre "**primeira melhoria**" ou "**melhor melhoria**". Curiosamente, em muitas das heurísticas se preferiu adotar o critério de "primeira melhoria" (ou seja, que muda a solução antes mesmo de terminar de verificar todas as soluções vizinhas), pois pode evitar 2.1 a aderência a ótimos locais de baixa qualidade, sem escapar, todavia, de ótimos locais encontrados em deformações mais abrangentes/profundas no espaço de solução, onde geralmente se encontram melho-

[8]Um grafo formal ou atual que liga umas soluções às outras (às soluções "vizinhas") por um procedimento simples de alteração de uma que a transforme noutra. Preferivelmente esse grafo é fortemente conexo.

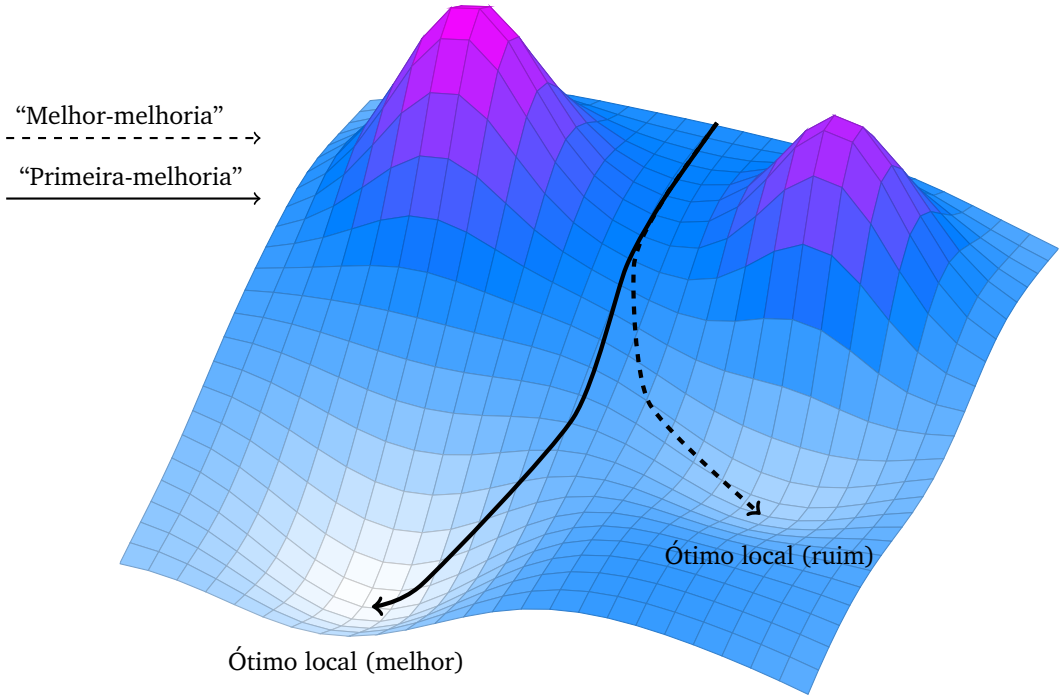


Figura 2.1: Ilustração de Busca Local de minimização.

res ótimos locais ou o ótimo global. A expressão ’ótimo local’ refere-se a soluções que, comparadas com as vizinhas, não acarretariam melhora quanto à função-objetivo se fossem trocadas por alguma delas.

2.6.3 Path-Relinking

O procedimento heurístico Path-Relinking consiste na comparação entre duas distintas soluções e na gradual tranformação da primeira na segunda. Foi primeiro introduzido por Glover como parte da Busca Tabu.

No caso do PQA, ocorre pela troca de posição de dois elementos da solução por vez. A escolha do par de elementos que são trocados segue o princípio de “melhor melhoria” (ou menos pior das pioras quando não há opção de melhora). Noutras palavras, trata-se de percorrer o caminho entre duas soluções em busca de alguma melhor que uma delas ou ambas. Ainda que durante o processo haja piora da solução atual, a melhor solução encontrada fica salva noutra variável até o final do processo. Em razão de Path-Relinking exigir mais de uma solução, seu uso geralmente acompanha uma lista de soluções elite e a escolha da solução guia requer um número mínimo de elementos distintos da solução origem, preferencialmente

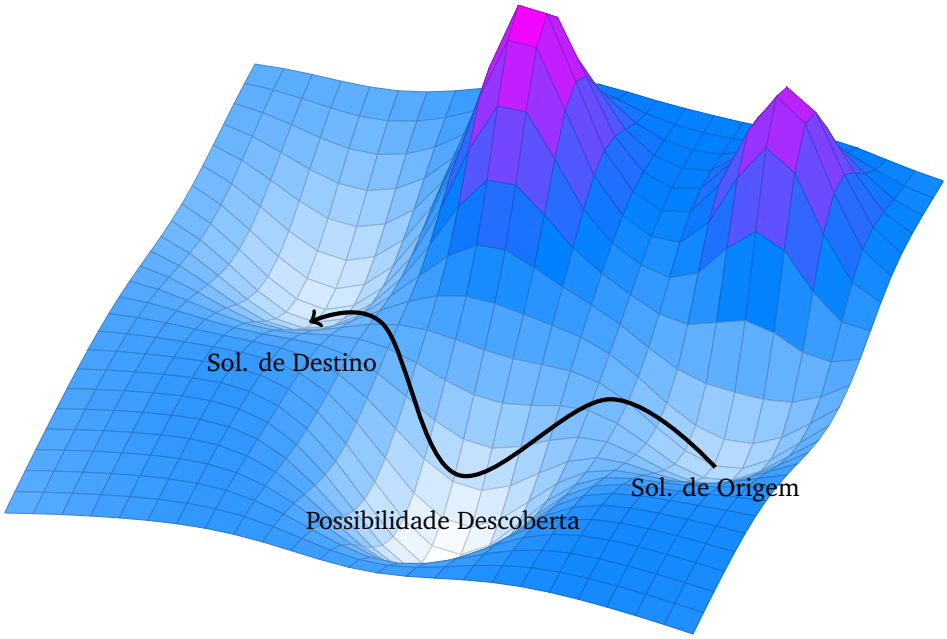


Figura 2.2: Ilustração de Path-Relinking de minimização.

maior que três, uma vez que a busca local provavelmente já haveria encontrado soluções tão pouco diferenciadas.

É importante notar que a melhor solução encontrada durante o Path-Relinking ainda está sujeita à melhora por busca local uma vez que o caminho percorrido entre a solução de origem e destino fica restrito a uma faixa de variação. Assim a solução encontrada poderia estar à beira de um declive não explorado no caminho, como ilustrado na figura 2.2.

Tabela 2.2: Comparação dos papéis das heurísticas.

	Construção Gulosa	**Busca Local**	**Path-Relinking**
Início independente:	**sim**	não	não
Melhora individual:	não	**sim**	não
Melhora combinada:	não	não	**sim**

2.7 Metaheurísticas

As metaheurísticas GRASP, Busca Tabu e FANT foram escolhidas foram revisadas. Simulated Annealing oi descartada em testes preliminares por baixa performance em testes para PQA.

2.7.1 GRASP

Greedy **R**andomized **A**daptive **S**earch **P**rocedure foi desenvolvida por Mauricio G.C. Resende e Thomas A. Feo[11]. GRASP procura consolidar as boas técnicas heurísticas que preservam a diversidade de soluções iniciais, ao mesmo tempo com tendência gulosa e aplicação posterior de busca local.

Considerando que um algoritmo "guloso" sem interferência de função randômica geraria sempre o mesmo resultado, GRASP inicia com um procedimento "guloso-randomizado", isto é, ordena as possibilidades de forma decrescente de aumento no valor de custo e escolhe aleatoriamente uma das possibilidades (sendo a lista restrita para escolha na proporção do parâmetro alfa).

A primeira fase do GRASP consiste em: listar os arcos da matriz de fluxo de modo decrescente, os de distância de modo crescente e formulando-se uma lista de pares nessa ordem (a qual é cortada na proporção do parâmetro beta). Do par de arcos sorteado se definem os dois primeiros vértices e os demais são gradativamente sorteados, na segunda fase (de uma lista ordenada de forma gulosa restrita na proporção de alfa). Em seguida aplica-se a Busca Local e/ou PathRelinking.

Construção da solução gulosa randômica

O procedimento de construção tem duas fases: a primeira gera os dois primeiros vértices da solução a partir de um algoritmo guloso-randômico de formação de n^2-n pares distância-fluxo (pares de arcos) do seguinte modo:

Primeira fase:

Listam-se as $n^2 - n$ distâncias d_{ij} num vetor e ordena-se tal vetor em ordem crescente de valores. Faz-se o mesmo com os fluxos, porém ordena-se de modo decrescente. Em um novo vetor, de mesmo tamanho, de pares distância-fluxo formado a partir da mesma posição na qual eles se encontravam nos vetores anteriores, aplica-se novamente a ordenação crescente, agora nesses pares, cujos valores são assim calculados: $f_{kl}d_{ij}$. Feita tal ordenação, ela pode ser guardada até o final de todas as iterações do GRASP a fim de economizar processamento na primeira fase da construção gulosa-randômica. Escolhe-se uma posição randômica integer ((n2-n)*rand())da qual dois vértices são escolhidos (dados d_{ij} e f_{kl}, temos, no vetor

solução, k, na posição i e l na posição j). Tendo o vetor de solução já preenchido em duas posições, $\phi = (j_1, l_1), (j_2, l_2)$, é preciso preencher as demais $n - 2$ posições. Isso é feito na segunda fase.

Algorithm 2.1 Pseudo-código de GRASP

Pseudo-código de GRASP

```
1: função GRASP (PQA, MaxIter, SementeRand)
2:     φ ← ∅                                   ▷ Melhor solução (a ser encontrada)
3:     MelhorCusto ← ∞                          ▷ Inicialização instrumental
4:     InicializaFuncRand(SementeRand)          ▷ Exemplo: Tempo.Agora()
5:     repetir
6:         sol ← ConstróiSolução(PQA)           ▷ Nova sol. gulosa randômica
7:         sol ← BuscaLocal(PQA, sol)
8:         se Custo(PQA, sol) ¡ MelhorCusto então
9:             φ ← sol
10:            MelhorCusto ← Custo(PQA, sol)
11:        fim se
12:    até-que CritérioDeParadaSatisfeito(MaxIter)
13:    retorna φ
14: fim função
```

Segunda fase:

Para cada um dos demais valores que faltam entrar no vetor, calcula-se o custo que acrescentaria para para cada posição livre. Todas as combinações são acumuladas numa lista que é, então ordenada de modo crescente. $C_{ik} \leftarrow \sum_{(j,l)\in\phi} f_{ij} d_{kl}$ Sendo len() a função que retorna o tamanho da lista, a posição escolhida é calculada pela fórmula integer $int(\alpha * len(list) * rand())$. O valor referente a essa posição na lista, é acrescentado em ϕ. Se a lista ainda não estiver completa, volte para o passo 1 dessa segunda fase. Caso contrário, a construção gulosa-randômica está concluída para essa iteração do GRASP.

Algorithm 2.2 GRASP Fase 2

Pseudo-código da Fase 2

```
1: função FASE2(α,(j1, l1),(j2, l2))          ▷ contexto: GRASP/ConstróiSolução/
2:     m ← 0
3:     para i = 3 . . . n faça
4:         lista ← []                                  ▷ (re)inicializa lista
5:         para k = 1 . . . n faça
6:             se (i, k) ∉ φ então
7:                 C_ik ← Σ_{(j,l)∈φ} f_ij d_kl
8:                 ACRESCENTANALISTA(C_ik,lista)
9:                 m ← m + 1
10:            fim se
11:        fim para
12:        ORDENANÃODESCRESCENTE(lista)
13:        φ ← φ ∪ {lista[int(α ∗ m ∗ rand())]}
14:    fim para
15:    retorna φ
16: fim função
```

2.7.2 FANT

Ant Systems é uma metaheurística baseada numa metáfora do comportamento das formigas Taillard [12]. Assim como muitos avanços científicos começaram por imitar a natureza e posteriormente abandonaram as características metafóricas que se mostraram inconvenientes à efetividade daquilo a que se pretendia (p.ex. Redes Neurais, o Avião que "não bate as asas"), *Ant System* também evoluiu.

Fast Ant System é uma versão simples de *Ant System* que procura incorporar estratégias de intensificação e diversificação. É feita de modo a aumentar sistematicamente a atratividade para a melhor solução encontrada, mas limpando a memódia e dando menos peso para a melhor solução quando o processo parece estagnado. Assim, *FANT* agrega alguns procedimentos de outras heurísticas como Algoritmos Genéticos e Busca Tabu. Os processos dar formudas são inicializados randomicamente com probabilidade maior para o que a matriz M de memória aponta como melhor. R é o único parâmetro requerido por *FANT* para balancear o update da matriz M.

2.7.3 Busca Tabu

A Busca Tabu (*Tabu Search*) desenvolvida por Glover [13] [14] é um procedimento de busca por soluções que evita retornar a soluções já visitadas sem ter que utilizar de muita memória para tal. Tendo em vista que não seria factível guardar em memória todos os passos já dados (e verificar se o atual já foi visitado), a memória é implementada tornando tabu movimentos que levariam a busca de volta a locais já visitados.

A Busca Tabu também implementa um critério de aspiração como meio de decidir a direção que se deve seguir em caso de todos os movimentos serem tabu. A busca tabu passível de variações quanto ao modo de representar a memória ou procurar fzer escolhas por caminhos. Para o PQA, Taillard desenvolveu uma robusta versão de Busca Tabu, a qual usamos em nossos testes.

Tabela 2.3: Comparação dos papéis das metaheurísticas.

	GRASP	**GRASP-PR**	**FANT**	**Tabu**
Início independente:	sim	sim	não	não
Uso de memória:	baixo	médio	alto	alto
Soluções Estáveis:	sim	sim	sim	não

2.8 Bases de Raiz Mista, Autômatos e Permutações

Em oposição às bases numéricas de raiz única tais como binária, octal, decimal e hexadecimal, as bases de raiz mista (*mixed radix*) tem valores arbitrariamente definidos para cada posição um dos símbolo ocupa numa cadeia. Uma vez que alguns dos algoritmos descritos adiante dependem desse conceito, apresentaremos brevemente uma revisão comparativa de sistemas numéricos na forma de autômatos encadeados. Essa revisão é um prelúdio do mapeamento entre o sistema numérico fatorádico (*factorial number system* ou *factoradic*) e a sequência de permutações geradas pelo algoritmo Steinhaus-Johnson-Trotter (SJT).

Os diagramas na figura 2.3 podem ser formalizados como Autômatos Contadores, Máquinas de Contagem ou numa forma limitada de Autômato de Pilha, sendo rele-

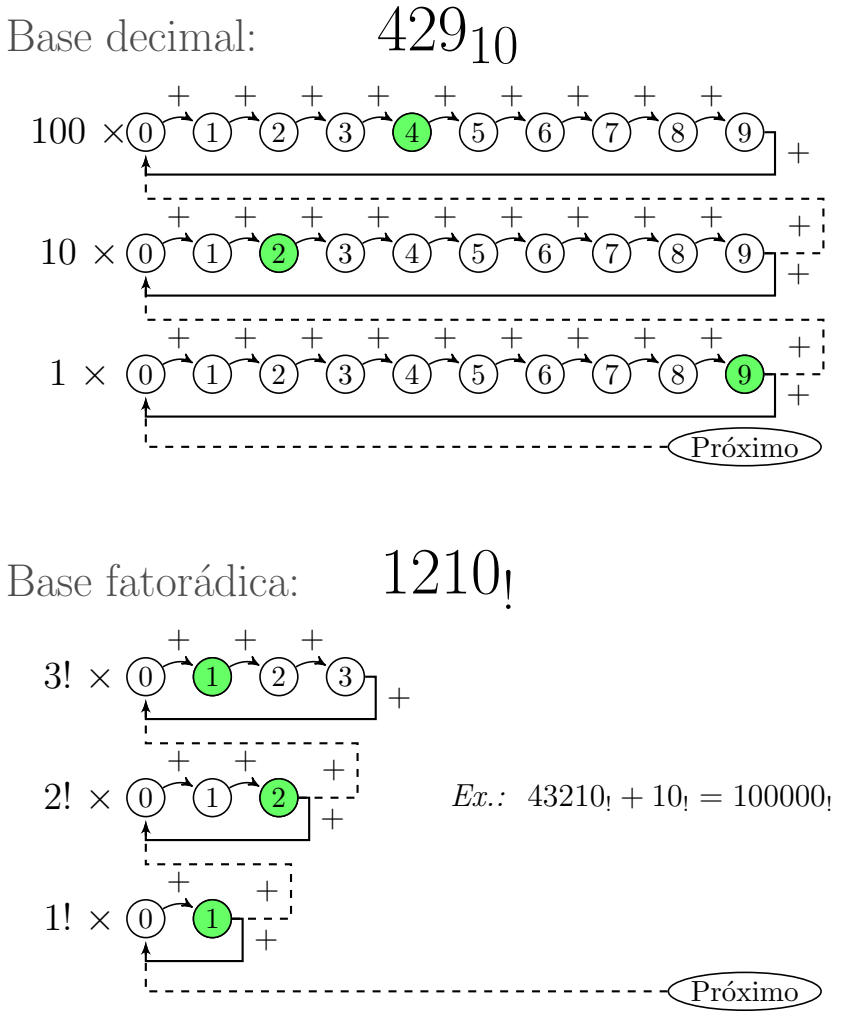


Figura 2.3: Comparação das bases decimal e fatorádica.

vante que o estado ativo seja preservado como estado inicial de uma nova operação representando um dos possíveis valores que um dígito de uma base numérica pode assumir.

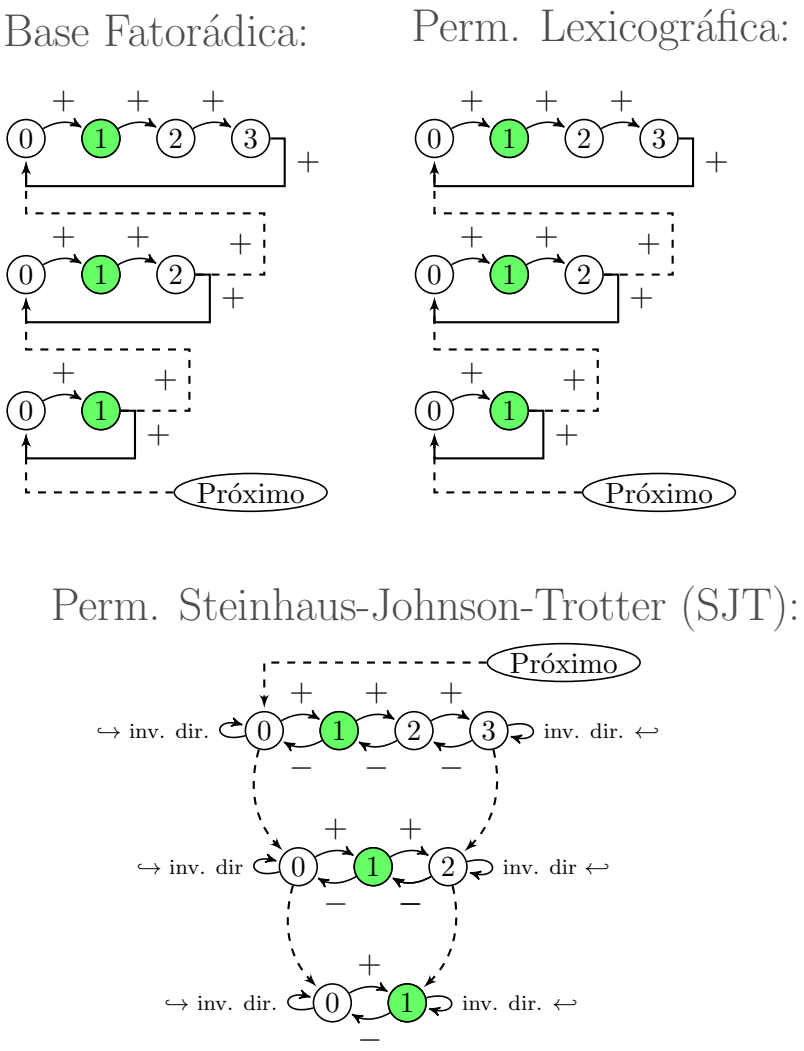


Figura 2.4: Base fatorádica e permutações.

Algoritmo Steinhaus-Johnson-Trotter

O algoritmo *Steinhaus-Johnson-Trotter* (SJT), também chamado *Plain Changes*, Knuth [15], gera uma sequência de permutações distintas entre si de n elementos e que contemplam todas a $n!$ possibilidades. Semelhantemente ao algoritmo *Heap*, a sequência é gerada de modo a alternar a posição de apenas dois elementos por vez, entretanto SJT apresenta a peculiaridade de que tal par de elementos serem sempre vizinhos no vetor permutação. Ou seja, o índice de elementos a serem trocados podem ser expressos no par ordenado $[i, i+1]$, (além disso, muitas das trocas ocorrem próximas umas às outras quanto à posição do par no vetor).

1	2	**3**
1	**3**	2
3	1	2
3	2	1
2	**3**	1
2	1	**3**

Figura 2.5: Sequência de permutação SJT de três elementos.

Tal característica pode ser explorada em heurísticas de otimização combinatória no caso de a posição relativa dos elementos a serem permutados guardar alguma propriedade gulosa, i.é, que aumente a probabilidade de encontrar melhores soluções em se trocando elementos vizinhos (em vez de distantes) de uma dada solução. Outra característica explorável para o mesmo fim consiste na maior frequência de trocas entre os elementos localizados no final (na direta) do vetor. Assim, se o problema for organizado de maneira a ordenar os índices de modo não-decrescente quanto à probabilidade de afetar a solução atual ao serem movidos, podemos explorar também esse fato de SJT em buscas locais.

Um modo simples de implementar SJT, sem que esteja necessariamente inserido num *loop*, utiliza dois vetores que assinalam as posições e direções dos móbiles sem que haja representação concreta da permutaçcão em si em memória. Tal permutaçcão é gerada por demanda a partir do vetor de posições.

Pseudo-código SJT: Inicialização

Algorithm 2.3 Pseudo-código SJT: Inicialização

```
1: função NovoSJT(n)          ▷ Arg.: n é o número de elementos da permutação
2:     pos ← [0, 0, 0 . . .]                                  ▷ Vetor com n zeros
3:     dir ← [0, 0, 0 . . .]                                  ▷ Vetor com n zeros
4:     para i = 1 . . . n faça
5:         pos[i] ← i
6:         dir[i] ← −1
7:     fim para
8:     retorna pos, dir
9: fim função
```

A função *Próxima* avança para a permutação seguinte sem efetivamente gerar um vetor de permutação.

Algorithm 2.4 Pseudo-código SJT: Próxima Permutação (um passo)o

```
1: função Próxima(pos,dir)          ▷ Args.: posições e direções dos móbiles
2:     para i = n . . . 1 faça                                  ▷ Ordem decrescente
3:         p ← pos[i]                                ▷ Posição do móbile p ∈ [1 . . . n]
4:         d ← dir[i]                                ▷ Direção do móbile d ∈ {−1, 1}
5:         se ¬((p = 1 ∧ d < 0) ∨ (p = i ∧ d > 0)) então  ▷ Móbile não-bloqueado?
6:             pos[i] = p + d                                ▷ Move na direção d
7:             Para repetição e sai da função!               ▷ Não continua iteração
8:         se-não
9:             dir[i] ← −d                  ▷ Inverte direção do móbile bloqueado
10:        fim se                                            ▷ Continua iteração
11:    fim para
12: fim função
```

A função *Perm* gera a permutação em si a partir do vetor de posições. As posições são relativas, elas se referem à posição que o índice ocuparia no vetor se não houvessem índices maiores que o do dado móbile.

Algorithm 2.5 Pseudo-código SJT: Gera vetor permutação

```
1: função Perm(pos)                  ▷ Arg.: vetor de posições dos móbiles
2:     res ← [1]                  ▷ Vetor já com o índice do primeiro móbile
3:     para i = 2 . . . n faça
4:         Insere i em res[pos[i]]   ▷ Copiando os valores já presentes em índices
   maiores que pos[i] para os índices seguintes
5:     fim para
6:     retorna res
7: fim função
```

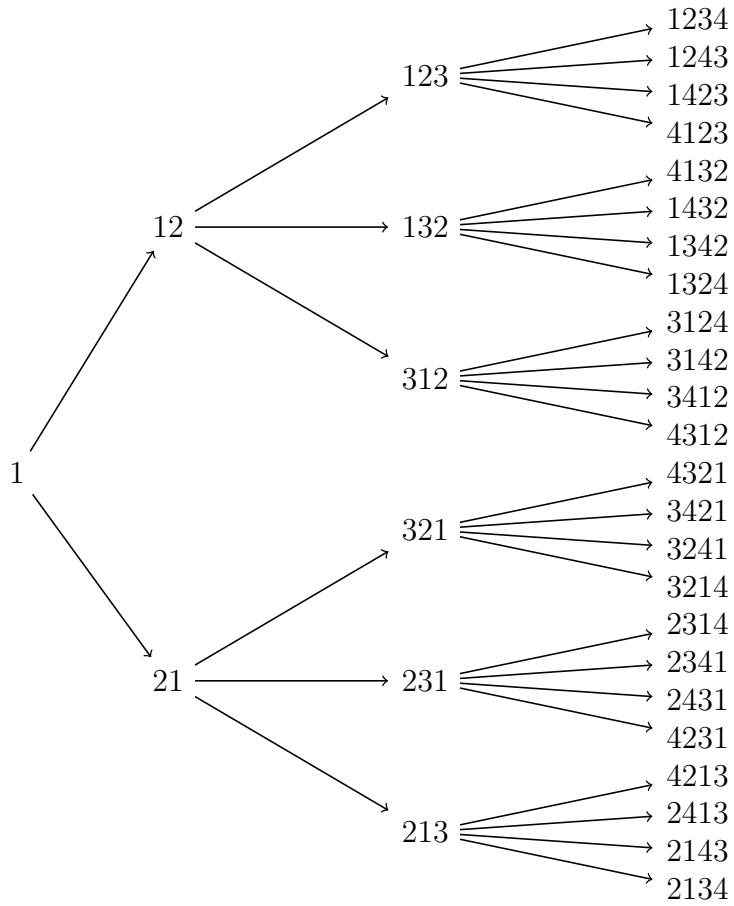


Figura 2.6: Árvore de geração STJ.

Capítulo 3

Método Proposto

> In fact the (analytical) engine may be described as being the material expression of any indefinite function of any degree of generality and complexity...
>
> Ada Lovelace. Notes (1842)

A maior parte das heurísticas para PQA possuem algum componente randômico[1]. Se por um lado ele confere a variabilidade necessária para que seja possível encontrar resultados diferentes – possivelmente melhores – a partir da repetição de um mesmo método, por outro as heurísticas se mostram especialmente difíceis de serem mensuradas quanto à qualidade de seu comportamento. De fato, até a verificação de que seu comportamento é precisamente o que dele se espera não é trivial, uma vez que se não pode tabelar os valores de saída para correspondentes valores de entrada.

A principal razão pela qual é difícil estabelecer um modo de comparar o quão melhor uma metaheurística é em relação a outra consiste na incompletude desse problema. Isto é, o valor de uma ferramenta computacional vai depender exatamente do tipo de uso que se pretende fazer dela. A fim de tornar o comportamento do software mais flexível e adaptável a uma maior variedade de problemas do mundo real, algumas heurísticas aceitam parâmetros de regulagem, deixando a cargo do usuário "completar" o programa definindo tais variáveis.

[1] Com exceção, por exemplo, da busca local por critério de melhor-melhoria.

Com o objetivo de proporcionar alguma exposição das qualidades e defeitos das metaheurísticas para PQA com respaldo na literatura do tema, empregamos os problemas da biblioteca QAPLib que são bem-estabelecidos como referência para tal. Não sendo possível prever se na demanda das diversas aplicações reais o critério de menor tempo ou maior qualidade de solução será o mais relevante, preferimos não arbitrar um modo artificial de relacioná-las. Assim, apenas apresentaremos os resultados dos testes levando em conta os dois critérios, considerando que até o presente momento não ocorreu de alguma das metaheurísticas vencer todas as demais por ambos os critérios. A "estabilidade" com que uma metaheurística melhora a qualidade das soluções encontradas em função do tempo, ou seja, se permite estimar com razoável probabilidade o tempo necessário para se atingir um determinado nível de qualidade, dadas as tentativas anteriores, também levada em consideração, mas, novamente, sem arbitrar o valor desse critério em relação aos demais.

3.1 Complementariedade de Metaheurísticas

A suposição confirmada por testes preliminares de que as metaheurísticas dificilmente substituem umas às outras, mas funcionam melhor de modo complementar – principalmente quando a relação entre critário de tempo e qualidade não está definida para o usuário –, orientou o método de implementação e teste das mesmas. Visando a utilização complementar, desenvolvemos um solver híbrido (uma hiperheurística) que resolve ao mesmo tempo em diferentes núcleos do processador o mesmo problema com as diversas metaheurísticas independentemente.

3.1.1 GRASP, FANT e Busca Tabu

As metaheurísticas GRASP (com e sem Path-Relinking), FANT e Busca Tabu foram testadas conforme a definição na literatura. GRASP em especial foi implementada com uma variação condicional da busca local e path-relinking hibridizadas com o fortalecimento do algoritmo SJT.

3.2 Proposta de Metaheurística: Íris

Venturosamente os problemas do mundo real não são tão "puros" quanto os que somos capazes de conceber inspirados por eles. Os problemas reais aparecem num contexto, numa história, num escopo restrito e por essa razão sempre podemos podar a busca bruta de soluções modelando restrições que limitem e orientem os algoritmos tornando-os mais inteligentes. A modelagem de um problema se dá pelo confronto intelectual com um corpo estranho. Seu estudo progride em experimentar e analisar tal corpo inferindo o funcionamento que lhe é próprio; a flexibilidade, inflexibilidade e articulação regular de suas partes que nos convida a formular um esquema que o represente suficientemente. Dado um modelo suficiente, supomos que resolvendo um problema conforme o modelo, também o fazemos para a realidade. Mas o fato desconcertante é que muitas vezes ocorre um imprevisto e acabamos por perceber que resolvemos muito mais problemas reais do que esperávamos a partir do estudo de um só problema abstrato porque de alguma forma cada corpo estranho aparece como um arranjo novo das mesmas articulações regulares como se tudo no universo funcionasse por variações combinatórias de aplicações regulares de um mesmo conjunto de regras.

O Problema Quadrático de Alocação pela formulação de Koopmans & Beckmann (1957)[1] é restrito ao produto interno de duas matrizes (sendo uma permutada em relação à outra), o que preserva uma estrutura algebricamente otimizável no problema. É possível, por exemplo, calcular a variação delta do valor de custo de uma solução "vizinha" em se permutando dois índices. O simples fato de haver um "atalho" que simplifique a busca pela baixa complexidade da modificação da solução torna o problema muito melhor.

Na formulação de Lawler (1963) [7] a matriz $C_{n^2 \times n^2}$ pode ser entendida como uma matriz $n \times n$ na qual as células são também matrizes $n \times n$. Sendo c_{ijkl} o endereço de uma célula, $c_{i\ldots}$ refere-se a uma linha de n matrizes $n \times n$, enquanto $c_{ij..}$ refere-se a uma matriz $n \times n$ e $c_{ij\phi(i)\phi(j)}$ refere-se à célula que contém o custo dessa combinação de índices, sendo i e j mapeados para $\phi(i)$ e $\phi(j)$, que corresponde a um dos n^2 valores que somados definem o custo da solução ϕ. Ainda nesse caso também é possível recalcular o custo da solução de modo simples (até mais do que na forma Koopmans-Beckmann) em se permutando um par de índices da solução.

Essa generalização entretanto desfaz um tanto da estruturação algébrica que havia quando o custo era o produto interno de matrizes justamente por permitir uma variedade maior, menos restrita, de combinações de valores que acumulam-se no custo. Essa formulação economiza cálculo, mas é dispendiosa quando ao uso da memória. Como nesse caso, muitas vezes podemos transladar a complexidade do tempo (de processamento) para o espaço (memória) mas não evitar que de um modo ou de outro ocorra a explosão de complexidade inerente ao problema.

Assim, nenhuma das duas formulações constinuem o problema de permutação mais bruto[2] possível, posto que o cálculo do delta – da variação de custo em função da troca de um par de índices – é menos complexo do que o procedimento do cálculo de custo de ambas as soluções e da diferença entre elas.

Um problema "puro" de permutação seria, portanto, aquele cuja função delta não pudesse simplificar o procedimento de busca, mas o custo ainda assim fosse calculável (limitando-nos ao escopo dos problemas completos de permutação, nos quais temos acesso a todos os dados necessários para cálculo de custo e cujo procedimento de cálculo seja tratável). Para um problema "puro", de pura força-bruta, impassível de qualquer orientação inteligente para o sentido da busca, o melhor algoritmo para resolvê-lo seria o de enumeração de permutações e verificação de tantas mais quanto possível no tempo disponível (sendo o algoritmo *Heap* um bom candidato). Entretanto, no escopo do presente trabalho nos limitamos a problemas de permutação com função delta simples, portanto limitado pela formulação de Lawler. Para esse caso (incluindo seus casos especiais) pudemos formular um arranjo de algoritmos combinatórios capaz de, pelo menos, orientar heurísticas de modo a evitar que fiquem restritas a uma determinada área homogênea do espaço de solução ou que circulem indefinidamente pelos mesmos caminhos e por uma dessas duas razões perca a oportunidade de encontrar novas soluções ao longo do tempo.

Particionar o espaço de solução de permutações de modo a evitar a revisita de soluções não é uma tarefa tão difícil: basta, por exemplo, apoiar-se na enumeração lexicográfica. Vasculhar a estrutura de vizinhança da troca de pares de modo eficiente com Heap ou SJT (explorando a eficiência da função delta) também não é um desafio e mostra-se interessante para intensificar a busca num subconjunto de

[2]Mais voltado para solução por força-bruta, inflexível à simplificação algébrica.

índices. Até esse ponto, entretanto, temos apenas uma orientação "macro" para diversificação da solução inicial e uma intensificação "micro" para fortalecer a busca local deixando um hiato no nível intermediário.

Nossa proposta com Íris é fornecer esse arcabouço macro-micro para exploração via hibridização heurística para quaisquer problemas de permutação (com delta simples) utilizando-se de características peculiares do algoritmo SJT, um mapeamento simples entre a sequência SJT e a base Fatorádica de modo que a iteração percorra frações bem-distribuídas e cada vez menores do espaço de solução, mas possa ainda assim iterar a partir de cada um desses pontos como se a sequencia de permutações geradas por SJT houvesse chegado ali do início e procedido normalmente. Esse arcabouço se enriquece das heurísticas bem-conhecidas e empregadas em diversas metaheurísticas como a Busca Local e Path-Reliking. Em especial nessa última creditamos mais tempo nos testes empíricos em razão de a iteração de Íris alternar entre permutações bastante diferentes entre si, o que aumentaria a probabilidade de se encontrar uma boa solução "percorrendo o caminho" entre elas e em seguida realizando a busca local. A fim de explorar o fato de os últimos índices da permutação se "movimentarem" mais que os primeiros pela iteração SJT, testamos também o artifício da permutação prévia de ambas as matrizes em ordem não-decrescente de coeficiente de variância do valor de seus arcos a fim de que os vértices de maior probabilidade de afetar o custo ao se movimentar sofram mais comparações na troca de pares que os demais.

Os resultados empíricos foram interessantes e podem indicar Íris como uma alternativa relevante para se explorar o PQA (e promover hibridizações com metaheurísticas existentes), mesmo em sua forma mais genérica (Lawler), não como se houvesse concebido um procedimento essencialmente novo, mas apenas por arranjar ferramentas plenamente disponíveis na literatura de um modo aparentemente produtivo para caso do PQA e outros problemas de permutação.

3.2.1 O Funcionamento de Íris

A metaheurística Íris se propõe a complementar o empreendimento heurístico de solução aproximada do PQA – e problemas semelhantes de permutação – oferecendo um arcabouço de diversificação e intensificação da busca permutatória baseada em duas estruturas de vizinhança que formam caminhos hamiltonianos independentes no espaço de solução de modo que os caminhos se encontrem uma e apenas uma vez em cada permutação (enquanto vértice do grafo) e jamais compartilhem um arco para qualquer par de vértices para permutações com mais de dois elementos.

Seja $G_{SJT} := (\Pi_n, \Delta)$ um grafo orientado (dígrafo) no qual Π_n são vértices que correspondem a cada uma das $n!$ permutações de n elementos conforme a sequência gerada pelo algoritmo SJT, e Δ o conjunto de arcos que representam as transições entre duas permutações de Π_n. Por definição, $\pi_0 \in \Pi_n$ é a permutação de n elementos em sua ordem trivial $\pi_0 = (0,\ 1,\ 2, ...,\ n-1)$. Definamos também $\delta_i \in \Delta$ como o arco que representa a função de transição de π_i para π_{i+1}, portanto se

$$\pi_0 = (0,\ 1,\ 2, ...,\ n-4,\ n-3,\ n-2,\ n-1)\ ,$$

então

$$\delta_0(\pi_0) = \pi_1 = (0,\ 1,\ 2, ...,\ n-4,\ n-3,\ n-1,\ n-2)\ .$$

Na forma geral,

$$\delta_i(\pi_i) = \pi_{i+1}\ ; \qquad \forall i, \quad 0 \leq i < n! - 2\ .$$

Em razão de havermos implementado SJT na forma de um circuito hamiltoniano, podemos definir uma transição especial

$$\delta_{n!-1}(\pi_{n!-1}) = \pi_0$$

sem perda de suas propriedades, uma vez que a última permutação na sequência SJT está a uma troca de pares de se tornar idência à primeira. Generalizando,

$$\delta_{i\ mod\ n!}(\pi_{i\ mod\ n!}) = \pi_{(i+1) mod\ n!}\ ; \qquad i, n \in \mathbb{Z}\ .$$

Desse modo, seria teoricamente simples – havendo algoritmos para tal – iniciar a sequência de busca por pontos bem-distribuídos no intervalo Π_n a fim de diversificar a solução inicial para a busca heurística de uma maneira mais controlada do que por permutações randômicas. Bastaria gerar a permutação a partir de seu índice e gerar índices bem-distribuídos. Suponhamos entretanto que desejemos gerar a permutação que se encontra no meio da sequência das permutações de 100 conforme a ordem SJT, o que seria a permutação $\pi_{100!/2}$. Ocorre, porém, de $100!/2$ ser um número grande. Suficientemente grande para desistirmos de representá-lo literalmente em base decimal ou outra base tradicional. Se poderia convertê-lo para uma forma fatorada em primos, mas ainda assim o tratamento aritmético de números nesse formato seria um tanto dispendioso considerando-se a simplicidade do que deva ser um índice.

Felizmente a base numérica fatorádica é capaz de representar todos os índices até $n!$ utilizando-se n "dígitos". Esse sistema numérico é trivialmente correlacionável com permutações em ordem lexicográfica, com a vantagem em relaçcão à permutação em si de ser passível de operações aritméticas[3], justamente o que faz dele mais interessante do que as sequências permutatórias em si. Então, mapeando-se a sequência SJT para Fatorádicos podemos finalmente representar um índice para a sequência utilizando-se muito menos memória e poderíamos iterar por frações de $|\Pi_n|$. Se o mapeamento da sequência SJT para Fatorádicos fosse feito de maneira direta (na qual os índices de um correspondessem ao do outro) já poderíamos realizar nosso intuito, porém, observando-se a árvore de geração de permutação SJT, podemos notar que os menores dígitos (os primeiros da sequência na permutação inicial) se movimentam menos dos que os maiores. Por exemplo, até a metade da sequência, o número 1 sempre aparece antes de 2 e após isso, sempre aparecem invertidos.

Explorando essa característica, o mapeamento SJT-Fatorádico poderia ser feito de modo a vincular o valor dos dígitos menos significativos do número fatorádico ao movimento dos números que menos se movimentam na sequência de permutações SJT. Com efeito, adicionando-se repetidamente uma unidade ao número fatorádico a próxima permutação na ordem SJT a ele vinculada se posicionará em local bem "distante" da permutação mapeada pelo índice anterior. Por exemplo, do mapea-

[3]Não que não se possa definir um modo de fazê-lo com as próprias permutações, mas é mais simples operar sobre fatorádicos.

mento fatorádico do índice zero para o índice um, teremos exatamente a permutação "reversa"[4]. A propriedade da mobilidade pendular dos elementos da permutação na sequência SJT é o que promove uma boa diversificação das permutações geradas por índices sequenciais na base fatorádica. Com isso temos a vantagem de podermos inicializar o número fatorádico por um decimal bastante pequeno e mesmo assim com poucas iterações visitar permutações bastante diversas entre si.

A Função Í

Seja ι (*iota*) um contador monótono, inicializado a partir de zero, de base irrelevante (desde que algoritmicamente mapeável para base fatorádica em correspondência idêntica de valor). A iteração $\acute{I}(\iota, n)$ produz um caminho hamiltoniano pelas permutações de n que inicia em π_0 e termina em $\pi_{n!-1}$ com o propósito inverso da sequência SJT: diversificar o mais possível a troca de posição de elementos da permutação de uma iteração para a oura, mas principalmente de visitar o espaço de enumeração de permutações SJT de modo "harmônico", bem-distribuído.

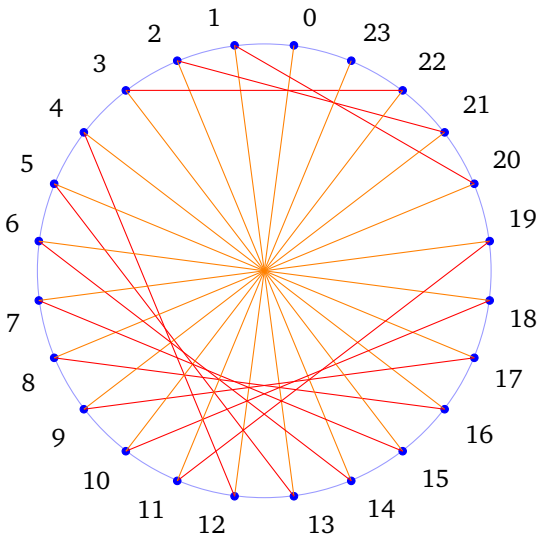


Figura 3.1: Ilustração do circuito e caminho hamiltonianos de SJT e $\acute{I}(\iota, 4)$

Dado o modo com que o mapeamento é feito, se pode garantir pelo menos que há uma alternância de sentido do salto em relação à sequência SJT a depender de ι

[4]Empregamos "reversa" (distintamente de "inversa") como sendo a permutação na forma $(n-1, n-2, ..., 2, 1)$.

ser par ou ímpar. Dado o arco

$$\pi_i = \acute{I}(k, n) \quad \rightarrow \quad \acute{I}(k+1, n) = \pi_j \quad ,$$

se pode afirmar que

$i < j$ sse. k é par ,

$i > j$ sse. k é ímpar .

Na figura 3.1 os arcos em azul indicam o circuito SJT (cujos índices aparecem escritos nos vértices), enquanto os demais se referem ao caminho gerado pela função $\acute{I}$. Os arcos em laranja sempre ligam um vértice de índice menor a um de índice maior (ι par) e os em vermelho fazem o oposto (ι ímpar).

Vejamos o funcionamento do mapeamento pela função $\acute{I}(i, n)$. A árvore de construção em cor preta ilustra como as permutações maiores são geradas a partir das menores na sequência SJT. Os números em cinza ligados às setas indicam a correlação dos dígitos de um número em base fatorádica com o caminho de construção da permutação da sequência SJT pela função $\acute{I}$ (os números mais à esquerda foram omitidos por clareza visual).

Se pode notar na figura 3.2 a correspondência biunívoca no mapeamento entre números fatorádicos e permutações SJT no intervalo $[0, n! - 1]$. Sendo que o dígito menos significativo de raíz não nula do número fatorádico é binário e define a mais radical divergência no caminho pela árvore de construção da permutaçcão SJT, seu efeito muda o sentido do salto a cada nova unidade acrescentada no número fatorádico (portanto o sentido de um arco de $\acute{I}$ – com relação à sequência SJT – é determinado pela paridade do índice desse arco na contagem do caminho hamiltoniano gerado por $\acute{I}$).

$$\acute{I}(3010_!, 4) = (3, 2, 1, 4)_{SJT}$$

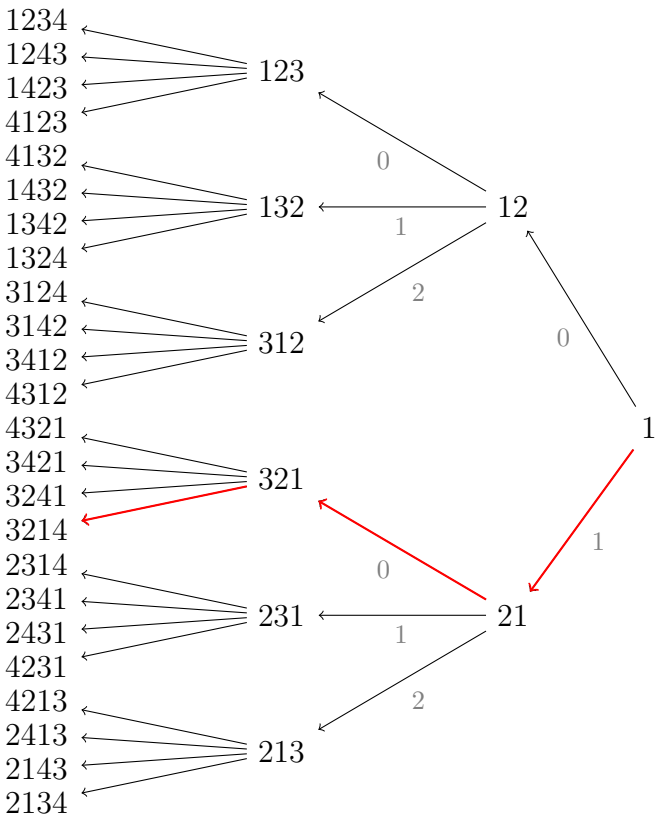


Figura 3.2: Mapeamento Í-SJT.

Algorithm 3.1 Geração de Permutação p por mapeamento Í

Função Í(fact, n)

```
1: função Í (fact, n)
2:     p = [] ← 0                    ▷ Inicialização de permutação com zero apenas
3:     dir ← −1                      ▷ direcão inicial SJT, direita para esquerda
4:     para v ← Proximo faça         ▷ "Dígito" fatorádico, de - para + significativo
5:         se dir = −1 então
6:             adiciona i + 1 em p, posição v da direita para a esquerda
7:         se-não
8:             adiciona i + 1 em p, posição v da esquerda para a direita
9:         fim se
10:        se v é ímpar então
11:            dir ← −dir                ▷ Inverte a direção
12:        fim se
13:    fim para
14:    retorna p
15: fim função
```

Uma das razões para a eficiência de Íris é sua simplicidade. É possível implementar a construção da permutação a partir de um índice em base fatorádica sob baixo custo computacional, um pouco mais do que no caso do mapeamento para permutação lexicográfica.

Algorithm 3.2 Geração de Permutação p por mapeamento Í em Go

```
1  // I gera a perm I(n) a partir do fatoradico atual
2  func (fact *Fact) I(n int64) (p []int64) { // metodo I do objeto Fact
3      var fact0 []int64
4      // (...) //  aloca fact em fact0 com tamanho n
5      p = make([]int64, 1, n) // inicia p = [0], apenas com zero
6      var dir int64 = -1 // direcao inicial SJT, dir. para esq.
7      for i, v := range fact0 { // itera do menos para o mais signif.
8          var k int64 // var k int
9          l := int64(i) + 1
10         // escolhe k de acordo com a direcao
11         if dir < 0 { k = l - v } else { k = v }
12         // insere valor i+1 em p na pos k
13         p = append(p[:k], append([]int64{l}, p[k:]...)...)
14         // inverte sentido se valor for impar
15         if v%2 != 0 { dir = -dir }
16     }
17     return
18 }
```

Algorithm 3.3 Algoritmo de Incremento (+1) de base Fatorádica em Go

```
1  // incrementa uma unidade no numero fatoradico
2  func (fact *Fact) Inc() { // metodo Inc do objeto Fact
3      for k, v := range fact.val { // itera do menos para o mais signif.
4          // Se o valor na posicao for maximo para a base,
5          // assinale zero e continue. Se nao, incremente e pare
6          if v == int64(k)+1 { fact.val[k] = 0 } else {
7              fact.val[k]++
8              return
9          }
10     }
11     // (...) // lida com sobrecarga, se houver (overflow)
12 }
```

É preciso esclarecer uma questão que pode induzir a erro a partir de um padrão perceptível na figura 3.2. Até a permutação de 4 elementos ocorre que a permutação "reversa" (não "permutação inversa") em relação à primeira (cujos últimos números aparecem nas primeiras posições e vice-versa) aparece na posição de índice $n!/2$. É possível provar que isso nunca acontece para $n > 4$. Em consequência, há duas maneiras diferentes de proceder com o mapeamento fatorádico-SJT que coincidem enquanto $n \leqslant 4$, ambas consistentes e que podem interessar a diferentes propósitos. O procedimento que definimos para a função $\acute{I}$ é tal que $\acute{I}(1, n)$ é sempre a permutação em ordem reversa de $\acute{I}(0, n)$ e, consequentemente para $n > 4$ não ocorre de $\acute{I}(1, n)$ encontrar-se na posição $n!/2$. Podemos definir $\check{I}(\iota, n)$ (Íris-regular) a função de mapeamento entre fatorádicos e SJT que procede por intervalos mais simples e regulares do intervalo $[0, n! - 1]$ de modo que, por exemplo, independentemente do valor de n, ocorrerá que $\check{I}(1, n)$ se encontrará em posição de índice $n!/2$ e ainda ocorrerá de que todo arco do caminho gerado por $\check{I}$ que for par na contagem a partir de zero transitará para o "local oposto" do espaço de solução em se representando como nas figuras apresentadas os vértices distribuídos em torno de um círculo representando todas as possíveis permutações de n ou seja, ocorrerá um salto a cada par de transições por $\check{I}$ de cardinalidade $n!/2$. Em nosso caso optamos por explorar no presente trabalho apenas $\acute{I}$ a fim de limitar pragmaticamente o escopo da pesquisa. De todo modo, apresentaremos a prova do conceito e indicaremos a diferença algorítmica simples do mapeamento de $\check{I}$ cuja diferença de $\acute{I}$ se baseia na prova a seguir.

Na sequência SJT, os móbiles apresentam comportamento pendular, mudando o

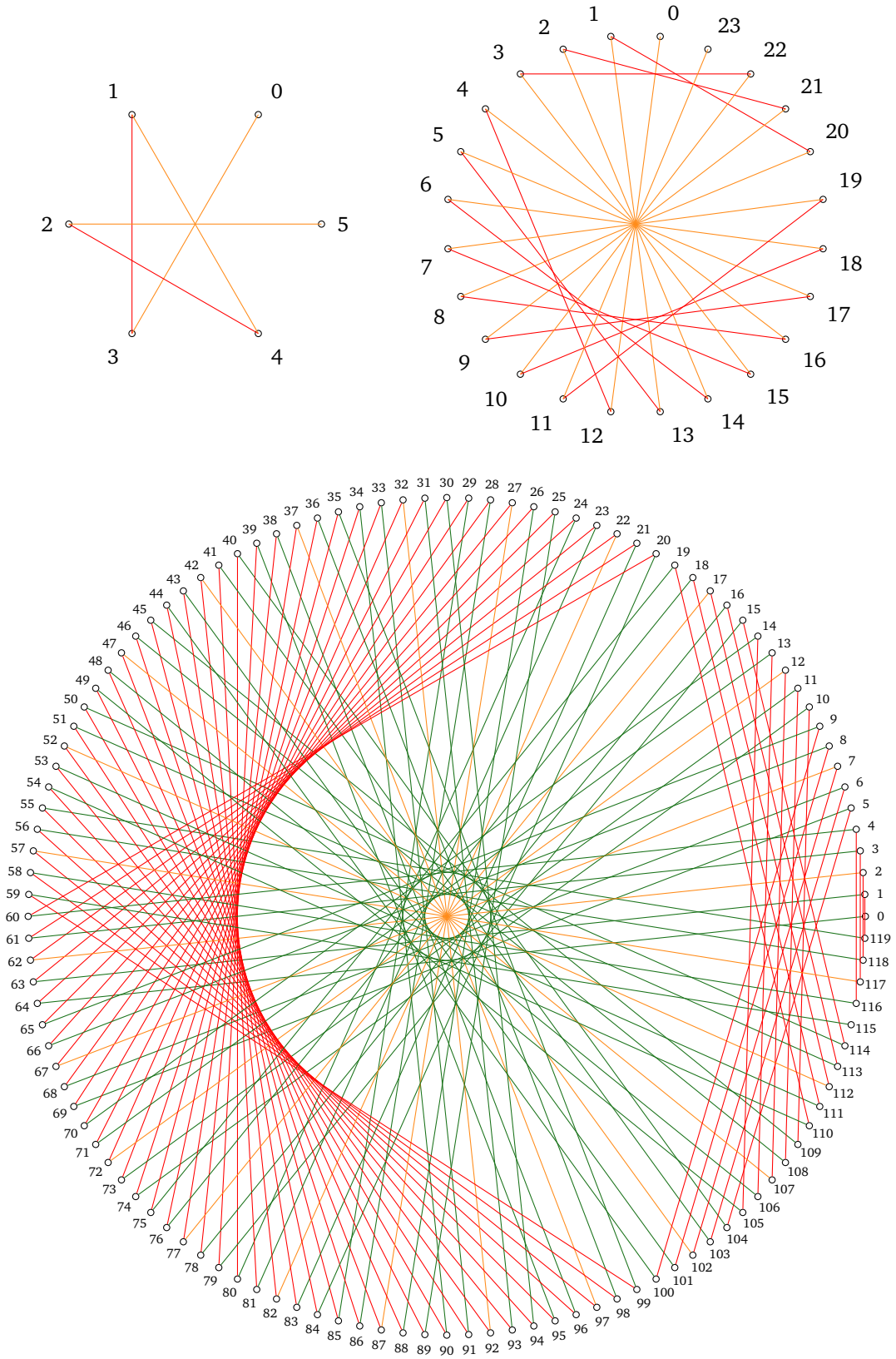


Figura 3.3: Ilustração da interação Í (para permutaçcão de 3, 4 e 5)

sentido de movimento a cada vez que atingem a posição máxima permitida naquele sentido (e nesse momento acionam um movimento da permutação subjacente $n-1$) e então invertem seu sentido permanecendo uma vez em mesma posição e em seguida se movimentam novamente no novo sentido até outra vez ficarem impedidos de fazê-lo. Assim sendo, ocorre que o elemento n movimenta-se $(n-1)!$ vezes atravessando todas as posições intermediárias do vetor de permutação subjacente. Desse modo, até que se chegue a $n!/2$ o móbile alterna entre as extemas posições do vetor $(n-1)!/2$ vezes. Evidentemente que iniciando-se na posição mais à direita, por definição SJT, o móbile se encontratá na extrema esquerda na permutação de índice $n!/2$ se, e somente se $(n-1)!/2$ for ímpar. Por exemplo, para o caso de 3, sendo $2!/2$ ímpar, o elemento 3 estará à esquerda em $3!/2$, o que de fato ocorre, correspondendo à permutaçcão $(3,2,1)$. No caso de 4, temos $3!/2$ ímpar, portanto o mesmo ocorre e a permutação na posição $4!/2$ é $(4,3,2,1)$. Entretanto no caso de 5, temos $4!/2$ par, e nesse caso o elemento 5 se terá mobilizado de volta para o lado que iniciou, portanto na posição $5!/2$ teremos a permutação $(4,3,2,1,5)$, e $(5,4,3,2,1)$ virá somente quatro transições adiante na sequência SJT. Conforme propriedade do número fatorial, não é possível que $i!$ seja divisível de forma inteira por k e $j!$ não o seja se $i<j$. Assim sendo, para $n \geq 5$ não pode ocorrer de $(n-1)!/2$ ser ímpar porquanto 4! divide-se inteiramente por 4 e assim como ele todos os demais $n!$ com $n \geq 4$ permanecerão pares após divididos por dois.

Há três motivos para que esse mapeamento "irregular" possa aumentar a probabilidade de maximizar a diferença entre permutações geradas em sequência: (1) os dígitos menos significativos do número fatorádico são correlacionados com as escolhas mais radicais na árvore de construção da permutação SJT, (2) enquanto o crescimento unitário em base fatorádica é serializado por ciclos sobre sequências aninhadas de símbolos, a sequência SJT apresenta um comportamento "pendular" (também aninhado) em vez de cíclico, pois ao alcançar o último símbolo (*overflow*) ocorre um retorno pelo sentido inverso em vez de retornar diretamente para o início da fila de símbolos e (3) por ser SJT uma sequência *"gray code"* para permutações, a proximidade de índices de sua sequência indica a semelhança de permutações, assim a distância entre seus índices – a depender do modo com que se varia o intervalo de coleta – pode ter o efeito contrário. Para uso em heurísticas de problemas como o

PQA, ambas as sequências são úteis como estruturas de vizinhança auxiliares para se buscar soluções semelhantes ou divergentes de um modo controlado que por ser baseado em enumeração pode, através de pares de índices fatorádicos, representar partições do espaço de solução do problema – seja para visitar ou evitar – de maneira econômica uma vez índices de enumerações implementam memória implícita.

$r \setminus n$	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
0	1	2	12	0	367	44	20764	3570	1884402	393797	250276538
1		3	8	62	8	2613	644	187418	52579	20650945	6296800
2			3	42	14	35	419	3468	22934	362102	2166603
3				15	240	37	12	2067	3974	156641	65842
4					90	1680	0	36	1710	39302	16
5						630	13440	0	0	18812	427680
6							5040	120960	0	0	225720
7								45360	1209600	0	0
8									453600	13305600	0
9										4989600	159667200
10											59875200
$\sum$	1	5	23	119	719	5039	40319	362879	3628799	39916799	479001599

Tabela 3.2: Frequência absoluta de repetição de posição de dígitos em transições da sequência de permutações geradas em ordem $\acute{I}$.

Em suma, o mapeamento da função $\acute{I}$ é reverso quanto ao valor significativo dos símbolos e cíclico-pendular (em múltiplas camadas de ciclo e de movimento pendular) por conta de SJT. Noutras palavras, o caminho é desenhado por uma máquina abstrata com articulações recursivas-cíclicas-pendulares de modo a percorrer uma e apenas uma vez todas as permutações iniciando-se na de ordem natural e terminando numa que pode ser convertida na primeira invertendo-se seus dois primeiros elementos.

A escolha da enumeração permutatória SJT em lugar das muitas outras existentes - ainda mais rápidas ou mais bem-distribuídas - se deve às suas características aparentemente mais exploráveis para o caso específico do PQA: a troca de pares sempre vizinhos e a maior atividade de uns móbiles do que outros. Considerando-se que

a troca ocorrerá sempre entre vizinhos no vetor solução e que os últimos elementos desse vetor se movimentam mais frequentemente, podemos reorganizar os dados do problema de modo a fazer com que o funcionamento normal da enumeração aumente a probabilidade de se encontrar boas soluções no início da enumeração. Se pudermos, portanto, ordenar os vértices dos dois grafos de modo não-decrescente de risco alteração significativa no valor do custo ao se movimentar, poderemos aumentar a probabilidade de se encontrar boas soluções com menos enumerações e em menos tempo, porquanto os vértices mais expressivos serão mais frequentemente trocados com os demais, principalmente com os que também apresentam maior risco.

A vantagem de se organizar um problema para um método de solução em vez do contrário – quando isso é possível – é que o método em si pode ser intensamente otimizado independentemente do conteúdo ou finalidade de aplicação e funcionar com uma eficiência difícil de ser alcançada por procedimentos genéricos de alto-nível que sobrecarregam o processamento com toda a "burocracia" necessária para manter a validade do conteúdo em meio a tanta flexibilidade e generalidade [5].

A metaheurística Íris se propõe a complementar a função das demais metaheurísticas no sentido de procurar garantir a diversidade de soluções iniciais e, por outro lado, oferecer um reforço via força-bruta para visitar todas as permutações de um subconjunto dos índices. Noutras palavras, utiliza-se de duas estruturas de vizinhança com funções opostas: (1) visitar rapidamente soluções semelhantes e (2) iterar por soluções bastante diferentes entre si (além da da própria busca local).

A figura 3.1 ilustra em azul a sequência de permutações SJT, semelhantes entre si, que formam um circuito hamiltoniano pelo permutoedro (*permutohedron* ou *permutahedron*). O caminho hamiltoniano (que não fecha um circuito) iniciado em 1 e terminado em 24 que alterna em cores alaranjado e vermelho, sempre pelo interior, se refere à sequencia que procura maximizar a diferença de uma permutação para a próxima.

[5] O que atualmente significa compor por demanda um método/subprograma para cada problema em tempo de execução, sem que o mesmo possa ser beneficiado pela melhor otimização que ocorre em tempo de compilação.

$r \setminus n$	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
0	1	0	0	0	0	0	0	0	0	0	0
1		5	0	0	0	0	0	0	0	0	0
2			23	0	0	0	0	0	0	0	0
3				119	0	0	0	0	0	0	0
4					719	0	0	0	0	0	0
5						5039	0	0	0	0	0
6							40319	0	0	0	0
7								362879	0	0	0
8									3628799	0	0
9										39916799	0
10											479001599
$\sum$	1	5	23	119	719	5039	40319	362879	3628799	39916799	479001599

Tabela 3.4: Frequência absoluta de repetição de posição de dígitos em transições da sequência de permutações geradas em ordem SJT.

$r \setminus n$	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
0	1	2	1	4	1	6	1	8	1	10	1
1		3	10	5	28	7	54	9	88	11	130
2			12	50	30	196	56	486	90	968	132
3				60	300	210	1568	504	4860	990	11616
4					360	2100	1680	14112	5040	53460	11880
5						2520	16800	15120	141120	55440	641520
6							20160	151200	151200	1552320	665280
7								181440	1512000	1663200	18627840
8									1814400	16632000	19958400
9										19958400	199584000
10											239500800
$\sum$	1	5	23	119	719	5039	40319	362879	3628799	39916799	479001599

Tabela 3.6: Frequência absoluta de repetição de posição de dígitos em transições da sequência de permutações geradas em ordem lexicográfica.

Capítulo 4

Resultados e Discussões

> Testing can only prove the presence of bugs, not their absence.
>
> E. Dijkstra

4.1 Metodologia para avaliação do Método

A biblioteca QAPLib foi utilizada para teste de performance das metaheurísticas implementadas em linguagem de programação Go. Na primeira bateria, as metaheurísticas GRASP, FANT e Busca Tabu foram testadas em condições aproximadamente equivalentes para um subconjunto expressivo de problemas do QAPLib. A fim de oferecer uma comparação razoável, cada problema foi resolvido diversas vezes por cada uma das metaheurísticas. A partir de testes preliminares foi estimado o número de repetições que cada metaheurística deveria realizar para tivessem oportunidade de encontrar soluções razoáveis e demorassem aproximadamente o mesmo tempo no total (sendo que o número de iterações internas de todas as metaheurísticas para cada tentativa do problema foi de exatamente 1000). Todo esse processo foi feito duas vezes, uma com os métodos sendo executados em paralelo e noutra sequencialmente a fim de comparar a interferência da concorrencia na performance individual.

De todas as tentativas feitas, a média entre o melhor resultado para cada heurística encontrado individualmente e em paralelo foi utilizada como valor da qualidade da metaheurística para cada problema. A média entre o tempo efetiva-

mente utilizado pela metaheurística desde sua reinicialização até que encontrasse o melhor resultado em paralelo e individualmente foi utilizado para mensurar o tempo que ela leva para encontrar uma soluçcão de tal qualidade.

Esse modo de avaliar a relação entre qualidade de uma solução e tempo de processamento necessário para encontrá-la não se presume sem defeito, mas nos foi interessante o bastante para fazer pequenos testes e realizar ajustes necessários até que se pôde realizar o teste quase completo do QAPLib com o mesmo método sem variações de parâmetros no decorrer.

As configurações do computador utilizado para testes incluindo algumas outras informações estão sumarizadas na tabela 4.1.

Por conta da urgência com que foram feitos os testes para a metaheurística Íris[1], os mesmos não puderam ser feitos em condições tão otimistas quanto as demais. Cada uma das três variações de métodos foi testada contra a bateria completa do QAPLib apenas uma vez, de modo que a solução e o tempo efetivo até encontrá-la numa única tentativa para cada problema estão registrados nas respectivas tabelas. Por outro lado, Íris dispôs de mais tempo de busca e em especial "Íris 3" conta com o funcionamento de 8 núcleos do processador trabalhando em paralelo. Assim sendo, salientamos que as comparações entre Íris e as demais metaheurísticas em gráficos devem ser relativizadas pois não foi testada em igualdade de condições, mas será comparada por razões didádicas com a presente ressalva.

Tabela 4.1: Configurações do computador utilizado para os testes de performance das metaheurísticas.

Computador de teste		
Modelo:	Avell	(laptop)
Processador:	Intel Core i7	HASWELL 4710MQ 2.5 GHZ
Núcleos:	8	
Mem. RAM:	16GB	
Sist. Oper.:	Linux	Ubuntu 15
Ling. de prog.:	Go	go1.4.2 linux/amd64

[1]Íris só foi completada e validada pouco antes da data de apresentação desse trabalho.

Limitantes ilustrativos

Para que se pudesse verificar nos gráficos a qualidade das soluções com alguma referência além do valor das próprias, as comparamos com limitantes inferiores e superiores de modo que o intervalo máximo de variação de soluções ajude a avaliar a dificuldade do problema e o progresso da solução nesse intervalo. Para tal, utilizamos de limitantes fracos, mas de fácil cálculo, suficientes apenas para emprego ilustrativo.

Os limitantes foram calculados utilizando-se a propriedade do produto escalar mínimo e máximo [16]. Assim, o limitante inferior foi calculado pela soma do produto escalar mínimo das diagonais principais das duas matrizes com o produto escalar mínimo dos valores que não pertencem às diagonas principais das duas matrizes. Para o limitante superior fêz-se o mesmo porém com produto escalar máximo.

A razão para a distinção com relação à diagonal consiste no fato de que as pemutações dos índices matriciais (aplicadas igualmente aos dois índices de uma matriz) jamais movimentarão um valor da diagonal para uma célula que não pertence à diagonal. Desse modo, o limitante encontrado se torna um pouco melhor. Por esse método, os limitantes de dos problemas QAPLib puderam ser calculados consumindo, ao todo, fração de segundo.

Metaheurística	N^o.
GRASP	2
Busca Tabu	50
FANT	10

Tabela 4.2: Número arbitrado de tentativas de cada metaheurística para cada problema de QAPLib por estimativa de equivalência de tempo total.

4.1.1 Comparação em problemas selecionados

Selecionamos alguns problemas que apresentaram divergências entre as metaheurísticas a fim de analisá-las quanto à aproximação sub-ótima.

Problema Chr22a

FANT e as variações de Íris apresentaram melhor desempenho.

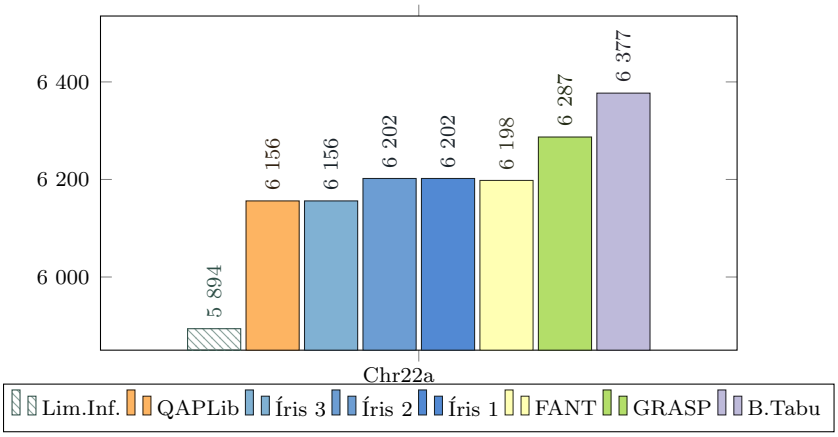


Figura 4.1: Comparação de metaheurísticas em Chr22a.

O mesmo gráfico com as proporções preservadas (sem corte mínimo abaixo) e incluindo o limitante superior indica uma razão para a dificuldade desse problema, uma vez que a relação entre a diferença de limitantes e o valor da média entre eles é grande em comparação com os problemas Bur.

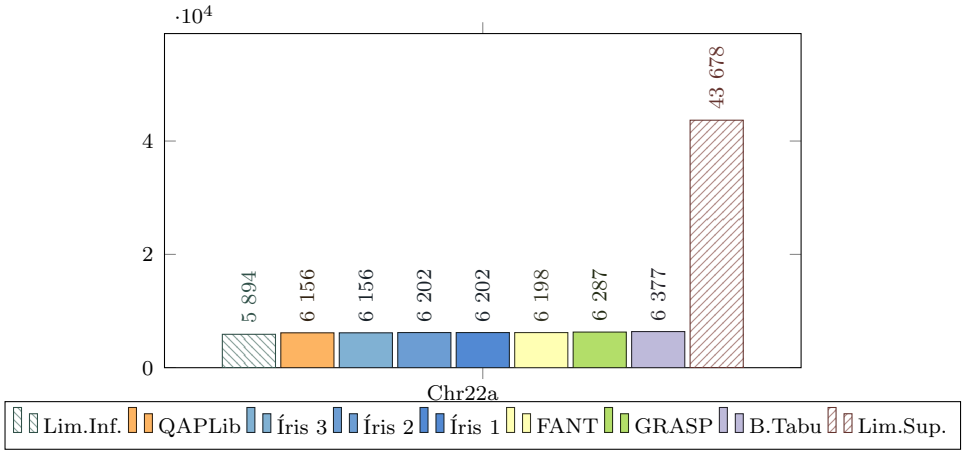


Figura 4.2: Comparação de metaheurísticas em Chr22a (com lim. sup.).

Problema Nug30

Não é tão difícil de se encontrar uma solução boa para o problema Nug30 como se pode notar pela pequena diferença no valor das soluções, porém essa pequena diferença é difícil de ser melhorada.

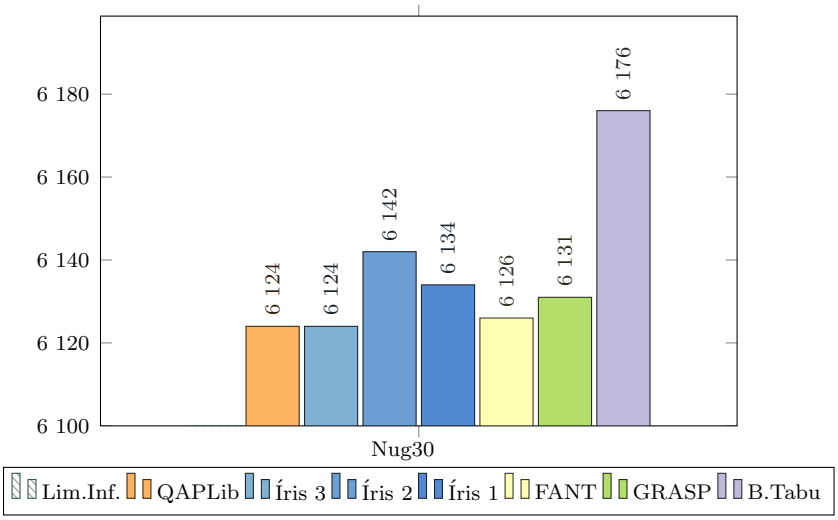


Figura 4.3: Comparação de metaheurísticas em Nug30.

Da perspectiva completa as diferenças ficam quase imperceptíveis. Somente[2] Íris 3 conseguiu alcançar o valor ótimo.

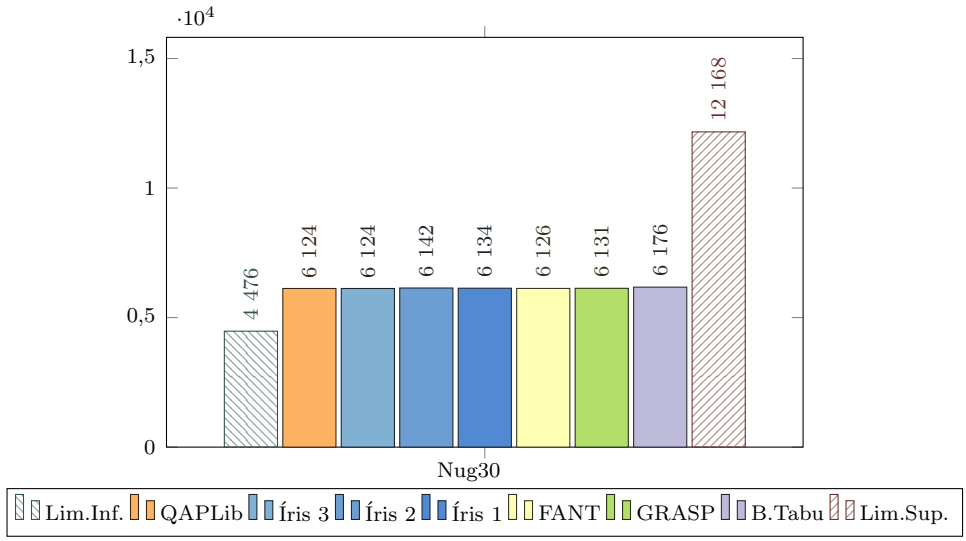


Figura 4.4: Comparação de metaheurísticas em Nug30 (com lim. sup.).

[2]FANT alcançou também o ótimo em um dos dois valores cuja média aparece no gráfico, o que significa no caso que o fez em pelo menos uma das duas tentativas que ocorreram sem interferência de processos em paralelo.

Problema Rou20

Semelhantemente a Nug30, Rou20 se torna difícil em se otimizar as últimas unidades de diferença para o ótimo.

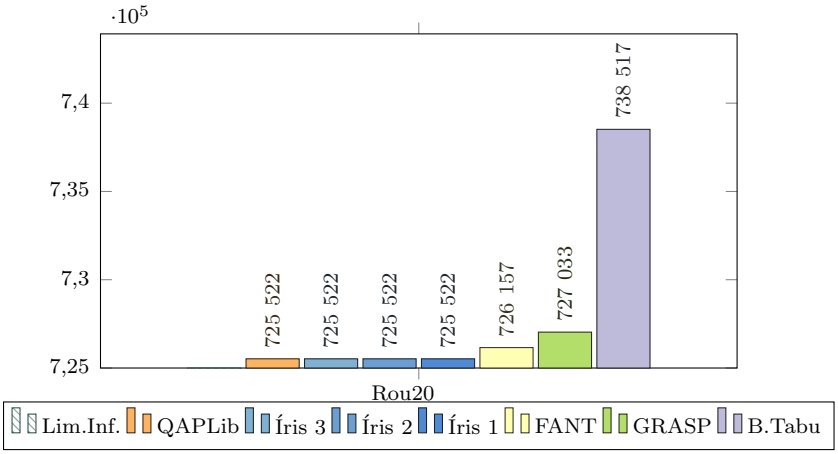


Figura 4.5: Comparação de metaheurísticas em Rou20.

Da perspectiva completa as diferenças ficam quase imperceptíveis. Mesmo as versões simples de Íris alcançaram o ótimo nesse problema, no qual curiosamente FANT[3] não alcançou em nenhuma tentativa.

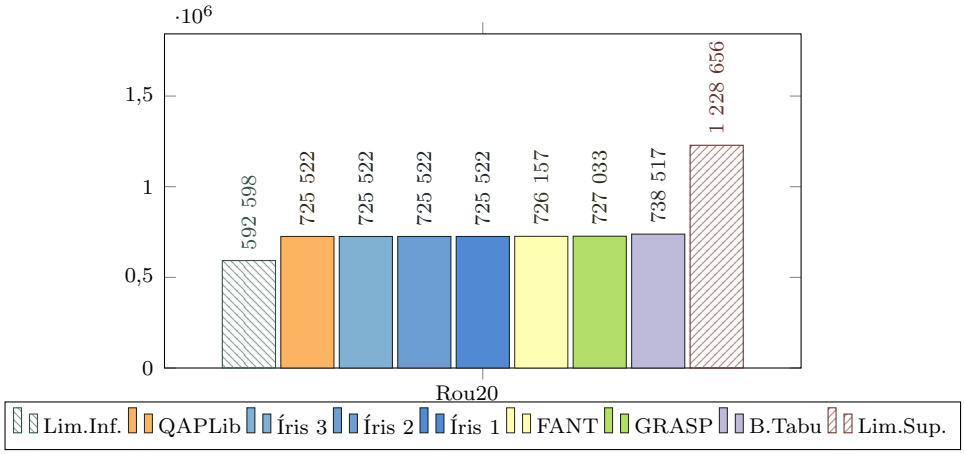


Figura 4.6: Comparação de metaheurísticas em Rou20 (com lim. sup.).

[3]GRASP alcançou também o ótimo em um dos dois valores cuja média aparece no gráfico, o que significa no caso que o fez em pelo menos uma das duas tentativas que ocorreram sem interferência de processos em paralelo.

4.1.2 Discussão de Resultados de GRASP, Busca Tabu e FANT

Os resultados discutidos aqui são baseados nos dados apresentados no apêndice, aos quais os gráficos fazem breve alusão.

GRASP

GRASP apresenta o melhor resultado em curto prazo e um bom resultado em médio prazo e grande estabilidade dos mesmos em diversas tentativas. Sua rapidez se deve à construção gulosa randômica que ao mesmo tempo preserva a variedade de soluções iniciais e evita o início por soluções muito ruins. Hibridizado com Path-Relinking se torna ainda mais rápida em apresentar boas soluções e melhora o resultado para instâncias esparsas.

Busca Tabu

Busca Tabu apresenta um comportamento errático. Iniciada por soluções randômicas não apresentou uma boa performance, mas há registro no QAPLib de excelentes soluções encontradas por ela. Provavelmente com mais tempo, ajuste de parâmetros ou soluções iniciais mais bem-distribuídas (por exemplo se hibridizada com Íris) apresentará, justamente pela erraticidade, soluções melhores do que as encontradas por outros métodos.

FANT

Dentre as três, FANT apresentou os resultados mais surpreendentes a médio e longo prazo para os problemas testados. Ela pôde, com razoável estabilidade nas diversas tentativas, convergir para boas soluções. Superou com facilitade na maioria dos testes GRASP e Busca Tabu.

O problema em comum

Apesar de eficientes no que se propõe, GRASP e FANT apresentam um problema em comum: elas "viciam", no sentido de, justamente ao procurar aumentar a probabilidade de encontrar boas soluções o mais rapidamente possível, vão se tornando

cada vez menos capazes de encontrar novas soluções que por alguma razão se esquivem da inteligência de sua busca. GRASP, pelo critério de corte da construção gulosa acaba sendo mais restritiva, enquanto FANT pode ser utilizada de maneira mais eclética e melhorada a partir de soluções iniciais mais bem-distribuídas do que simplesmente randômicas[4]. O mesmo acaba ocorrendo de outra forma com Busca Tabu, pois a finitude da memória, mesmo que inteligentemente administrada, torna impraticável que não se acabe retornando aos mesmo circuitos já visitados.

Esse problema em comum de tornar mais difícil encontrar o "exótico" ao privilegiar o "familiar" (ou de recair no "familiar" ao procurar o "exótico") nos pareceu insolúvel por boa parte da pesquisa porque sugeria uma dicotomia intranspassável (e muito pouco balanceável) da escolha entre "bom" e "rápido". Entetanto, Íris demonstrou empiricamente habilidade em pelo menos balancear essa dicotomia, já que transpassá-la implicaria na derrubada da classe de complexidade computacional a que pertence o problema.

4.1.3 Comparando Íris e FANT

Por haver apresentado excelente resultado comparativo em testes anteriores, escolhemos FANT como referência para o desenvolvimento de Íris. Não se tratando de te procurar estabelecer uma substituição, mas sim um desenvolvimento visando explorar o que parece o ponto de maior dificuldade das demais, Íris foi concebida para preservar a rapidez heurística de encontrar boas soluções, sem porém correr o risco de prender-se indefinidamente num local do espaço de solução ou em circuitos repetitivos. Para demonstrar empiricamente que Íris obteve êxito quanto a esse quesito, analisaremos testes com enfoque não no tempo de execução, mas na capacidade de não "viciar a busca" em tempo de execução.

Seguem os testes comparativos com número arbitrário de execuções, procurando entretanto, manter as duas metaheurísticas com tempo razoavelmente similar de busca. Em cada página, apresentamos na esquerda a plotagem de custo por tempo independentemente de qual das tentativas gerou tais ponto e na direita o caminho percorrido por cada uma das 10 tentativas para cada heurística no dado problema.

[4]Contraintuitivamente, se são randômicas, então não são necessariamente (nem provavelmente) bem-distribuídas. Nos parece inconveniente o frequente randomismo em procedimentos dos quais se espera boa diversidade, isto é, uma regular variedade.

A plotagem se dá relacionando custo (eixo Y, uniforme) e tempo (eixo X, em log) para analizar como elas se comportam após longos tempos de busca.

Taillard[12] implementou FANT em linguagem C. Em maior parte de nossos testes, utilizamos Go com as funções $Impr()$ (equivalente a $-Delta()$) e $Cost()$ otimizadas manualmente em assembly por se tratarem dos gargalos que mais retardavam Íris como um todo. A fim de conferir alguma similaridade de condições de teste com a versão original de FANT, implementamos Íris em C também para realizar o teste representado em 4.1.3. Exceto quando assinalado o contrário, todos os testes foram feitos em linguagem Go, convertendo-se da linguagem original (ou desenvolvendo-as a partir da definição nos respectivos artigos). Desenvolvemos Íris desde sua concepção em Go, sendo traduzida para C apenas para testagem comparativa. Utilizando-se a versão otimizada em Go, a performance comparativa é bem mais evidente. A versão de Íris em Go e assembly ficou pouco menos eficiente que a versão em C, quando otimizada com parâmetro -O3 e razoavelmente mais rápida que C quando usada a compilação sem otimização do GCC.

É possível perceber em 4.1.3 uma clara bifurcação na busca de FANT, indicando que uma parte das buscas ficaram presas a ótimos locais de baixa qualidade por um longo tempo, enquanto Íris continua a convergir para a mesma qualidade de boas soluções quando é executado tempo suficiente. Assim, tanto em Go quanto em C Íris se mostrou comparativamente interessante quando se trata de aplicar um pesado processamento em busca de uma solução de PQA, esperando que em longo prazo se garanta a possibilidade de melhora da solução garantindo-se que a busca jamais ficará restrita a um local já explorado do espaço de solução do problema.

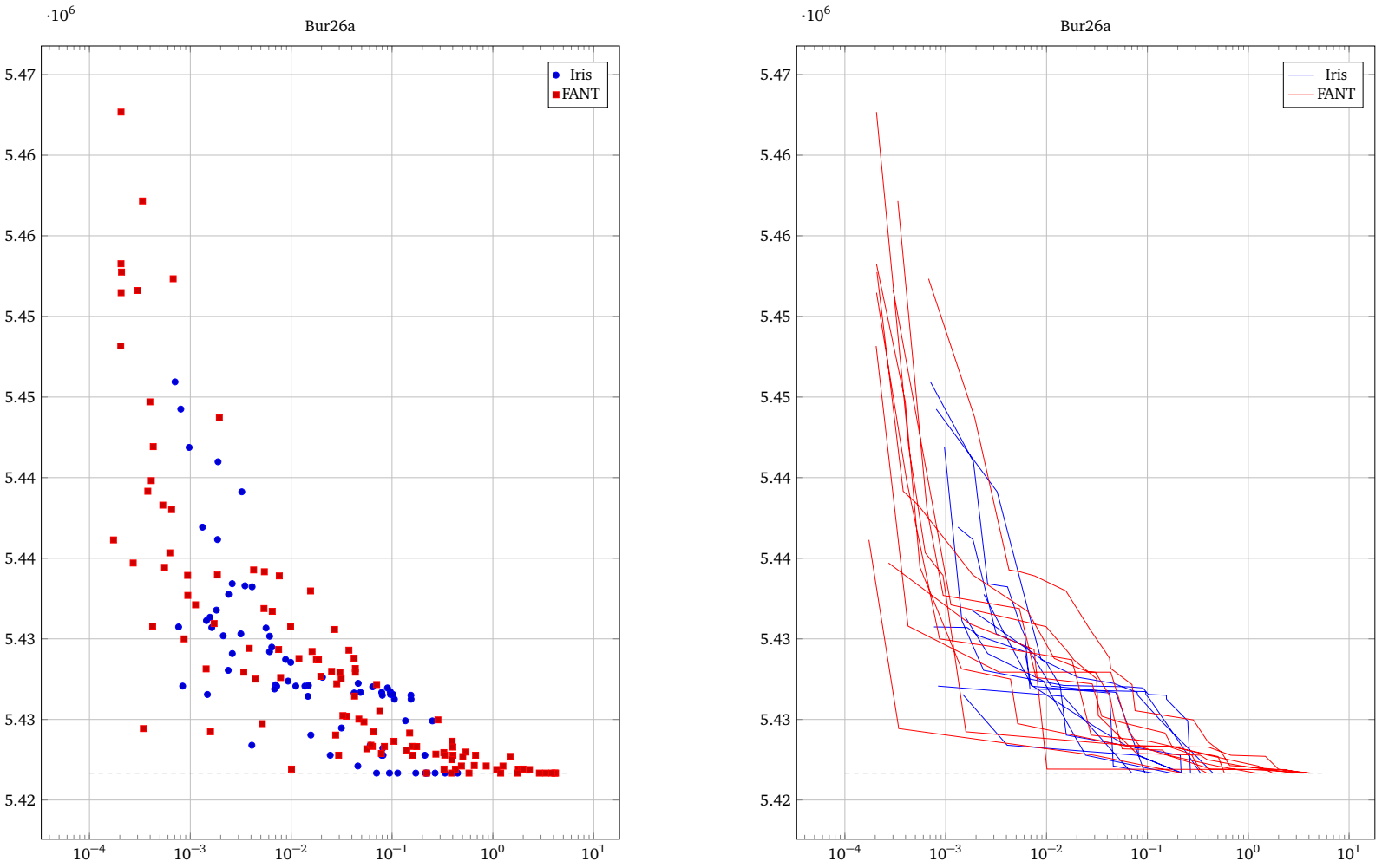


Figura 4.7: Bur26a Íris e FANT: performance / tempo

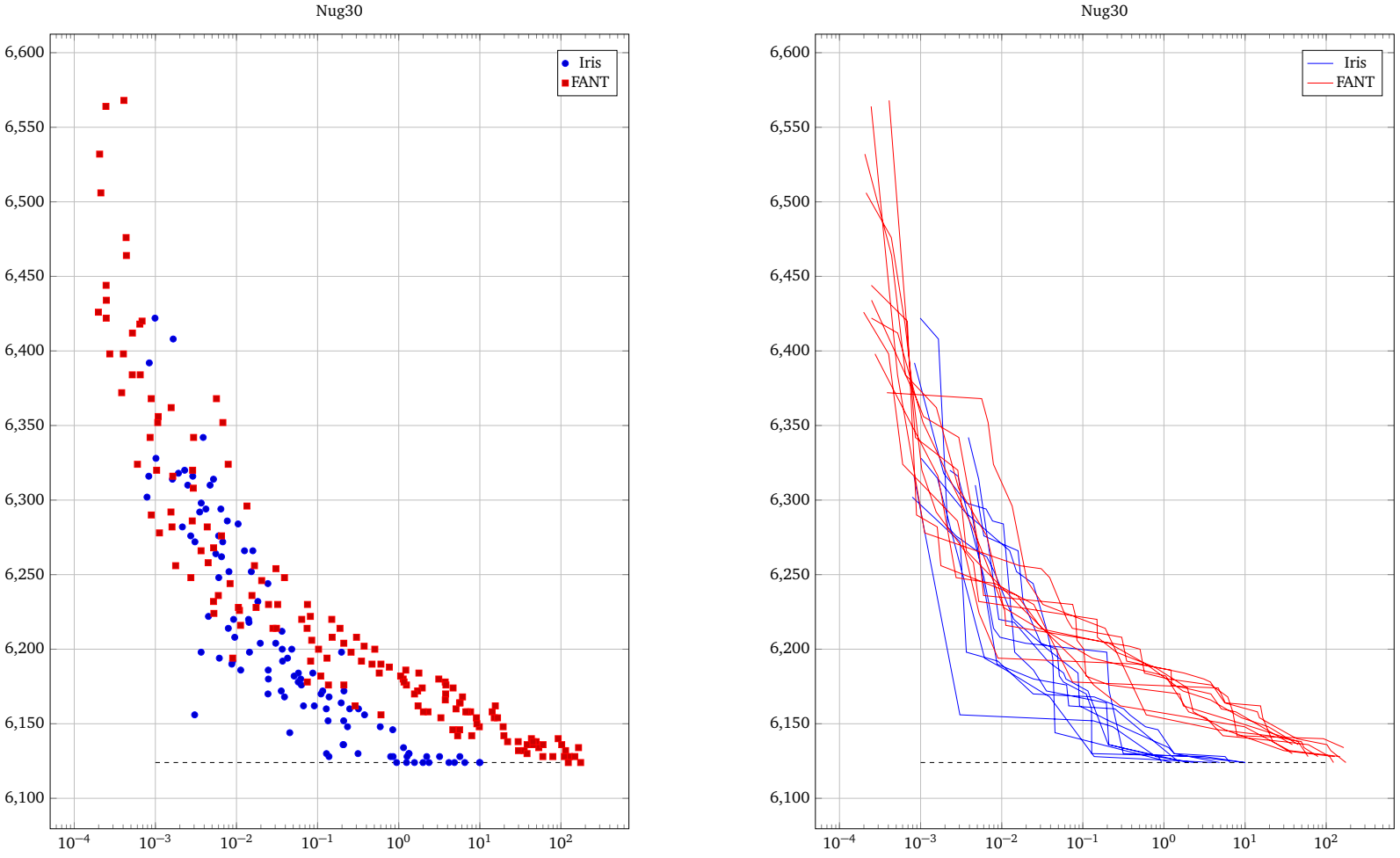


Figura 4.8: Nug30 Íris e FANT: performance / tempo

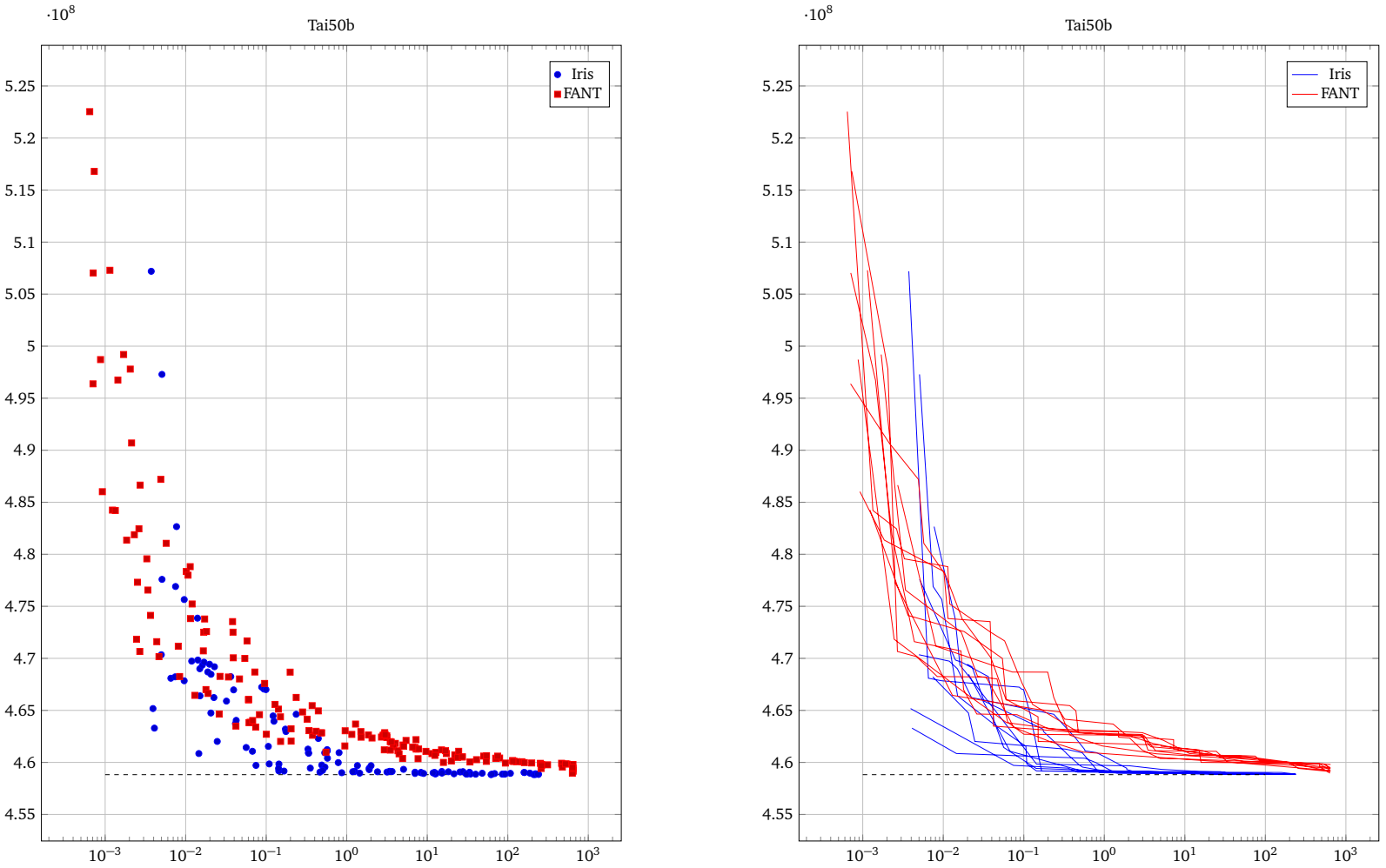


Figura 4.9: Tai50b Íris e FANT: performance / tempo

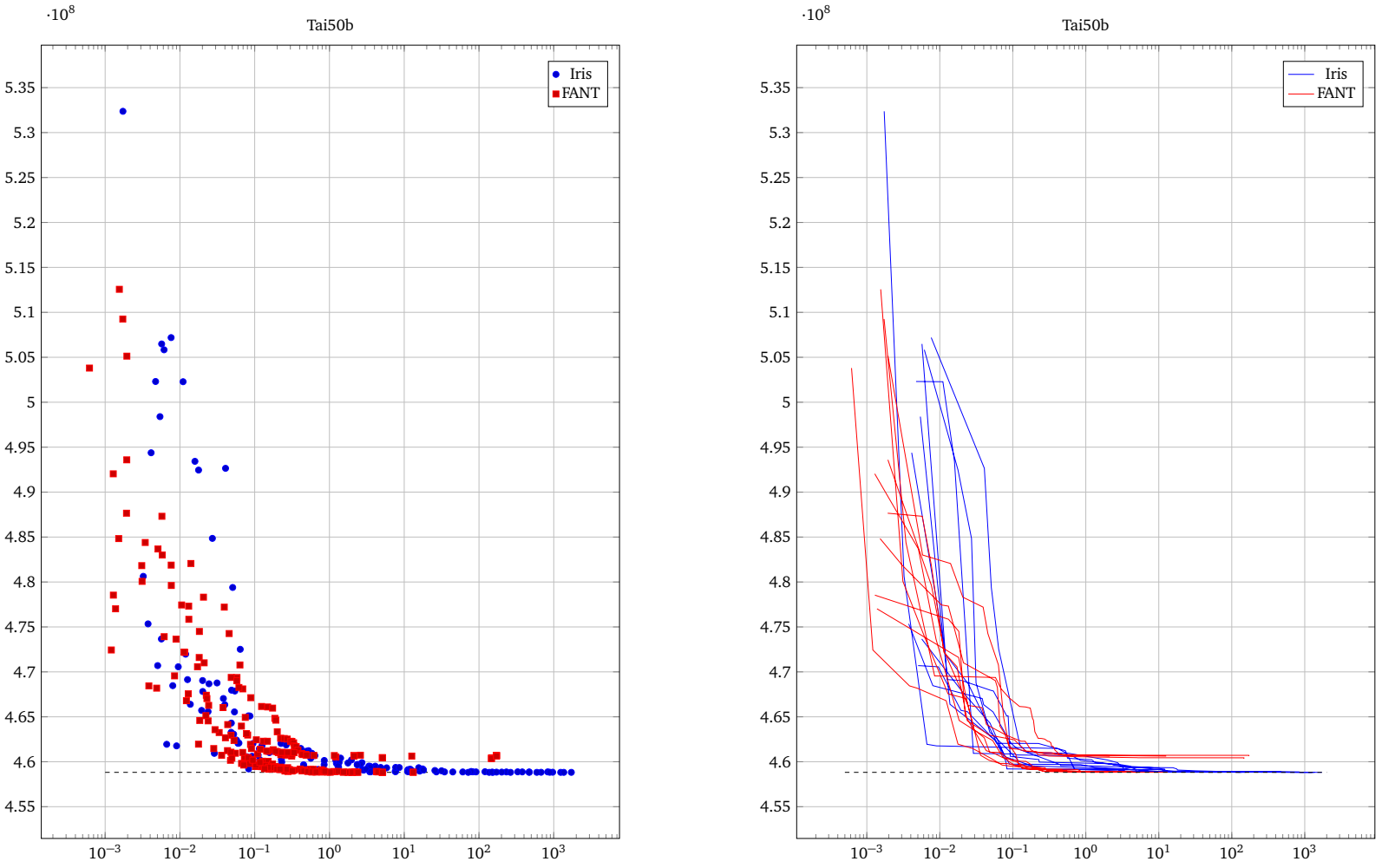


Figura 4.10: Tai50b Íris e FANT: performance / tempo em C

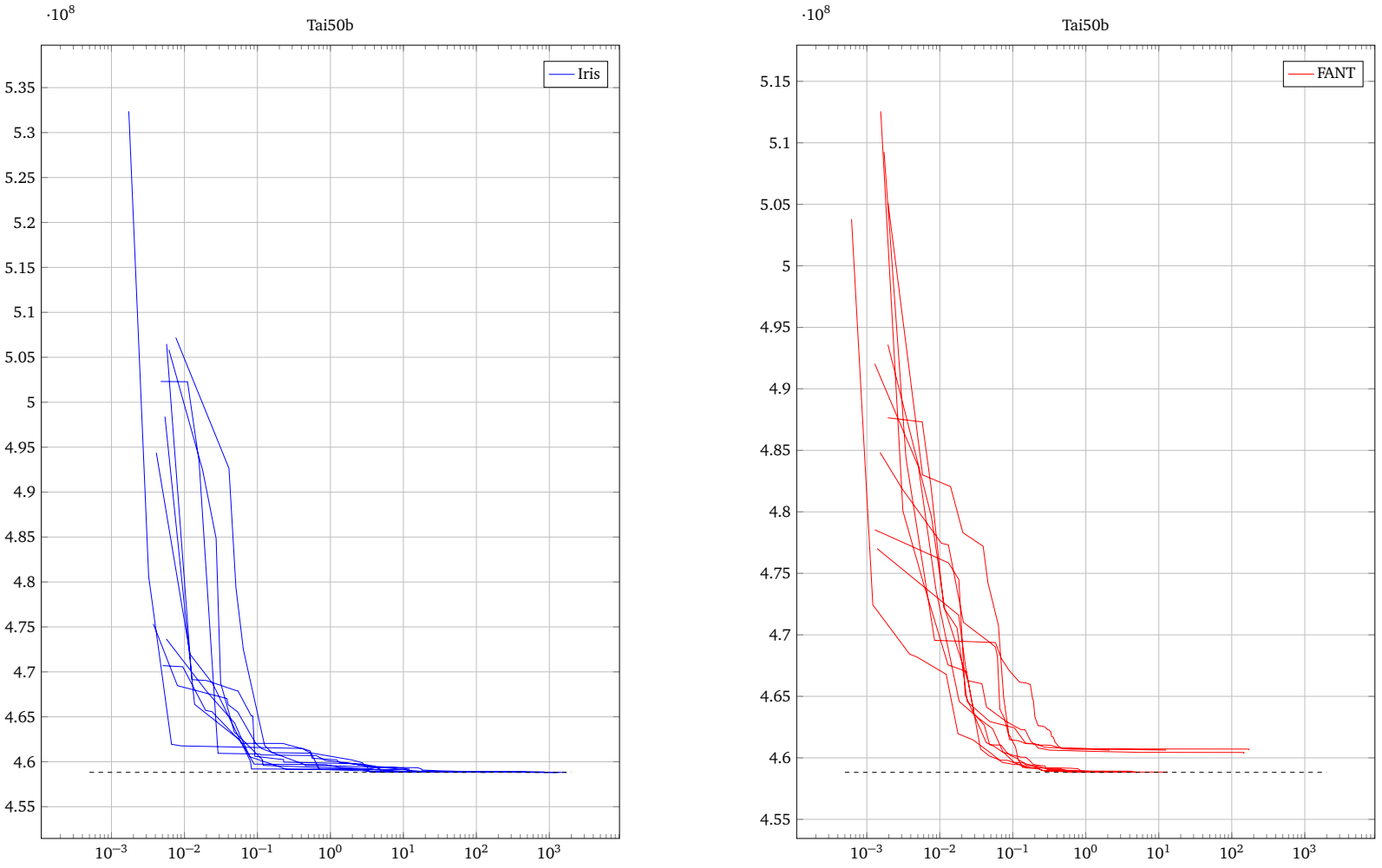


Figura 4.11: Tai50b Íris e FANT: em C (FANT preso em ótimo local)

4.1.4 Testes completos de Íris em QAPLib.

Enfocaremos a partir desse ponto a performance da melhor versão de Íris nos testes de QAPLib, comparando-se com as melhores soluções conhecidas e limitantes[5].

Bur26a-Bur26h

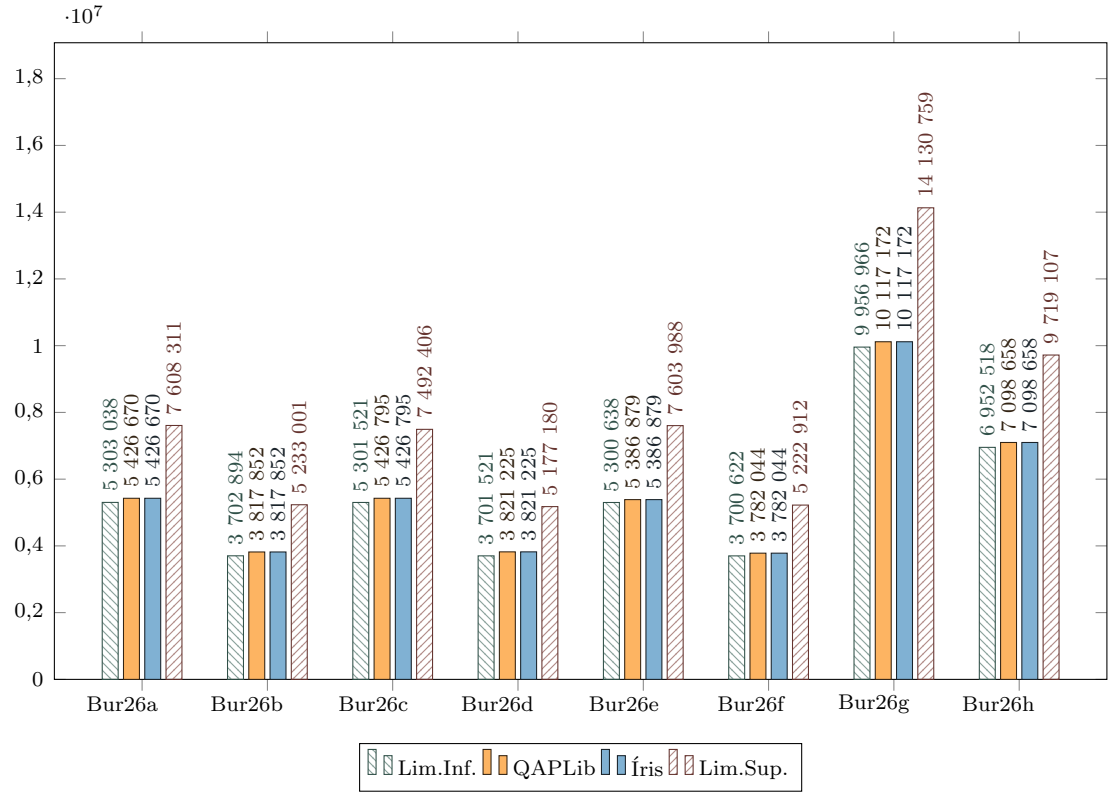


Figura 4.12: Gráficos Bur26a-Bur26h.

[5]Limitantes fracos obtidos pelo método anteriormente descrito que serão aqui representados para que se possa estimar o quanto a solução foi melhorada em relação a um intervalo mais factível de variação.

Chr12a-Chr18b (sem lim. sup.)

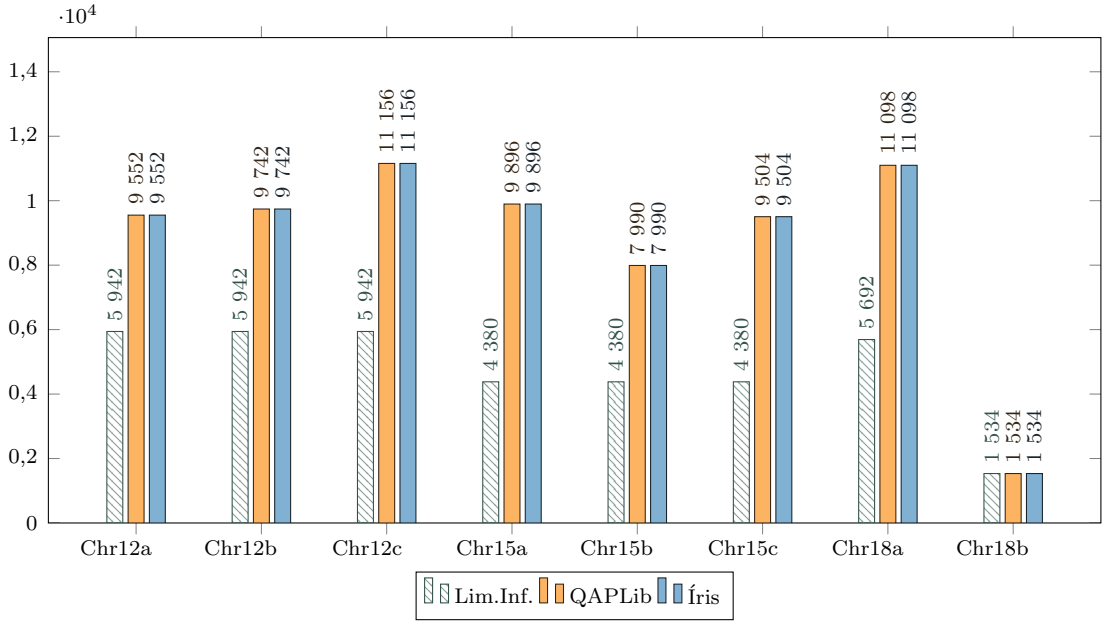


Figura 4.13: Gráficos Chr12a-Chr18b (sem lim. sup.).

Chr12a-Chr18b

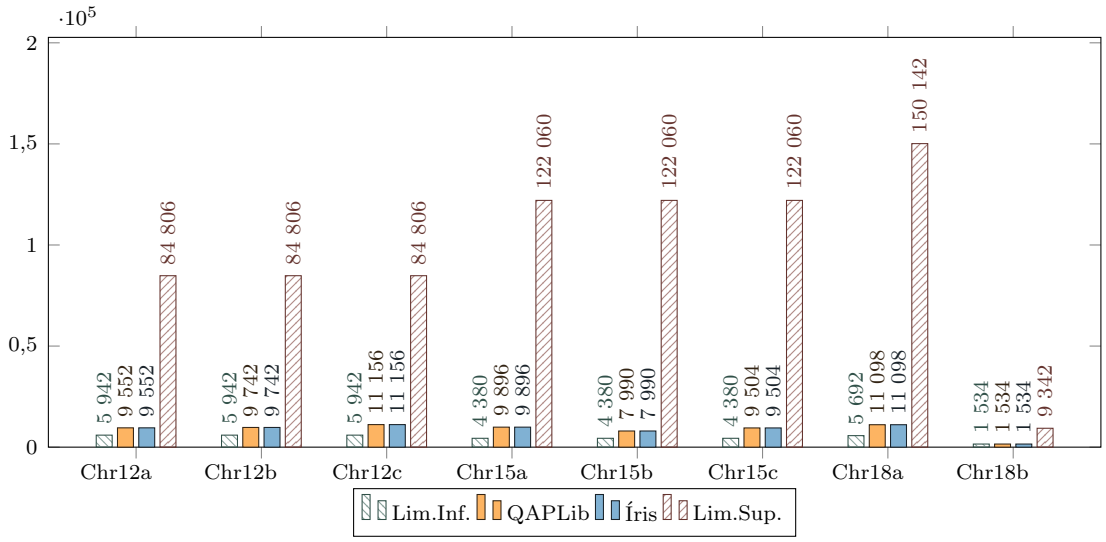


Figura 4.14: Gráficos Chr12a-Chr18b.

Chr20a-Els19 (sem lim. sup.)

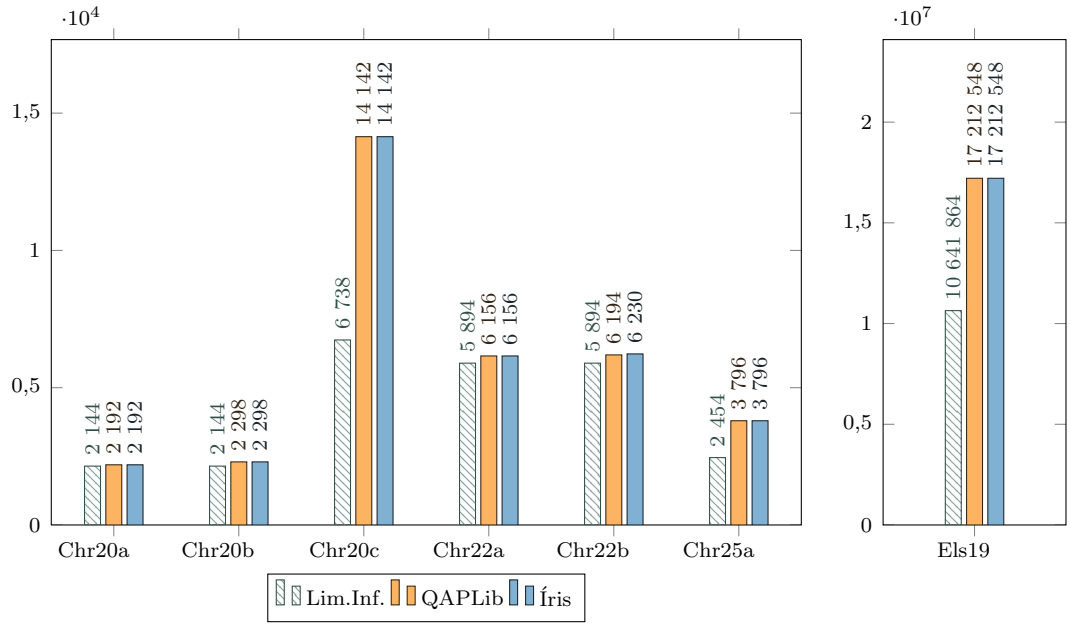


Figura 4.15: Gráficos Chr20a-Els19 (sem lim. sup.).

Chr20a-Els19

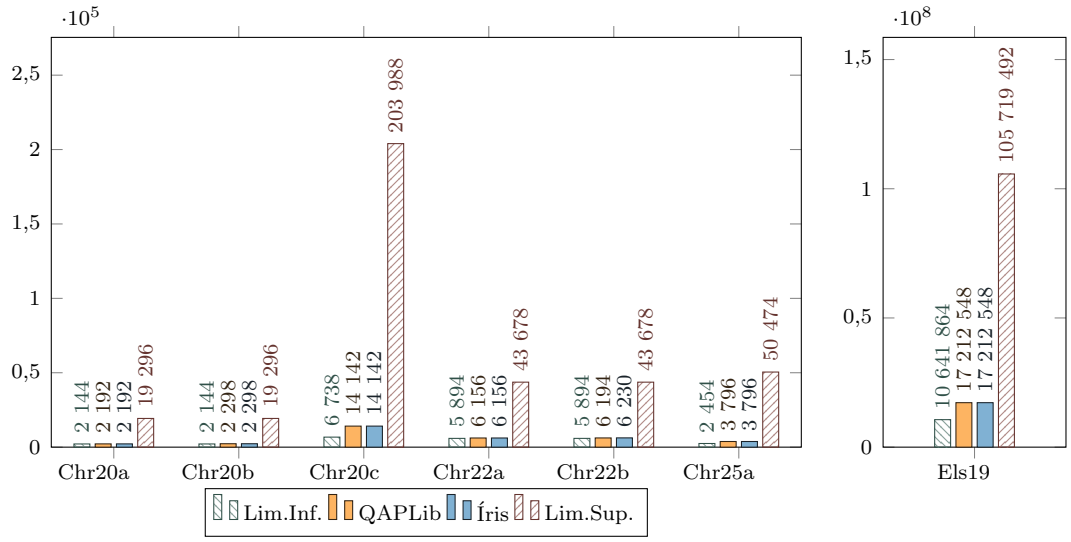


Figura 4.16: Gráficos Chr20a-Els19.

Esc16a-Esc16j (sem lim. sup.)

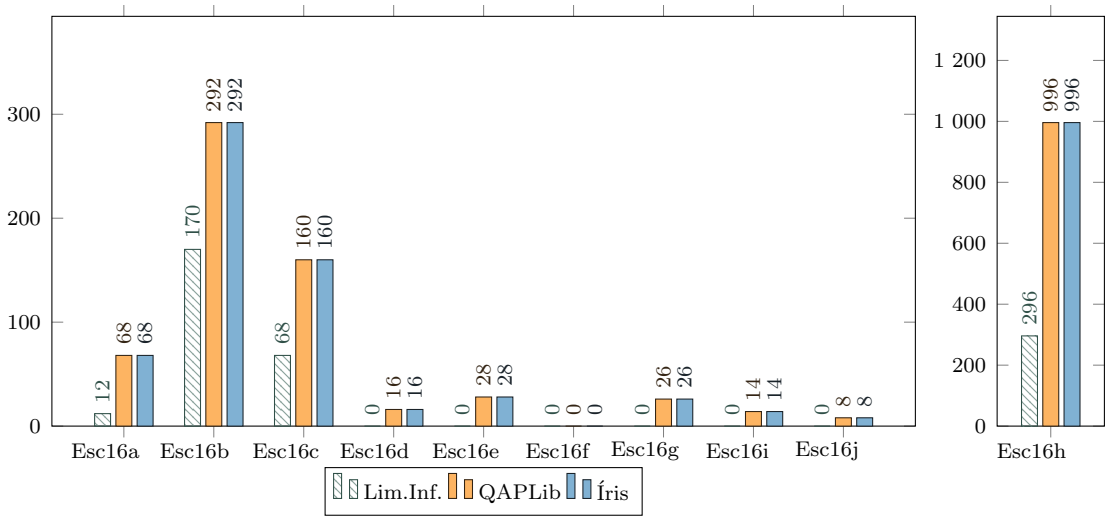


Figura 4.17: Gráficos Esc16a-Esc16j (sem lim. sup.).

Esc16a-Esc16j

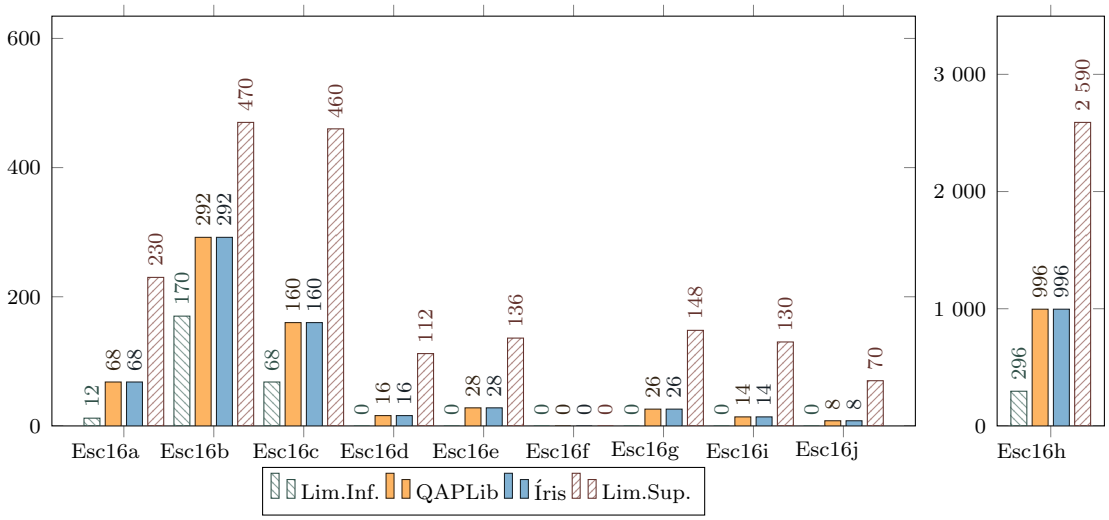


Figura 4.18: Gráficos Esc16a-Esc16j.

Esc32a-Esc128

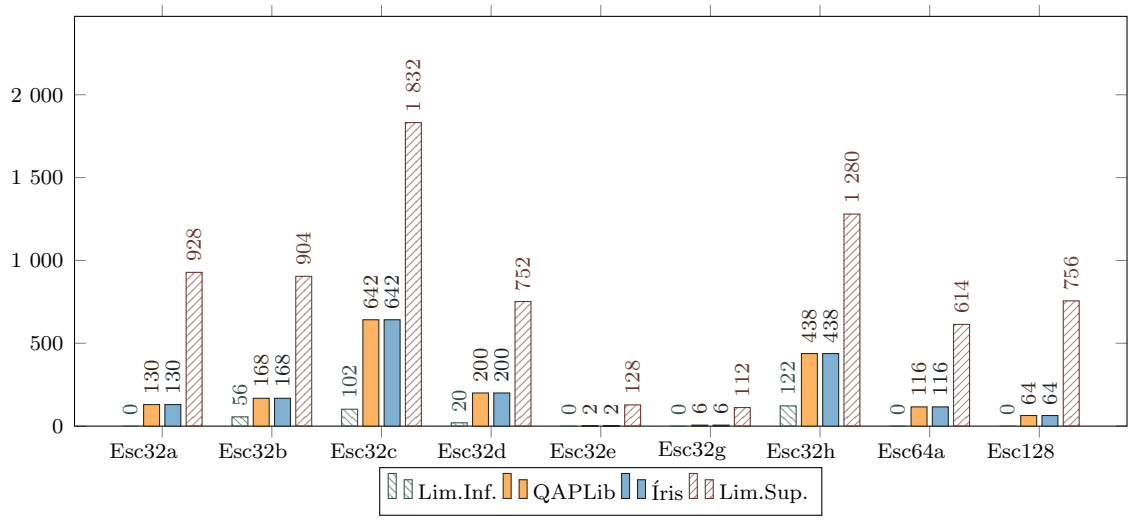


Figura 4.19: Gráficos Esc32a-Esc128.

Had12-Kra32

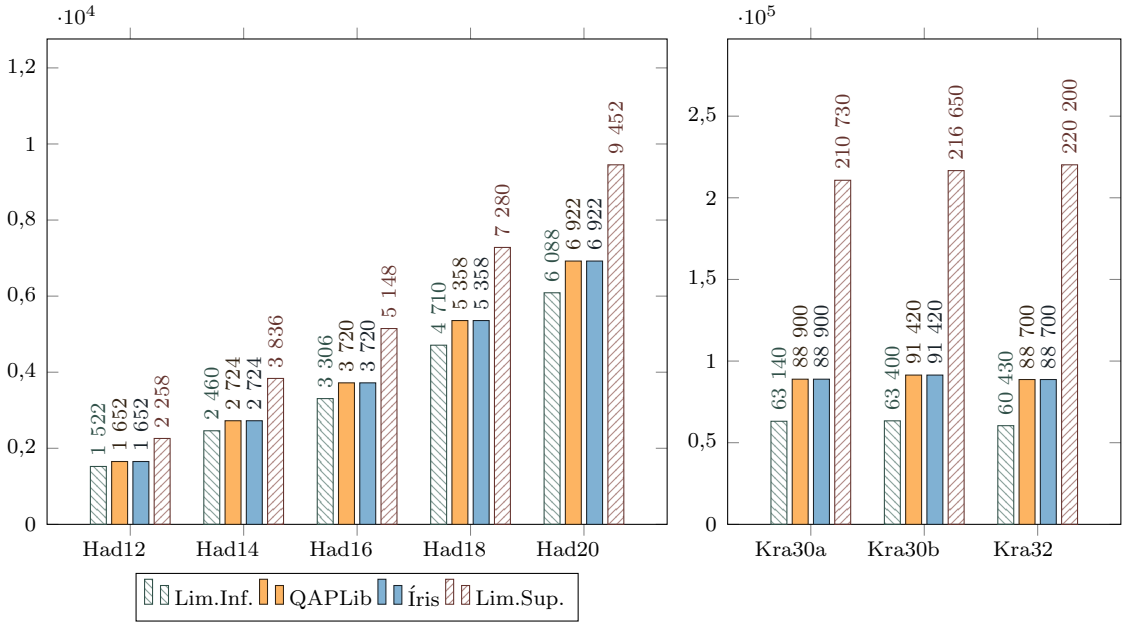


Figura 4.20: Gráficos Had12-Kra32.

Lipa20a-Lipa50b (sem lim. sup.)

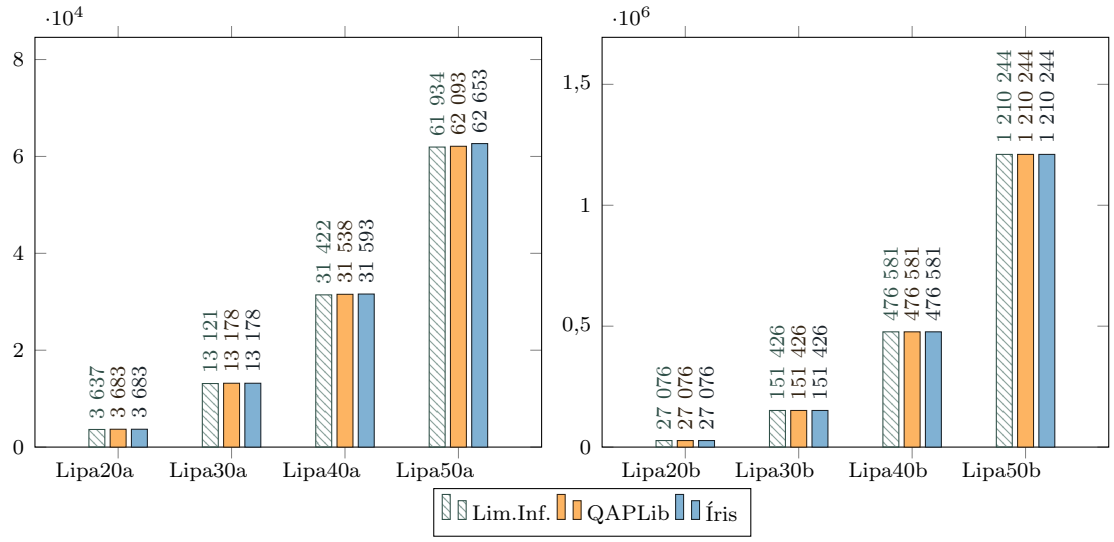


Figura 4.21: Gráficos Lipa20a-Lipa50b (sem lim. sup.).

Lipa20a-Lipa50b

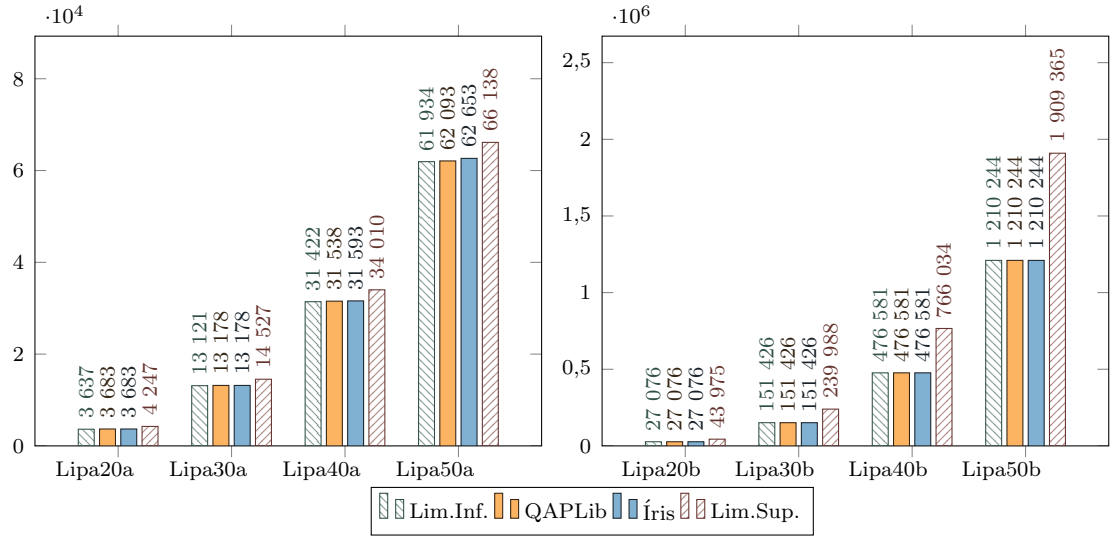


Figura 4.22: Gráficos Lipa20a-Lipa50b.

Lipa60a-Lipa90b (sem lim. sup.)

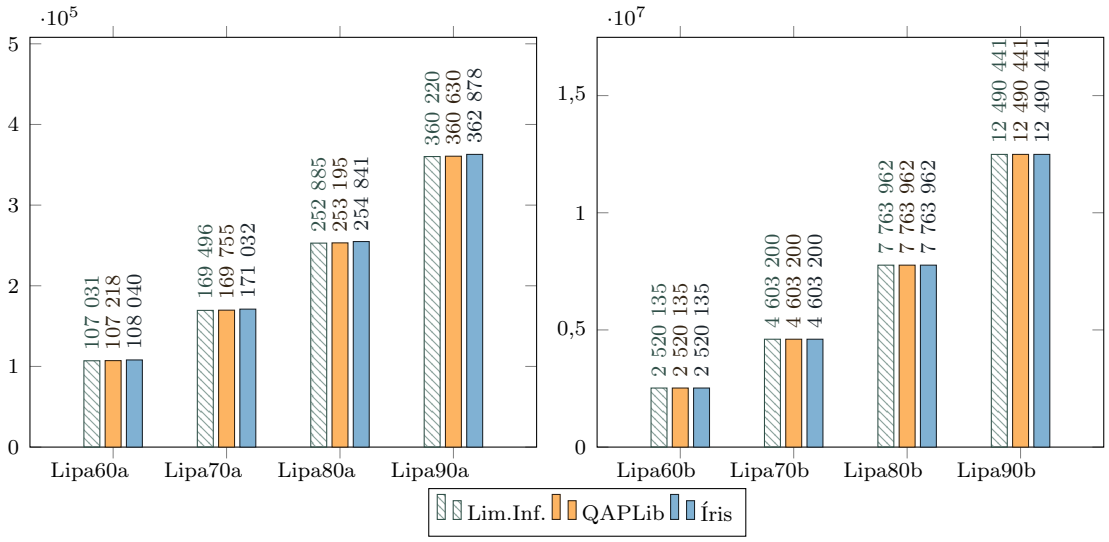


Figura 4.23: Gráficos Lipa60a-Lipa90b (sem lim. sup.).

Lipa60a-Lipa90b

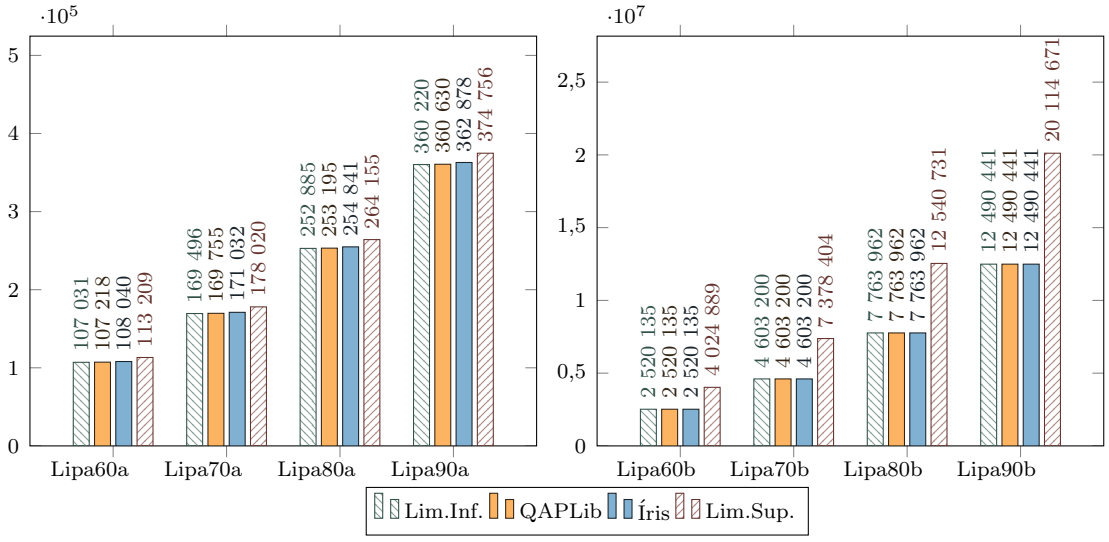


Figura 4.24: Gráficos Lipa60a-Lipa90b.

Nug12-Nug18

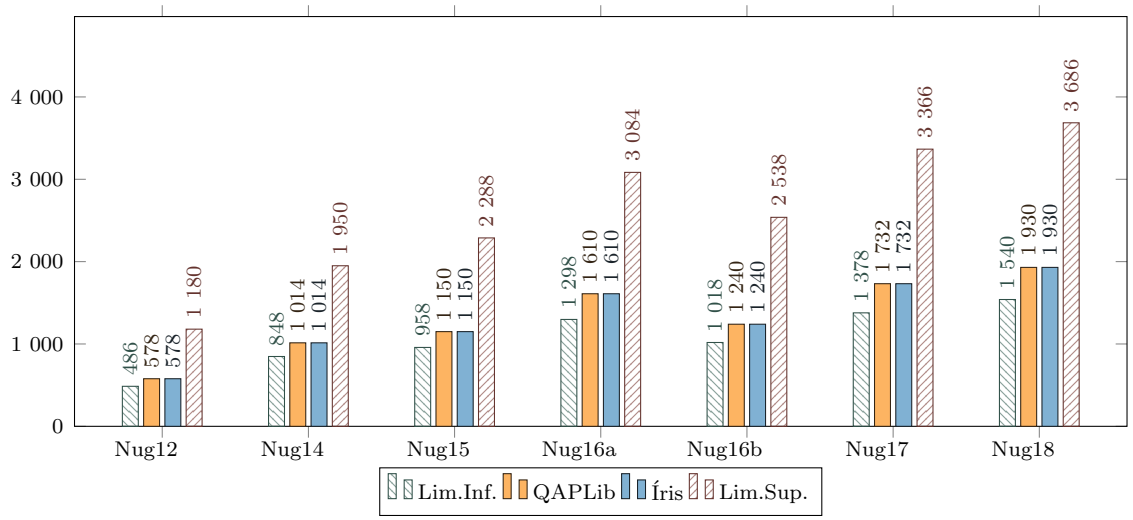


Figura 4.25: Gráficos Nug12-Nug18.

Nug20-Nug30

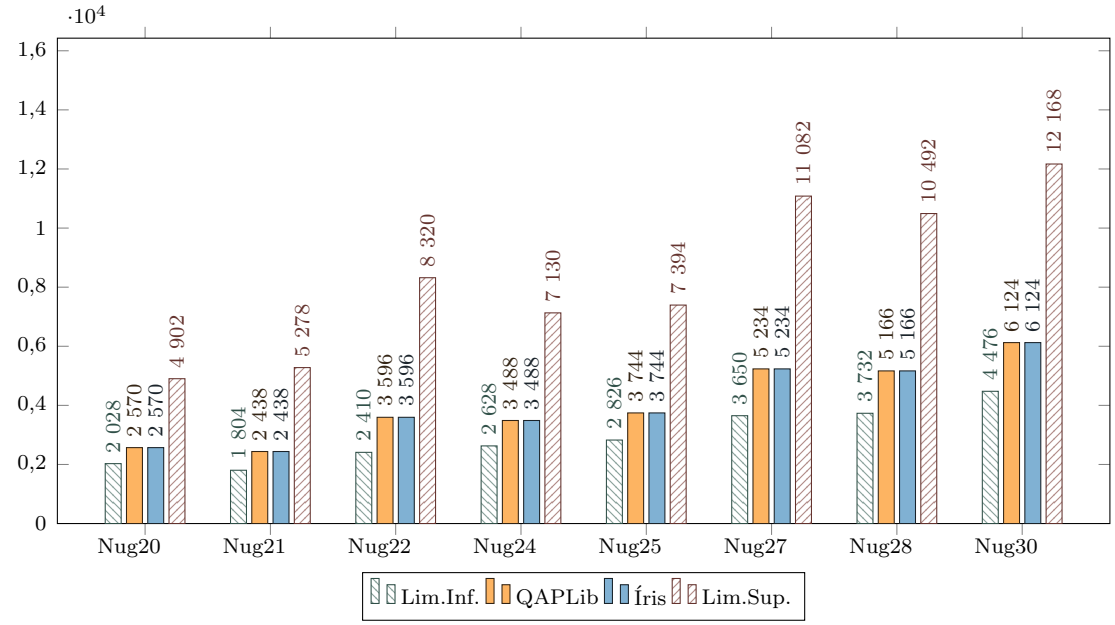


Figura 4.26: Gráficos Nug20-Nug30.

Rou12-Scr20

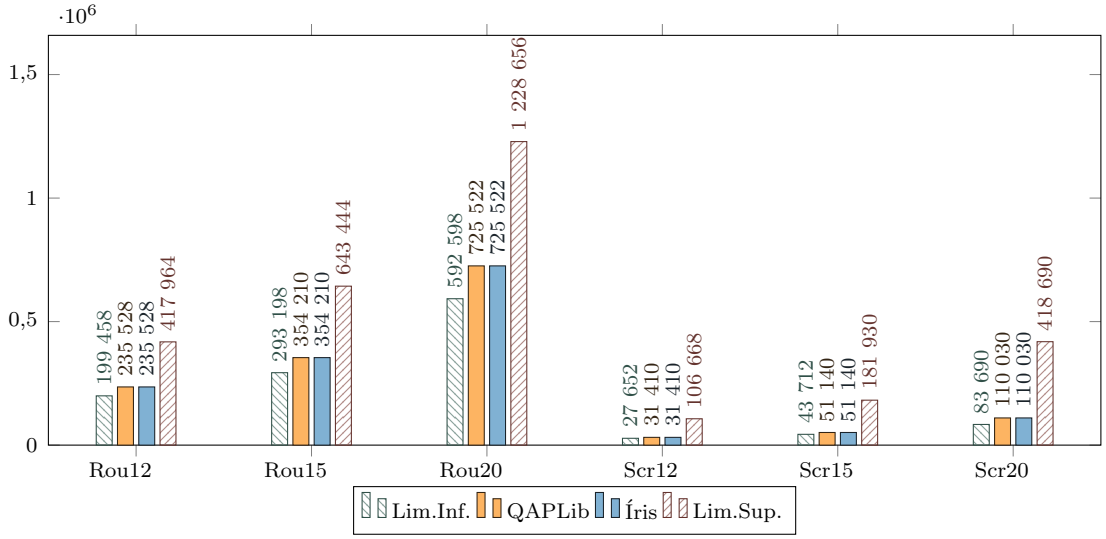


Figura 4.27: Gráficos Rou12-Scr20.

Sko42-Sko90

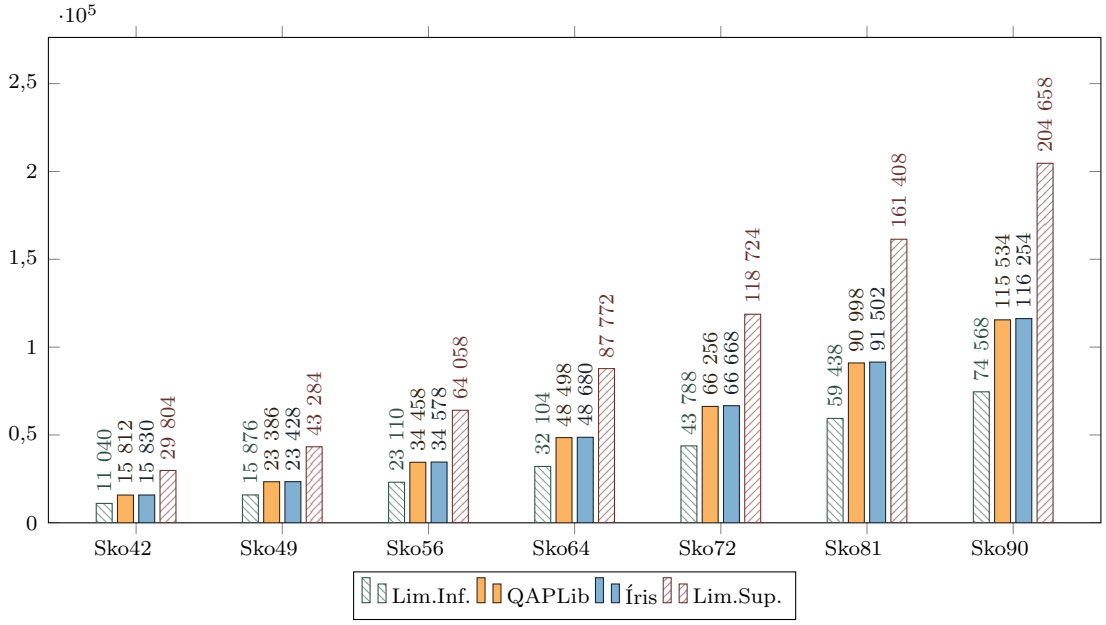


Figura 4.28: Gráficos Sko42-Sko90.

Sko100a-Sko100f (sem lim. sup.)

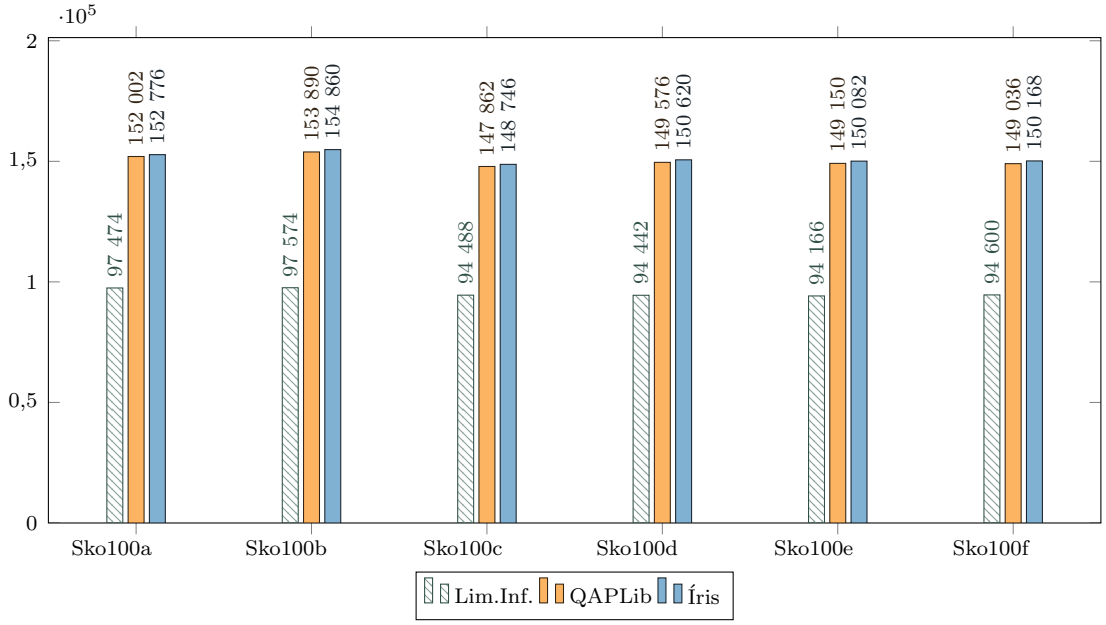


Figura 4.29: Gráficos Sko100a-Sko100f (sem lim. sup.)

Sko100a-Sko100f

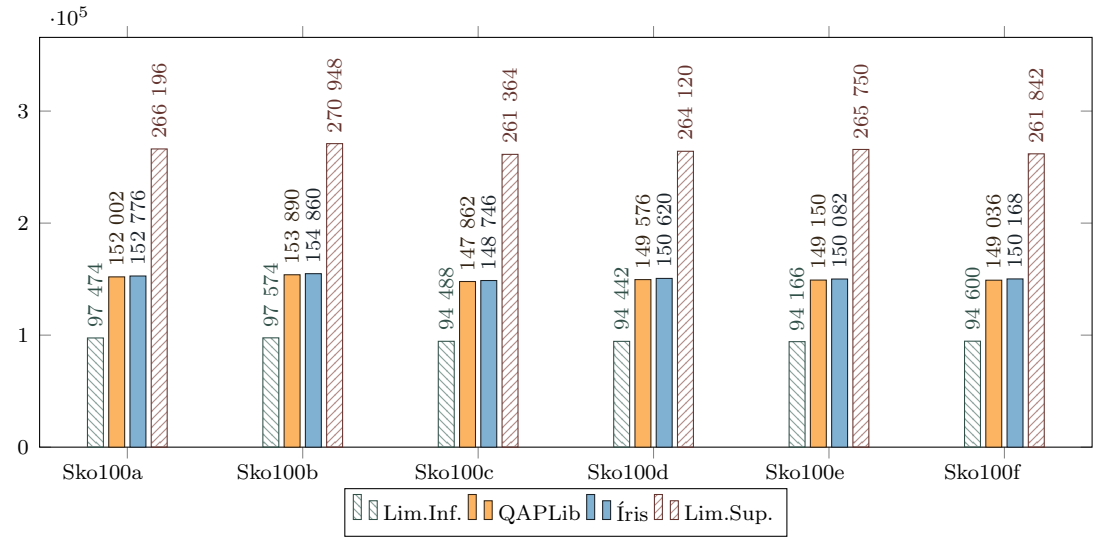


Figura 4.30: Gráficos Sko100a-Sko100f.

Ste36a-Ste36c (sem lim. sup.)

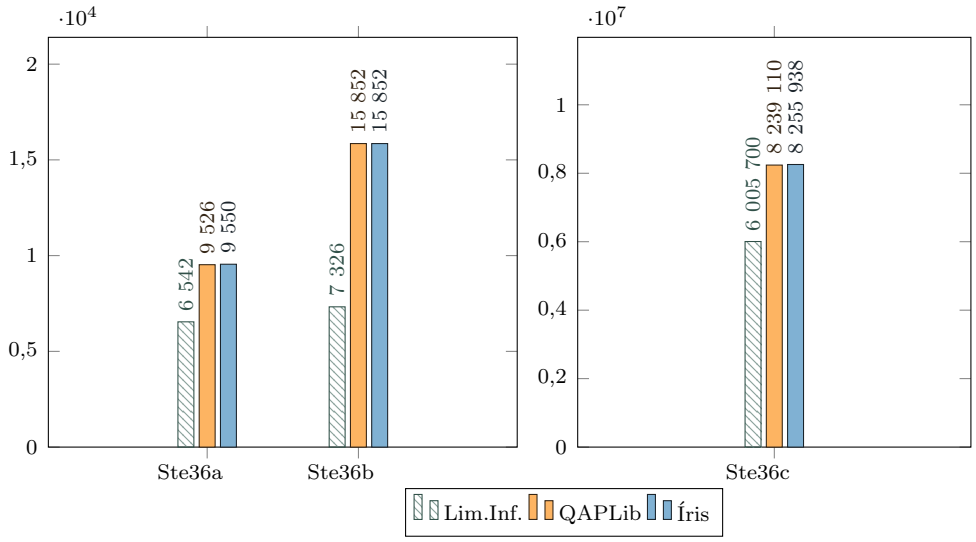


Figura 4.31: Gráficos Ste36a-Ste36c (sem lim. sup.).

Ste36a-Ste36c

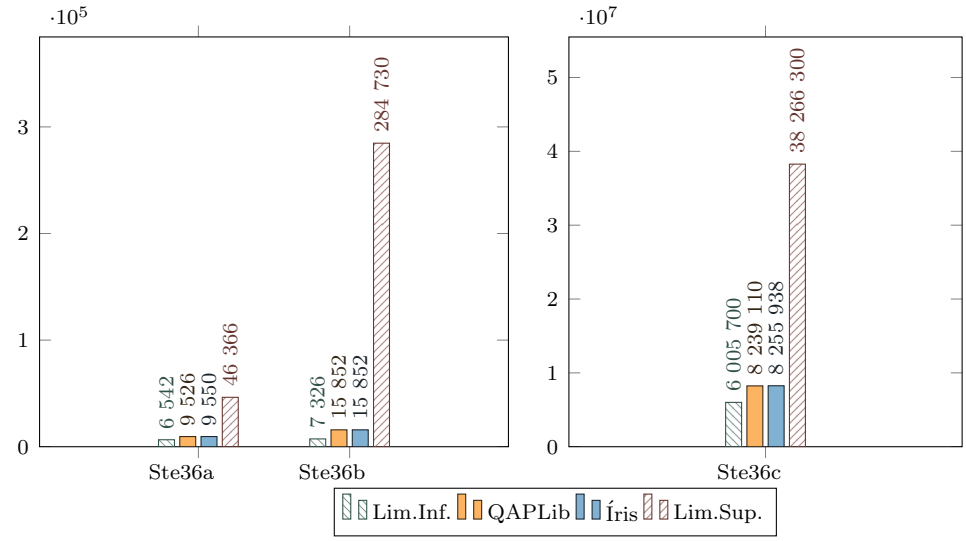


Figura 4.32: Gráficos Ste36a-Ste36c.

Tai12a-Tai20b (sem lim. sup.)

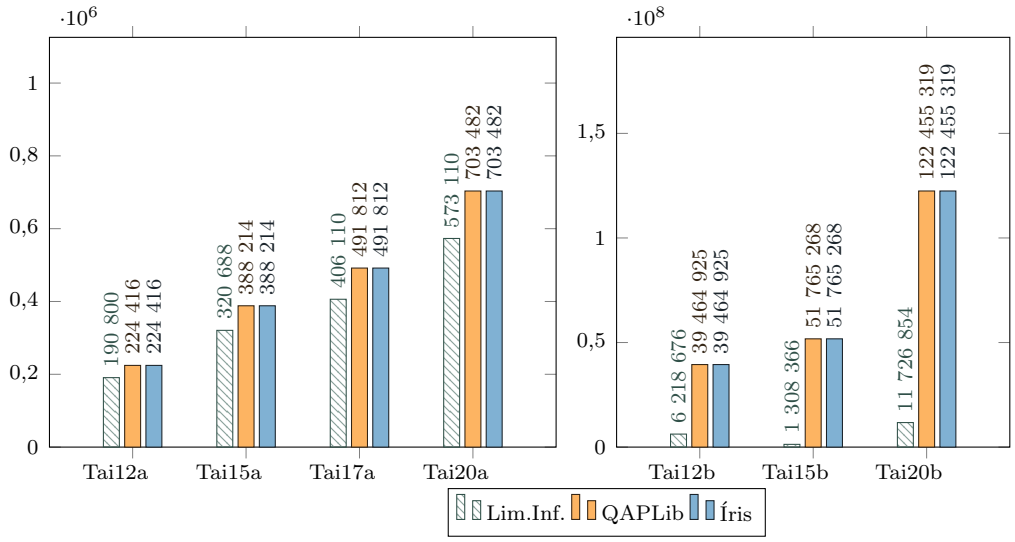


Figura 4.33: Gráficos Tai12a-Tai20b (sem lim. sup.).

Tai12a-Tai20b

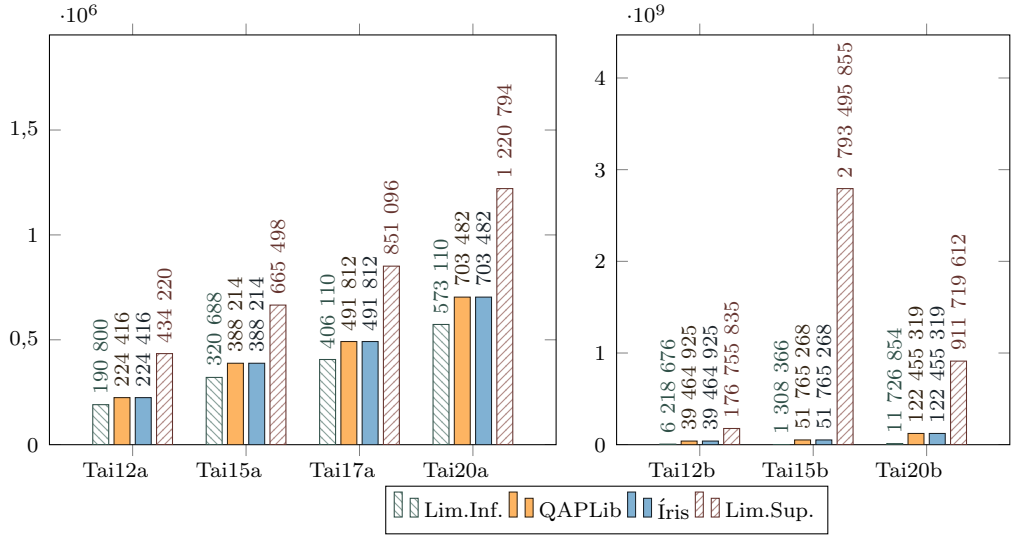


Figura 4.34: Gráficos Tai12a-Tai20b.

Tai25a-Tai40b (sem lim. sup.)

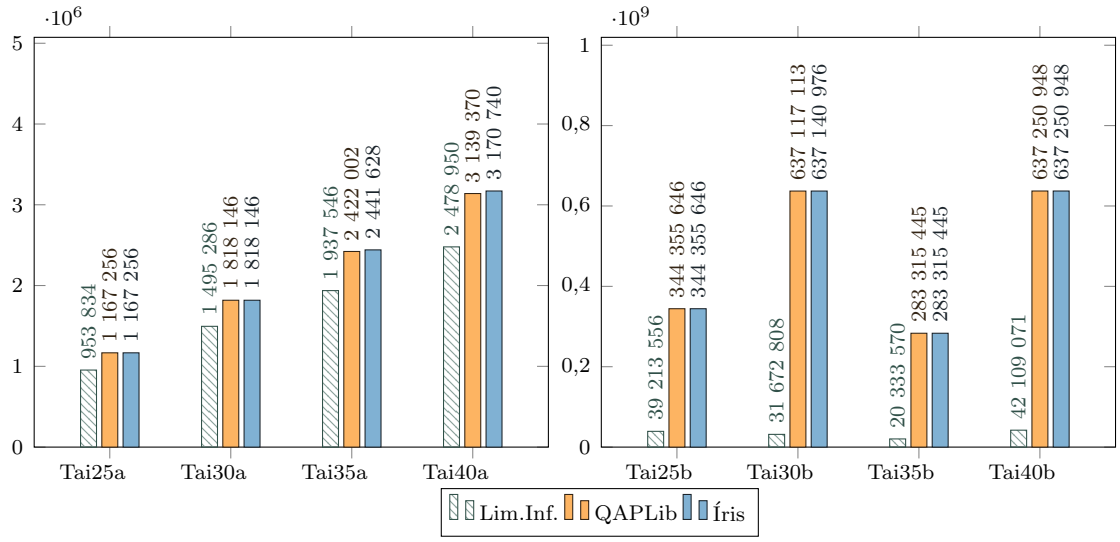


Figura 4.35: Gráficos Tai25a-Tai40b (sem lim. sup.).

Tai25a-Tai40b

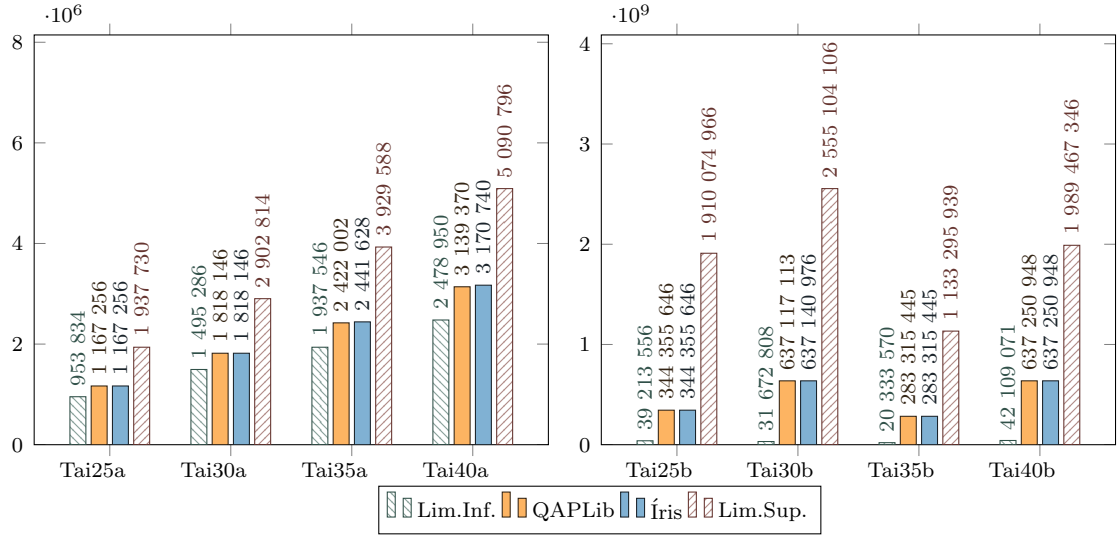


Figura 4.36: Gráficos Tai25a-Tai40b.

Tai50a-Tai80b (sem lim. sup.)

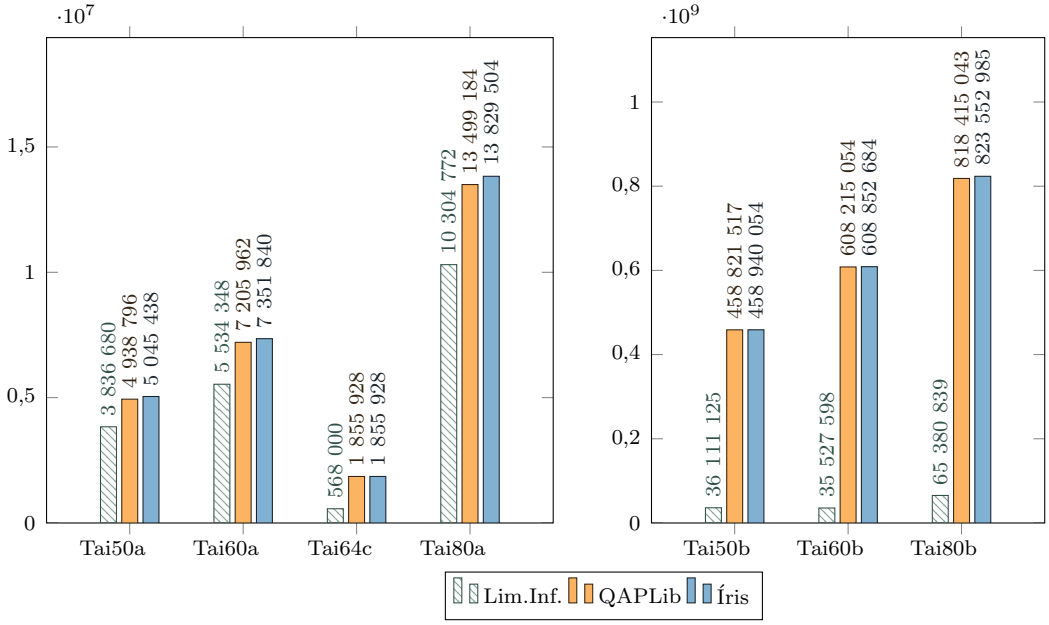


Figura 4.37: Gráficos Tai50a-Tai80b (sem lim. sup.).

Tai50a-Tai80b

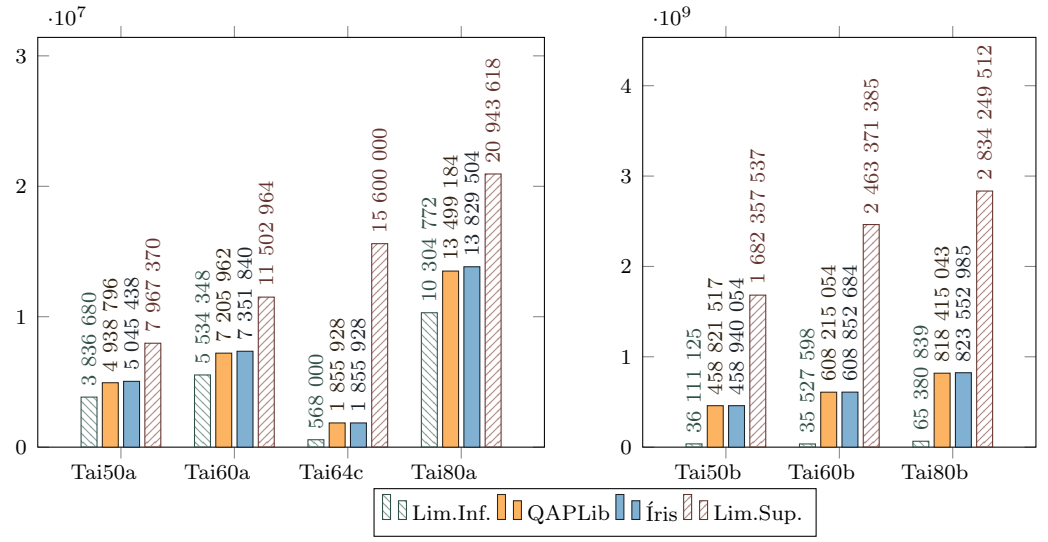


Figura 4.38: Gráficos Tai50a-Tai80b.

Tai100a-Tai256c (sem lim. sup.)

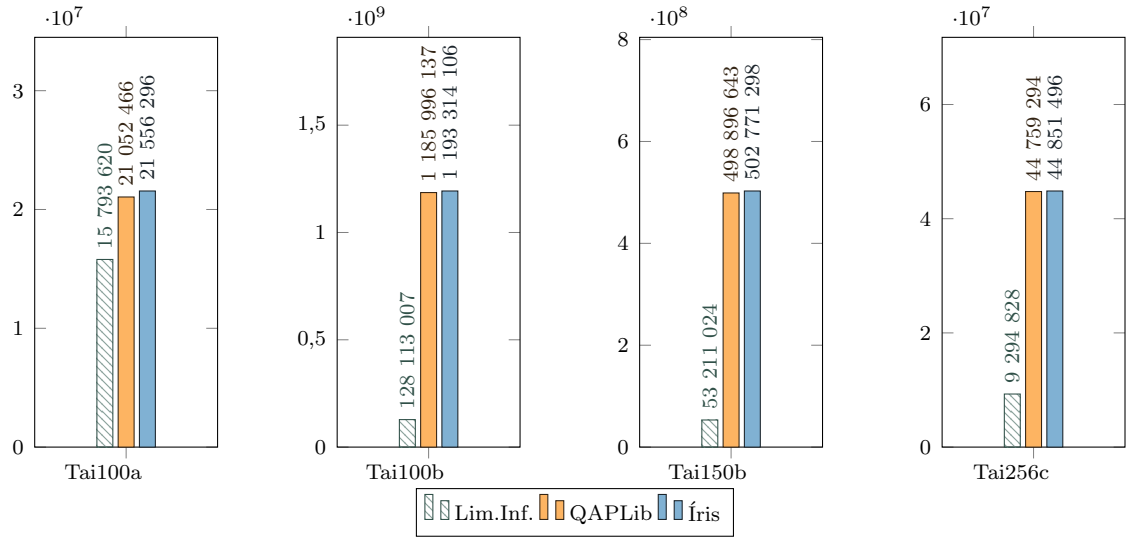


Figura 4.39: Gráficos Tai100a-Tai256c (sem lim. sup.).

Tai100a-Tai256c

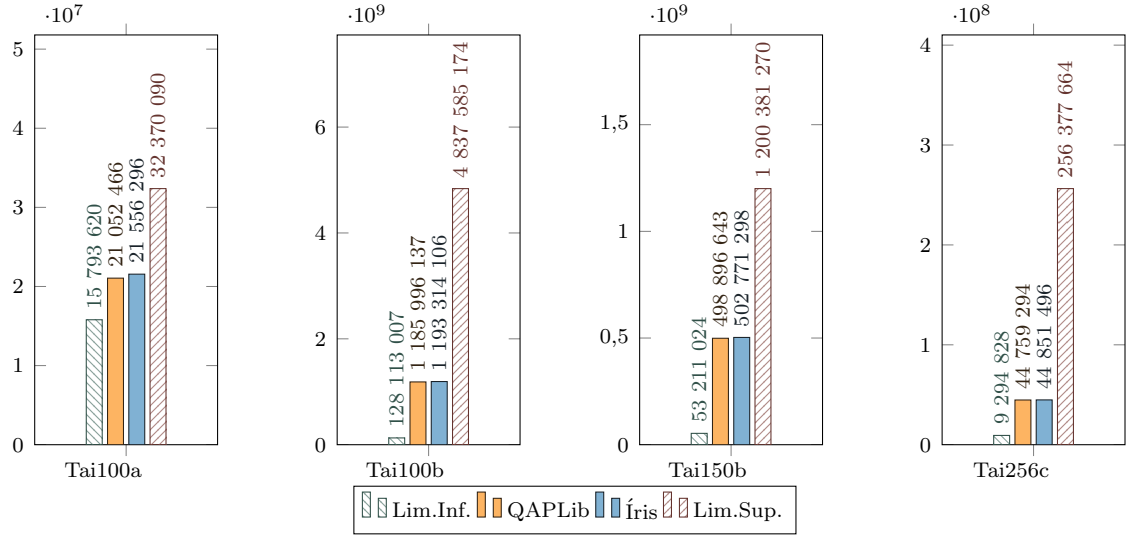


Figura 4.40: Gráficos Tai100a-Tai256c.

Tho30-Wil100 (sem lim. sup.)

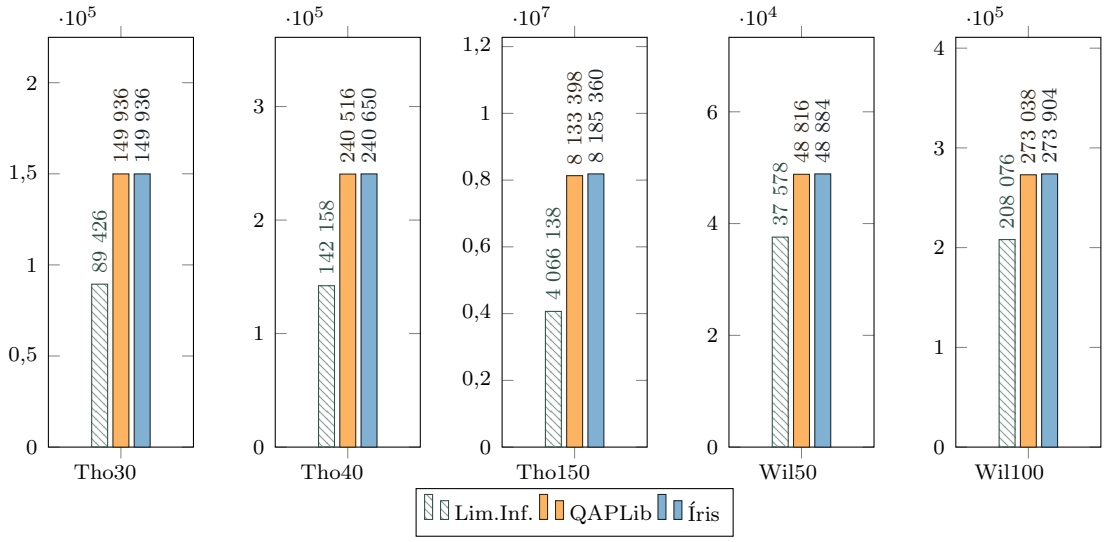


Figura 4.41: Gráficos Tho30-Wil100 (sem lim. sup.).

Tho30-Wil100

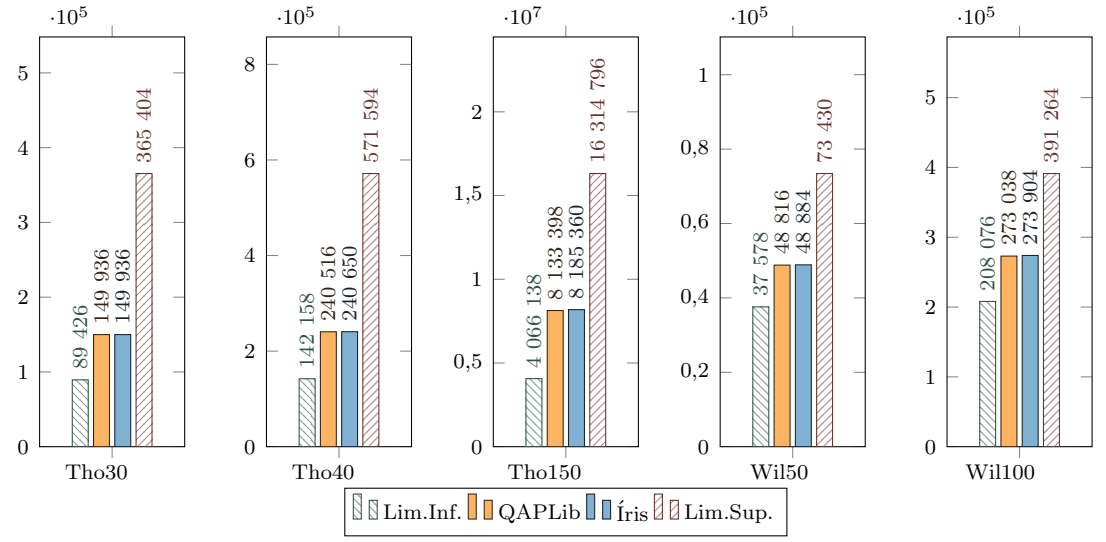


Figura 4.42: Gráficos Tho30-Wil100.

4.1.5 Discussão de Resultados de Íris

Íris procura estabelecer uma boa relação entre rapidez e qualidade de solução – semelhantemente às demais heurísticas – mas não deixa a diversidade das soluções iniciais a cargo de randomismo nem de construção gulosa (pois esses são os fatores que mais "viciam" a busca heurística). Cada uma de suas características seriam muito pouco efetivas isoladamente: a diversidade inicial, busca-bruta com SJT, busca local e path-relinking, mas a articulação desses componentes consegue balancear tempo e qualidade conforme demostraram os testes empíricos. O maior tempo de procura em alguns problemas mostra maior compromisso com a estabilidade do ritmo de melhora em função do tempo do que com os atalhos gulosos, cujo uso em demasia poderia desvirtuar a busca.

Nossa proposta com Íris é fornecer um arcabouço modular para hibridizações heurísticas em vez de uma solução definitiva e "autônoma". Ela fornece um comportamento estável de diversidade de soluções iniciais e uma ferramenta de busca de força-bruta que pode ser explorada quando há uma ordenação "gulosa" dos índices das matrizes com relação à probabilidade de um índice alterar significativamente o resultado ao ser permutado com outro. Nossos testes não são conclusivos, pois não dispúnhamos de tempo nem dos massivos recursos computacionais necessário para consolidar as conclusões sobre a qualidade de seu comportamento, mas ainda assim, Íris mostrou-se, para uma primeira experiência, satisfatória e promissora.

Capítulo 5

Conclusões

> Machines take me by surprise with great frequency.
>
> A. M. Turing (1950)[17]

A revisão da literatura heurística de PQA e realização da implementação e testes empíricos de metaheurísticas nos permitiu compreender melhor o problema e assimilar algumas noções e padrões dos métodos de solução aproximada experimentados para o PQA. Dado o desconcertante aumento de dificuldade do problema em função de seu tamanho e a vastidão do espaço de solução de um problema razoavelmente pequeno ($n \simeq 30$) se pôde notar a ineficácia de muitos possíveis métodos que até poderiam fazer sentido num conjunto menor de soluções, mas que se mostraram infrutíferos na prática.

Pode-se notar um decréscimo de publicações sobre PQA [4] nos últimos anos, provavelmente porque não se supõe que se possa avançar muito mais do que o que já se fez por PQA assim como por outros problemas NP-Difíceis. Lidar com problemas NP-Difíceis é uma tarefa árdua e pouco intuitiva que torna fácil se enganar sobre a consequência da mudança de algum comportamento de métodos heurísticos. Por essa razão foi extremamente difícil de, dentre as diversas variações que testamos, encontrar algumas que efetivamente melhorassem o resultado em verificações empíricas para um conjunto razoável de problemas do QAPLib. Felizmente algumas tentativas frutificaram no sentido de melhorar o comportamento de algumas heurísticas existentes e até mesmo formular uma metaheurística que sanasse problemas que identificamos no conjunto de métodos que testamos. Ainda assim, a conclusão, em

poucas palavras, é que ainda que haja sobreposição de algumas funcionalidades, elas funcionam melhor como complementares do que autônomas.

Com relação à Íris e seus resultados que consideremos promissores, infelizmente só a pudemos apresentar em nível bastante exploratório. Não podemos afirmar, por exemplo, que ela não pudesse ser assimilada como parte de outro método de modo muito mais interessante tal qual FANT evoluiu de Ant System assimilando características de Algoritmos Genéticos, e GRASP se compôs quase inteiramente de métodos publicados por Busca Tabu. Assim, se pode considerar que Íris foi apenas uma ponte entre algoritmos combinatórios e números fatorádicos com heurísticas para PQA, ou podemos vê-la através dos conceitos subjacentes dessa composição que inclui uma indicação empírica de que o randomismo deve ser contido e bem-canalizado em vez de tratado como fonte confiável de diversidade e como uma tentativa de conciliar a estabilidade de algoritmo exatos com atalhos heurísticos e, principalmente, como um modo simples, via memória implícita, de se referir e iterar com diversidade por um espaço de solução tão astronomicamente grande que sequer poderíamos escrever em base decimal o índice da permutação para a qual desejaríamos iterar em seguida e que, pelo menos na literatura de PQA, não encontrei uma composição absolutamente idêntica.

5.1 Trabalhos Futuros

Ensaiamos testes sobre outra variação[1] de Íris – mas terá de ser melhor estabelecida futuramente – com a ordenaçcão prévia de ambas as matrizes por algum fator que aumente a probabilidade de encontrar no início da enumeração melhores soluções (conforme o comportamento do enumerador) sem, contudo, tolhir a busca posterior dado que a estrutura de Íris previne o vício de padrões de solução inicial. Em nossos ensaios, permutamos previamente ambas as matrizes pela ordenação de índices-vértices a partir de um fato e traduzimos solução para o problema original revertendo-se o processo através das permutações que foram aplicadas às matrizes. A ordenação dos vértices aplicada no ensaio foi não-decrescente por coeficiente de variância dos valores dos arcos ligados ao vértice.

[1] Que em tese seria a melhor, pois a utilização de SJT em vez de Heap faz muito mais sentido quando é o caso de a busca ser mais interessante entre índices vizinhos privilegiando os últimos.

Essa ordenação prévia entretanto só tinha efeito quanto à busca local via SJT pois a busca local de primeira-melhoria tem ordem randomizada. Retirar o randomismo desse passo promovendo uma inteligência de busca pela ordenação matricial de índices por um critério guloso também pode ser proveitoso. Do mesmo modo, interessa explorar hibridizações de Íris com FANT e outras metaheurísticas evitando sobreposição de funcionalidades, mas ampliando a funcionalidade pela complementariedade.

Mas principalmente esperamos, em trabalhos futuros, explorar o PQA quanto ao modo de tratar e recuperar os dados. Por exemplo, as células de matrizes de distância poderiam ser calculadas por demanda a partir de vetores em espaço de Hilbert e no mesmo problema termos um grafo incompleto para representar o fluxo em vez de matrizes esparsas. Diferentemente dos zeros das matrizes, a ausência de arcos poderia não ser visitada nos cálculos de custo e delta agilizando o processo e possivelmente oferecendo recursos para uma busca local mais inteligente em problemas enormes de fluxo esparso como muitas vezes encontramos em demandas reais.

Finalmente, esperamos aplicar Íris noutros problemas permutatórios, uma vez que seu comportamento já é compatível com eles, bastando alterar a função de custo, delta e o comportamento da busca local.

Figura 5.1: Deusa Minerva.

Referências Bibliográficas

[1] KOOPMANS, T. C., BECKMANN, M. "Assignment problems and the location of economic activities", *Econometrica: journal of the Econometric Society*, pp. 53–76, 1957.

[2] KUHN, H. W. "The Hungarian method for the assignment problem", *Naval research logistics quarterly*, v. 2, n. 1-2, pp. 83–97, 1955.

[3] NYBERG, A., OTHERS. "Some reformulations for the quadratic assignment problem", 2014.

[4] LOIOLA, E. M., ABREU, N. M. M. D., BOAVENTURA NETTO, P. O. "Uma revisão comentada das abordagens do problema quadrático de alocação", *Pesquisa Operacional*, v. 24, n. 1, pp. 73–109, 2004.

[5] PIKE, R. "Go Proverbs". 2015. Disponível em: <`http://go-proverbs.github.io`>.

[6] GAREY, M. R., JOHNSON, D. S. "Computer and intractability", *A Guide to the Theory of NP-Completeness*, 1979.

[7] LAWLER, E. L. "The quadratic assignment problem", *Management science*, v. 9, n. 4, pp. 586–599, 1963.

[8] FRIEZE, A., YADEGAR, J. "On the quadratic assignment problem", *Discrete applied mathematics*, v. 5, n. 1, pp. 89–98, 1983.

[9] BURKARD, R. E., DELL'AMICO, M., MARTELLO, S. *Assignment Problems, Revised Reprint*. Siam, 2009.

[10] CHOMSKY, N. "Three models for the description of language", *Information Theory, IRE Transactions on*, v. 2, n. 3, pp. 113–124, 1956.

[11] FEO, T. A., RESENDE, M. G. "Greedy randomized adaptive search procedures", *Journal of global optimization*, v. 6, n. 2, pp. 109–133, 1995.

[12] TAILLARD, E. "FANT: fast ant system", 1998.

[13] GLOVER, F. "Tabu search-part I", *ORSA Journal on computing*, v. 1, n. 3, pp. 190–206, 1989.

[14] GLOVER, F. "Tabu search-part II", *ORSA Journal on computing*, v. 2, n. 1, pp. 4–32, 1990.

[15] KNUTH, D. E. *The Art of Computer Programming, Volume 4A, Section 7.2.1.2: Generating All Permutations*. Addison-Wesley Professional, 2011.

[16] HARDY, G. H., LITTLEWOOD, J. E., PÓLYA, G. *Inequalities*. Cambridge university press, 1952.

[17] TURING, A. M. "Computing machinery and intelligence", *Mind*, pp. 433–460, 1950.

[18] BURKARD, R., ÇELA, E., PARDALOS, P., etãl. "The Quadratic Assignment and Related Problems", *DingZhu Du and PM Pardalos (eds.), Handbook of Combinatorial Optimization*, v. 4, 1998.

[19] BURKARD, R. E., FINCKE, U. "The asymptotic probabilistic behaviour of quadratic sum assignment problems", *Zeitschrift für Operations Research*, v. 27, n. 1, pp. 73–81, 1983.

Apêndice A

Testes QAPLib

> C is quirky, flawed, and an enormous success.
>
> Dennis Ritchie

Para viabilizar a realização massiva de testes, geramos uma biblioteca de código em Go já com todos os dados numéricos da biblioteca QAPLib e a publicamos[1] de modo que outros pesquisadores possam utilizá-la se optarem por implementar soluções em linguagem Go.

A.1 Metaheurística: Íris

Com 8 cores, Busca Local de primeira-melhoria e Path-Relinking.

Prob.	Dif./QAPLib	QAPLib		Íris	Tempo (seg)
Bur26a	0.00000000	5426670	=	**5426670**	0.30
Bur26b	0.00000000	3817852	=	**3817852**	0.26
Bur26c	0.00000000	5426795	=	**5426795**	0.05
Bur26d	0.00000000	3821225	=	**3821225**	0.47
Bur26e	0.00000000	5386879	=	**5386879**	0.17
Bur26f	0.00000000	3782044	=	**3782044**	0.08
Bur26g	0.00000000	10117172	=	**10117172**	1.79
Bur26h	0.00000000	7098658	=	**7098658**	0.14
Chr12a	0.00000000	9552	=	**9552**	0.02
Chr12b	0.00000000	9742	=	**9742**	0.00
Chr12c	0.00000000	11156	=	**11156**	0.02
Chr15a	0.00000000	9896	=	**9896**	0.04
Chr15b	0.00000000	7990	=	**7990**	0.02

[1] `https://github.com/mazza/qaplib`

Prob.	Dif./QAPLib	QAPLib		Íris	Tempo (seg)
Chr15c	0.00000000	9504	=	**9504**	0.17
Chr18a	0.00000000	11098	=	**11098**	0.12
Chr18b	0.00000000	1534	=	**1534**	0.01
Chr20a	0.00000000	2192	=	**2192**	48.58
Chr20b	0.00000000	2298	=	**2298**	15.48
Chr20c	0.00000000	14142	=	**14142**	0.16
Chr22a	0.00000000	6156	=	**6156**	448.69
Chr22b	0.00581208	6194	¡	6230	444.44
Chr25a	0.00000000	3796	=	**3796**	12.65
Els19	0.00000000	17212548	=	**17212548**	0.01
Esc16a	0.00000000	68	=	**68**	0.00
Esc16b	0.00000000	292	=	**292**	0.00
Esc16c	0.00000000	160	=	**160**	0.00
Esc16d	0.00000000	16	=	**16**	0.00
Esc16e	0.00000000	28	=	**28**	0.00
Esc16f	0.00000000	0	=	**0**	0.00
Esc16g	0.00000000	26	=	**26**	0.00
Esc16h	0.00000000	996	=	**996**	0.00
Esc16i	0.00000000	14	=	**14**	0.00
Esc16j	0.00000000	8	=	**8**	0.00
Esc32a	0.00000000	130	=	**130**	6.67
Esc32b	0.00000000	168	=	**168**	0.79
Esc32c	0.00000000	642	=	**642**	0.01
Esc32d	0.00000000	200	=	**200**	0.02
Esc32e	0.00000000	2	=	**2**	0.00
Esc32g	0.00000000	6	=	**6**	0.00
Esc32h	0.00000000	438	=	**438**	0.02
Esc64a	0.00000000	116	=	**116**	0.02
Esc128	0.00000000	64	=	**64**	0.38
Had12	0.00000000	1652	=	**1652**	0.01
Had14	0.00000000	2724	=	**2724**	0.00
Had16	0.00000000	3720	=	**3720**	0.00
Had18	0.00000000	5358	=	**5358**	0.00
Had20	0.00000000	6922	=	**6922**	0.02
Kra30a	0.00000000	88900	=	**88900**	44.79
Kra30b	0.00000000	91420	=	**91420**	425.07
Kra32	0.00000000	88700	=	**88700**	35.51
Lipa20a	0.00000000	3683	=	**3683**	0.07
Lipa20b	0.00000000	27076	=	**27076**	0.00
Lipa30a	0.00000000	13178	=	**13178**	1.54
Lipa30b	0.00000000	151426	=	**151426**	0.00
Lipa40a	0.00174393	31538	¡	31593	196.89
Lipa40b	0.00000000	476581	=	**476581**	0.00
Lipa50a	0.00901873	62093	¡	62653	780.27
Lipa50b	0.00000000	1210244	=	**1210244**	0.00
Lipa60a	0.00766662	107218	¡	108040	693.29
Lipa60b	0.00000000	2520135	=	**2520135**	0.02
Lipa70a	0.00752261	169755	¡	171032	176.11
Lipa70b	0.00000000	4603200	=	**4603200**	0.01
Lipa80a	0.00650092	253195	¡	254841	651.33
Lipa80b	0.00000000	7763962	=	**7763962**	0.02

Prob.	Dif./QAPLib	QAPLib		Íris	Tempo (seg)
Lipa90a	0.00623354	360630	¡	362878	627.10
Lipa90b	0.00000000	12490441	=	**12490441**	0.02
Nug12	0.00000000	578	=	**578**	0.00
Nug14	0.00000000	1014	=	**1014**	0.04
Nug15	0.00000000	1150	=	**1150**	0.01
Nug16a	0.00000000	1610	=	**1610**	0.01
Nug16b	0.00000000	1240	=	**1240**	0.01
Nug17	0.00000000	1732	=	**1732**	0.06
Nug18	0.00000000	1930	=	**1930**	0.24
Nug20	0.00000000	2570	=	**2570**	0.10
Nug21	0.00000000	2438	=	**2438**	0.06
Nug22	0.00000000	3596	=	**3596**	0.32
Nug24	0.00000000	3488	=	**3488**	0.25
Nug25	0.00000000	3744	=	**3744**	0.29
Nug27	0.00000000	5234	=	**5234**	0.33
Nug28	0.00000000	5166	=	**5166**	31.62
Nug30	0.00000000	6124	=	**6124**	528.48
Rou12	0.00000000	235528	=	**235528**	0.03
Rou15	0.00000000	354210	=	**354210**	0.00
Rou20	0.00000000	725522	=	**725522**	0.64
Scr12	0.00000000	31410	=	**31410**	0.00
Scr15	0.00000000	51140	=	**51140**	0.02
Scr20	0.00000000	110030	=	**110030**	0.09
Sko42	0.00113838	15812	¡	15830	806.92
Sko49	0.00179595	23386	¡	23428	408.46
Sko56	0.00348250	34458	¡	34578	434.46
Sko64	0.00375273	48498	¡	48680	94.86
Sko72	0.00621830	66256	¡	66668	491.01
Sko81	0.00553858	90998	¡	91502	34.18
Sko90	0.00623193	115534	¡	116254	714.19
Sko100a	0.00509204	152002	¡	152776	414.39
Sko100b	0.00630320	153890	¡	154860	669.34
Sko100c	0.00597855	147862	¡	148746	629.42
Sko100d	0.00697973	149576	¡	150620	386.31
Sko100e	0.00624874	149150	¡	150082	702.50
Sko100f	0.00759548	149036	¡	150168	157.15
Ste36a	0.00251942	9526	¡	9550	884.34
Ste36b	0.00000000	15852	=	**15852**	69.86
Ste36c	0.00204245	8239110	¡	8255938	768.52
Tai12a	0.00000000	224416	=	**224416**	0.00
Tai12b	0.00000000	39464925	=	**39464925**	0.02
Tai15a	0.00000000	388214	=	**388214**	0.11
Tai15b	0.00000000	51765268	=	**51765268**	0.00
Tai17a	0.00000000	491812	=	**491812**	2.13
Tai20a	0.00000000	703482	=	**703482**	9.37
Tai20b	0.00000000	122455319	=	**122455319**	0.54
Tai25a	0.00000000	1167256	=	**1167256**	264.58
Tai25b	0.00000000	344355646	=	**344355646**	0.42
Tai30a	0.00000000	1818146	=	**1818146**	633.48
Tai30b	0.00003745	637117113	¡	637140976	170.93
Tai35a	0.00810321	2422002	¡	2441628	392.66

Prob.	Dif./QAPLib	QAPLib		Íris	Tempo (seg)
Tai35b	0.00000000	283315445	=	**283315445**	233.82
Tai40a	0.00999245	3139370	¡	3170740	519.50
Tai40b	0.00000000	637250948	=	**637250948**	104.98
Tai50a	0.02159271	4938796	¡	5045438	576.45
Tai50b	0.00025835	458821517	¡	458940054	800.47
Tai60a	0.02024407	7205962	¡	7351840	129.88
Tai60b	0.00104836	608215054	¡	608852684	880.50
Tai64c	0.00000000	1855928	=	**1855928**	0.05
Tai80a	0.02446963	13499184	¡	13829504	650.86
Tai80b	0.00627792	818415043	¡	823552985	580.64
Tai100a	0.02393211	21052466	¡	21556296	389.49
Tai100b	0.00617031	1185996137	¡	1193314106	649.17
Tai150b	0.00776645	498896643	¡	502771298	555.05
Tai256c	0.00205995	44759294	¡	44851496	911.03
Tho30	0.00000000	149936	=	**149936**	55.29
Tho40	0.00055714	240516	¡	240650	81.48
Tho150	0.00638872	8133398	¡	8185360	897.53
Wil50	0.00139299	48816	¡	48884	108.27
Wil100	0.00317172	273038	¡	273904	364.60

A.1.1 Íris: Com apenas um núcleo.

Com 1 core, Busca Local de primeira-melhoria e Path-Relinking.

Prob.	Dif./QAPLib	QAPLib		Íris	Tempo (seg)
Bur26a	0.00000000	5426670	=	**5426670**	0.00
Bur26b	0.00000000	3817852	=	**3817852**	0.02
Bur26c	0.00000000	5426795	=	**5426795**	0.03
Bur26d	0.00000000	3821225	=	**3821225**	0.03
Bur26e	0.00000000	5386879	=	**5386879**	0.27
Bur26f	0.00000000	3782044	=	**3782044**	0.76
Bur26g	0.00000000	10117172	=	**10117172**	0.03
Bur26h	0.00000000	7098658	=	**7098658**	0.14
Chr12a	0.00000000	9552	=	**9552**	0.03
Chr12b	0.00000000	9742	=	**9742**	0.00
Chr12c	0.00000000	11156	=	**11156**	0.28
Chr15a	0.00000000	9896	=	**9896**	0.30
Chr15b	0.00000000	7990	=	**7990**	0.04
Chr15c	0.00000000	9504	=	**9504**	0.16
Chr18a	0.00000000	11098	=	**11098**	8.15
Chr18b	0.00000000	1534	=	**1534**	0.01
Chr20a	0.00182482	2192	¡	2196	10.41
Chr20b	0.03133159	2298	¡	2370	7.61
Chr20c	0.00000000	14142	=	**14142**	0.77
Chr22a	0.00747238	6156	¡	6202	1.83
Chr22b	0.01743623	6194	¡	6302	14.29
Chr25a	0.09694415	3796	¡	4164	2.04
Els19	0.00000000	17212548	=	**17212548**	0.07
Esc16a	0.00000000	68	=	**68**	0.00
Esc16b	0.00000000	292	=	**292**	0.00
Esc16c	0.00000000	160	=	**160**	0.00
Esc16d	0.00000000	16	=	**16**	0.00
Esc16e	0.00000000	28	=	**28**	0.00
Esc16f	0.00000000	0	=	**0**	0.00
Esc16g	0.00000000	26	=	**26**	0.00
Esc16h	0.00000000	996	=	**996**	0.00
Esc16i	0.00000000	14	=	**14**	0.00
Esc16j	0.00000000	8	=	**8**	0.00
Esc32a	0.03076923	130	¡	134	0.18
Esc32b	0.00000000	168	=	**168**	2.64
Esc32c	0.00000000	642	=	**642**	0.00
Esc32d	0.00000000	200	=	**200**	0.00
Esc32e	0.00000000	2	=	**2**	0.00
Esc32g	0.00000000	6	=	**6**	0.00
Esc32h	0.00000000	438	=	**438**	0.03
Esc64a	0.00000000	116	=	**116**	0.03
Esc128	0.00000000	64	=	**64**	0.44
Had12	0.00000000	1652	=	**1652**	0.00
Had14	0.00000000	2724	=	**2724**	0.01
Had16	0.00000000	3720	=	**3720**	0.00
Had18	0.00000000	5358	=	**5358**	0.00
Had20	0.00000000	6922	=	**6922**	0.00

Prob.	Dif./QAPLib	QAPLib		Íris	Tempo (seg)
Kra30a	0.00787402	88900	¡	89600	7.23
Kra30b	0.00251586	91420	¡	91650	6.72
Kra32	0.00225479	88700	¡	88900	12.73
Lipa20a	0.00000000	3683	=	**3683**	0.30
Lipa20b	0.00000000	27076	=	**27076**	0.00
Lipa30a	0.00000000	13178	=	**13178**	1.40
Lipa30b	0.00000000	151426	=	**151426**	0.00
Lipa40a	0.01078065	31538	¡	31878	18.59
Lipa40b	0.00000000	476581	=	**476581**	0.00
Lipa50a	0.01019439	62093	¡	62726	18.83
Lipa50b	0.00000000	1210244	=	**1210244**	0.00
Lipa60a	0.00843142	107218	¡	108122	28.51
Lipa60b	0.00000000	2520135	=	**2520135**	0.01
Lipa70a	0.00781715	169755	¡	171082	4.04
Lipa70b	0.00000000	4603200	=	**4603200**	0.01
Lipa80a	0.00737771	253195	¡	255063	0.64
Lipa80b	0.00000000	7763962	=	**7763962**	0.00
Lipa90a	0.00666334	360630	¡	363033	0.17
Lipa90b	0.00000000	12490441	=	**12490441**	0.01
Nug12	0.00000000	578	=	**578**	0.04
Nug14	0.00000000	1014	=	**1014**	0.16
Nug15	0.00000000	1150	=	**1150**	0.01
Nug16a	0.00000000	1610	=	**1610**	0.02
Nug16b	0.00000000	1240	=	**1240**	0.01
Nug17	0.00000000	1732	=	**1732**	0.03
Nug18	0.00000000	1930	=	**1930**	0.25
Nug20	0.00000000	2570	=	**2570**	0.15
Nug21	0.00000000	2438	=	**2438**	0.47
Nug22	0.00000000	3596	=	**3596**	0.44
Nug24	0.00000000	3488	=	**3488**	0.24
Nug25	0.00000000	3744	=	**3744**	2.77
Nug27	0.00000000	5234	=	**5234**	1.05
Nug28	0.00193573	5166	¡	5176	1.22
Nug30	0.00293926	6124	¡	6142	10.52
Rou12	0.00000000	235528	=	**235528**	0.01
Rou15	0.00000000	354210	=	**354210**	0.00
Rou20	0.00000000	725522	=	**725522**	1.67
Scr12	0.00000000	31410	=	**31410**	0.00
Scr15	0.00000000	51140	=	**51140**	0.25
Scr20	0.00000000	110030	=	**110030**	0.41
Sko42	0.00581837	15812	¡	15904	3.77
Sko49	0.00598649	23386	¡	23526	0.97
Sko56	0.00673283	34458	¡	34690	24.64
Sko64	0.00433008	48498	¡	48708	9.45
Sko72	0.01080657	66256	¡	66972	5.18
Sko81	0.00804413	90998	¡	91730	9.05
Sko90	0.00832655	115534	¡	116496	24.84
Sko100a	0.00615781	152002	¡	152938	69.18
Sko100b	0.00820066	153890	¡	155152	9.74
Sko100c	0.00719590	147862	¡	148926	3.77
Sko100d	0.00570947	149576	¡	150430	21.72

Prob.	Dif./QAPLib	QAPLib		Íris	Tempo (seg)
Sko100e	0.01225612	149150	¡	150978	9.06
Sko100f	0.01223865	149036	¡	150860	3.84
Ste36a	0.01112744	9526	¡	9632	8.36
Ste36b	0.01085037	15852	¡	16024	7.47
Ste36c	0.00608003	8239110	¡	8289204	11.30
Tai12a	0.00000000	224416	=	**224416**	0.00
Tai12b	0.00000000	39464925	=	**39464925**	0.02
Tai15a	0.00000000	388214	=	**388214**	0.07
Tai15b	0.00000000	51765268	=	**51765268**	0.00
Tai17a	0.00000000	491812	=	**491812**	1.37
Tai20a	0.00000000	703482	=	**703482**	17.54
Tai20b	0.00000000	122455319	=	**122455319**	0.01
Tai25a	0.00867847	1167256	¡	1177386	22.69
Tai25b	0.00000000	344355646	=	**344355646**	12.05
Tai30a	0.01091442	1818146	¡	1837990	14.86
Tai30b	0.00270287	637117113	¡	638839157	0.10
Tai35a	0.01925597	2422002	¡	2468640	1.75
Tai35b	0.00114371	283315445	¡	283639475	27.42
Tai40a	0.02131638	3139370	¡	3206290	12.43
Tai40b	0.00005117	637250948	¡	637283558	11.76
Tai50a	0.02434035	4938796	¡	5059008	6.18
Tai50b	0.00096862	458821517	¡	459265941	0.22
Tai60a	0.02790689	7205962	¡	7407058	5.22
Tai60b	0.00480610	608215054	¡	611138198	15.80
Tai64c	0.00000000	1855928	=	**1855928**	0.00
Tai80a	0.02447778	13499184	¡	13829614	22.39
Tai80b	0.01632900	818415043	¡	831778942	3.23
Tai100a	0.02687723	21052466	¡	21618298	1.30
Tai100b	0.01691432	1185996137	¡	1206056455	19.48
Tai150b	0.01891694	498896643	¡	508334242	1.13
Tai256c	0.00253154	44759294	¡	44872604	14.72
Tho30	0.00382830	149936	¡	150510	13.38
Tho40	0.00504748	240516	¡	241730	20.76
Tho150	0.01175941	8133398	¡	8229042	14.14
Wil50	0.00311373	48816	¡	48968	5.95
Wil100	0.00573547	273038	¡	274604	4.70

A.1.2 Íris: Com apenas um núcleo.

Com 1 core e Busca Local de primeira-melhoria (sem Path-Relinking).

Prob.	Dif./QAPLib	QAPLib		Íris	Tempo (seg)
Bur26a	0.00000000	5426670	=	**5426670**	1.84
Bur26b	0.00000000	3817852	=	**3817852**	3.55
Bur26c	0.00000000	5426795	=	**5426795**	0.69
Bur26d	0.00000000	3821225	=	**3821225**	0.15
Bur26e	0.00000000	5386879	=	**5386879**	0.00
Bur26f	0.00000000	3782044	=	**3782044**	1.53
Bur26g	0.00000000	10117172	=	**10117172**	3.89
Bur26h	0.00000000	7098658	=	**7098658**	0.00
Chr12a	0.00000000	9552	=	**9552**	0.00
Chr12b	0.00000000	9742	=	**9742**	0.00
Chr12c	0.00000000	11156	=	**11156**	0.26
Chr15a	0.00000000	9896	=	**9896**	0.02
Chr15b	0.00000000	7990	=	**7990**	0.01
Chr15c	0.00000000	9504	=	**9504**	0.59
Chr18a	0.00000000	11098	=	**11098**	12.54
Chr18b	0.00000000	1534	=	**1534**	0.01
Chr20a	0.01459854	2192	¡	2224	12.50
Chr20b	0.00000000	2298	=	**2298**	25.29
Chr20c	0.00000000	14142	=	**14142**	2.93
Chr22a	0.00747238	6156	¡	6202	5.43
Chr22b	0.01840491	6194	¡	6308	44.36
Chr25a	0.04162276	3796	¡	3954	19.67
Els19	0.00000000	17212548	=	**17212548**	1.27
Esc16a	0.00000000	68	=	**68**	0.00
Esc16b	0.00000000	292	=	**292**	0.00
Esc16c	0.00000000	160	=	**160**	0.00
Esc16d	0.00000000	16	=	**16**	0.00
Esc16e	0.00000000	28	=	**28**	0.01
Esc16f	0.00000000	0	=	**0**	0.00
Esc16g	0.00000000	26	=	**26**	0.00
Esc16h	0.00000000	996	=	**996**	0.00
Esc16i	0.00000000	14	=	**14**	0.00
Esc16j	0.00000000	8	=	**8**	0.00
Esc32a	0.00000000	130	=	**130**	3.96
Esc32b	0.00000000	168	=	**168**	1.16
Esc32c	0.00000000	642	=	**642**	0.00
Esc32d	0.00000000	200	=	**200**	0.01
Esc32e	0.00000000	2	=	**2**	0.00
Esc32g	0.00000000	6	=	**6**	0.00
Esc32h	0.00000000	438	=	**438**	0.02
Esc64a	0.00000000	116	=	**116**	0.05
Esc128	0.00000000	64	=	**64**	0.40
Had12	0.00000000	1652	=	**1652**	0.00
Had14	0.00000000	2724	=	**2724**	0.00
Had16	0.00000000	3720	=	**3720**	0.00
Had18	0.00000000	5358	=	**5358**	0.00
Had20	0.00000000	6922	=	**6922**	0.10

Prob.	Dif./QAPLib	QAPLib		Íris	Tempo (seg)
Kra30a	0.01529809	88900	¡	90260	16.11
Kra30b	0.00185955	91420	¡	91590	8.27
Kra32	0.01307779	88700	¡	89860	1.93
Lipa20a	0.00000000	3683	=	**3683**	0.05
Lipa20b	0.00000000	27076	=	**27076**	0.00
Lipa30a	0.00000000	13178	=	**13178**	0.67
Lipa30b	0.00000000	151426	=	**151426**	0.00
Lipa40a	0.01103431	31538	¡	31886	1.12
Lipa40b	0.00000000	476581	=	**476581**	0.00
Lipa50a	0.01056480	62093	¡	62749	19.34
Lipa50b	0.00000000	1210244	=	**1210244**	0.00
Lipa60a	0.00867392	107218	¡	108148	1.63
Lipa60b	0.00000000	2520135	=	**2520135**	0.01
Lipa70a	0.00777591	169755	¡	171075	38.64
Lipa70b	0.00000000	4603200	=	**4603200**	0.01
Lipa80a	0.00710125	253195	¡	254993	22.93
Lipa80b	0.00000000	7763962	=	**7763962**	0.02
Lipa90a	0.00666889	360630	¡	363035	3.76
Lipa90b	0.00000000	12490441	=	**12490441**	0.03
Nug12	0.00000000	578	=	**578**	0.02
Nug14	0.00000000	1014	=	**1014**	0.36
Nug15	0.00000000	1150	=	**1150**	0.02
Nug16a	0.00000000	1610	=	**1610**	0.12
Nug16b	0.00000000	1240	=	**1240**	0.01
Nug17	0.00000000	1732	=	**1732**	0.04
Nug18	0.00000000	1930	=	**1930**	1.04
Nug20	0.00000000	2570	=	**2570**	0.10
Nug21	0.00000000	2438	=	**2438**	0.04
Nug22	0.00000000	3596	=	**3596**	0.75
Nug24	0.00000000	3488	=	**3488**	1.16
Nug25	0.00000000	3744	=	**3744**	2.60
Nug27	0.00000000	5234	=	**5234**	8.39
Nug28	0.00000000	5166	=	**5166**	5.71
Nug30	0.00163292	6124	¡	6134	4.59
Rou12	0.00000000	235528	=	**235528**	0.05
Rou15	0.00000000	354210	=	**354210**	0.18
Rou20	0.00000000	725522	=	**725522**	1.38
Scr12	0.00000000	31410	=	**31410**	0.01
Scr15	0.00000000	51140	=	**51140**	0.08
Scr20	0.00000000	110030	=	**110030**	0.13
Sko42	0.00695674	15812	¡	15922	10.60
Sko49	0.00607201	23386	¡	23528	13.58
Sko56	0.00551396	34458	¡	34648	5.26
Sko64	0.00964988	48498	¡	48966	11.25
Sko72	0.00926709	66256	¡	66870	8.19
Sko81	0.00806611	90998	¡	91732	9.99
Sko90	0.00834386	115534	¡	116498	24.58
Sko100a	0.00878936	152002	¡	153338	20.96
Sko100b	0.00882449	153890	¡	155248	2.43
Sko100c	0.00914366	147862	¡	149214	37.66
Sko100d	0.00974755	149576	¡	151034	8.49

Prob.	Dif./QAPLib	QAPLib		Íris	Tempo (seg)
Sko100e	0.00821991	149150	¡	150376	8.00
Sko100f	0.01019888	149036	¡	150556	2.00
Ste36a	0.02162503	9526	¡	9732	26.54
Ste36b	0.01488771	15852	¡	16088	39.11
Ste36c	0.00938961	8239110	¡	8316472	10.69
Tai12a	0.00000000	224416	=	**224416**	0.01
Tai12b	0.00000000	39464925	=	**39464925**	0.05
Tai15a	0.00000000	388214	=	**388214**	0.10
Tai15b	0.00000000	51765268	=	**51765268**	0.02
Tai17a	0.00000000	491812	=	**491812**	1.86
Tai20a	0.00469664	703482	¡	706786	1.36
Tai20b	0.00000000	122455319	=	**122455319**	0.05
Tai25a	0.00881041	1167256	¡	1177540	9.76
Tai25b	0.00000000	344355646	=	**344355646**	16.29
Tai30a	0.01061411	1818146	¡	1837444	14.68
Tai30b	0.00090536	637117113	¡	637693934	27.48
Tai35a	0.01658958	2422002	¡	2462182	48.05
Tai35b	0.00220667	283315445	¡	283940628	1.01
Tai40a	0.02116985	3139370	¡	3205830	7.24
Tai40b	0.00005117	637250948	¡	637283558	55.45
Tai50a	0.02688064	4938796	¡	5071554	6.78
Tai50b	0.00116554	458821517	¡	459356290	49.16
Tai60a	0.02751083	7205962	¡	7404204	34.67
Tai60b	0.00445869	608215054	¡	610926898	4.42
Tai64c	0.00000000	1855928	=	**1855928**	0.04
Tai80a	0.02813652	13499184	¡	13879004	28.12
Tai80b	0.01501960	818415043	¡	830707311	33.59
Tai100a	0.02651414	21052466	¡	21610654	21.01
Tai100b	0.01491213	1185996137	¡	1203681867	0.63
Tai150b	0.01239791	498896643	¡	505081918	21.79
Tai256c	0.00317775	44759294	¡	44901528	19.81
Tho30	0.00336143	149936	¡	150440	30.87
Tho40	0.00646942	240516	¡	242072	22.77
Tho150	0.01263875	8133398	¡	8236194	15.79
Wil50	0.00192560	48816	¡	48910	10.69
Wil100	0.00600649	273038	¡	274678	1.61

A.2 Metaheurísticas: GRASP e GRASP-PR

Prob.	QAPLib	GRASP	T.(seg)	GRASP-PR	T.(seg)
Bur26a	5426670	**5426670**	0.17	**5426670**	1.83
Bur26b	3817852	**3817852**	0.30	**3817852**	0.56
Bur26c	5426795	**5426795**	0.24	**5426795**	0.27
Bur26d	3821225	**3821225**	1.44	**3821225**	3.43
Bur26e	5386879	**5386879**	4.25	**5386879**	5.29
Bur26f	3782044	**3782044**	1.86	**3782044**	4.19
Bur26g	10117172	10117393	0.92	10118177	1.48
Bur26h	7098658	**7098658**	0.09	**7098658**	0.79
Chr12a	9552	**9552**	0.05	**9552**	0.09
Chr12b	9742	**9742**	0.00	**9742**	0.02
Chr12c	11156	**11156**	0.11	**11156**	0.04
Chr15a	9896	**9896**	4.70	**9896**	1.23
Chr15b	7990	**7990**	0.26	**7990**	0.49
Chr15c	9504	**9504**	0.57	9780	0.53
Chr18a	11098	11142	2.72	**11098**	2.60
Chr18b	1534	**1534**	0.10	**1534**	0.39
Chr20a	2192	2224	4.81	2290	1.07
Chr20b	2298	2434	3.36	2460	11.81
Chr20c	14142	**14142**	0.39	14810	0.04
Chr22a	6156	6244	6.31	6280	3.25
Chr22b	6194	6364	4.75	6390	4.33
Chr25a	3796	3972	6.91	4296	10.83
Els19	17212548	**17212548**	0.19	**17212548**	0.08
Esc16a	68	**68**	0.01	**68**	0.01
Esc16b	292	**292**	0.00	**292**	0.21
Esc16c	160	**160**	0.00	**160**	0.01
Esc16d	16	**16**	0.00	**16**	0.01
Esc16e	28	**28**	0.00	**28**	0.02
Esc16f	0	**0**	0.00	**0**	0.01
Esc16g	26	**26**	0.00	**26**	0.01
Esc16h	996	**996**	0.00	**996**	0.01
Esc16i	14	**14**	0.00	**14**	0.02
Esc16j	8	**8**	0.00	**8**	0.01
Esc32a	130	138	2.56	136	0.15
Esc32b	168	**168**	1.09	**168**	1.66
Esc32c	642	**642**	0.03	**642**	0.08
Esc32d	200	**200**	0.06	**200**	0.09
Esc32e	2	**2**	0.01	**2**	0.07
Esc32g	6	**6**	0.01	**6**	0.20
Esc32h	438	**438**	1.30	**438**	0.18
Esc64a	116	**116**	0.33	**116**	0.73
Esc128	64	**64**	5.41	68	12.97
Had12	1652	**1652**	0.03	**1652**	0.01
Had14	2724	**2724**	0.01	**2724**	0.03
Had16	3720	**3720**	0.00	**3720**	0.08
Had18	5358	**5358**	0.09	**5358**	0.22
Had20	6922	**6922**	0.29	**6922**	0.15
Kra30a	88900	90800	3.75	89900	7.26
Kra30b	91420	91490	1.45	91900	3.26

Prob.	QAPLib	GRASP	T.(seg)	GRASP-PR	T.(seg)
Kra32	88700	90020	3.17	88900	13.51
Lipa20a	3683	**3683**	1.34	**3683**	0.66
Lipa20b	27076	**27076**	0.00	**27076**	0.02
Lipa30a	13178	13256	1.40	13320	4.02
Lipa30b	151426	**151426**	0.00	**151426**	0.07
Lipa40a	31538	31917	2.43	31931	3.42
Lipa40b	476581	**476581**	0.00	**476581**	0.16
Lipa50a	62093	62763	6.85	62766	9.79
Lipa50b	1210244	**1210244**	0.00	**1210244**	0.37
Lipa60a	107218	108235	0.37	108247	4.94
Lipa60b	2520135	**2520135**	0.00	**2520135**	0.70
Lipa70a	169755	171195	11.17	171183	2.73
Lipa70b	4603200	**4603200**	0.00	**4603200**	1.13
Lipa80a	253195	255131	0.77	255145	2.42
Lipa80b	7763962	**7763962**	0.00	**7763962**	1.92
Lipa90a	360630	363184	0.55	363123	11.76
Lipa90b	12490441	**12490441**	0.01	**12490441**	2.80
Nug12	578	**578**	0.06	**578**	0.07
Nug14	1014	**1014**	0.27	**1014**	0.30
Nug15	1150	**1150**	0.08	**1150**	0.18
Nug16a	1610	**1610**	0.02	**1610**	0.10
Nug16b	1240	**1240**	0.03	**1240**	0.09
Nug17	1732	**1732**	5.07	**1732**	0.86
Nug18	1930	**1930**	0.51	**1930**	1.71
Nug20	2570	**2570**	1.37	**2570**	1.93
Nug21	2438	**2438**	4.98	**2438**	7.53
Nug22	3596	**3596**	0.16	**3596**	1.91
Nug24	3488	**3488**	0.14	**3488**	0.20
Nug25	3744	3748	0.03	**3744**	3.08
Nug27	5234	**5234**	2.00	5268	1.44
Nug28	5166	5182	6.40	5192	4.95
Nug30	6124	**6124**	7.28	6148	4.62
Rou12	235528	**235528**	0.03	**235528**	0.01
Rou15	354210	**354210**	0.08	**354210**	0.43
Rou20	725522	726988	3.64	726812	4.71
Scr12	31410	**31410**	0.01	**31410**	0.03
Scr15	51140	**51140**	0.00	**51140**	0.02
Scr20	110030	**110030**	2.41	**110030**	0.23
Sko42	15812	16020	1.35	15926	6.10
Sko49	23386	23612	1.11	23660	1.13
Sko56	34458	34858	0.59	34720	4.20
Sko64	48498	49076	4.62	49238	4.13
Sko72	66256	67050	0.28	67026	7.54
Sko81	90998	91564	0.44	92088	2.96
Sko90	115534	117032	5.13	117092	3.19
Sko100a	152002	154010	0.89	154048	4.18
Sko100b	153890	155854	3.72	155628	6.52
Sko100c	147862	149912	4.56	149852	10.50
Sko100d	149576	151158	2.56	151076	5.85
Sko100e	149150	151648	5.07	151110	14.57
Sko100f	149036	150646	3.00	151090	9.77
Ste36a	9526	9730	3.25	9694	3.82

Prob.	QAPLib	GRASP	T.(seg)	GRASP-PR	T.(seg)
Ste36b	15852	15892	3.05	15892	1.73
Ste36c	8239110	8383806	0.03	8383628	0.58
Tai12a	224416	**224416**	0.01	**224416**	0.03
Tai12b	39464925	**39464925**	0.02	**39464925**	0.01
Tai15a	388214	**388214**	0.13	388870	1.74
Tai15b	51765268	**51765268**	0.03	**51765268**	0.10
Tai17a	491812	**491812**	0.03	**491812**	1.03
Tai20a	703482	710898	1.00	706786	0.17
Tai20b	122455319	**122455319**	0.06	**122455319**	1.70
Tai25a	1167256	1183496	0.14	1184522	0.78
Tai25b	344355646	344653810	0.29	344855160	0.27
Tai30a	1818146	1842410	2.54	1855038	3.04
Tai30b	637117113	637925537	0.84	638868135	4.16
Tai35a	2422002	2485804	1.08	2476346	7.68
Tai35b	283315445	284522955	5.71	284156225	2.11
Tai40a	3139370	3229282	6.28	3201386	4.61
Tai40b	637250948	637558686	3.61	649348115	1.01
Tai50a	4938796	5100762	1.68	5074786	2.88
Tai50b	458821517	462218184	8.04	461789836	2.16
Tai60a	7205962	7428158	3.41	7443952	8.21
Tai60b	608215054	609372882	7.16	611948714	5.63
Tai64c	1855928	**1855928**	0.71	**1855928**	0.86
Tai80a	13499184	13930424	10.02	13881992	6.04
Tai80b	818415043	843813730	2.62	838174876	4.99
Tai100a	21052466	21651484	1.79	21617136	17.12
Tai100b	1185996137	1212162894	4.58	1207991741	7.61
Tai150b	498896643	509782023	5.97	508958403	23.22
Tai256c	44759294	98685678	0.02	46202282	111.27
Tho30	149936	**149936**	5.29	151178	5.49
Tho40	240516	242530	14.23	241794	5.05
Tho150	8133398	8277830	2.04	8229022	21.21
Wil50	48816	48976	11.27	49052	6.42
Wil100	273038	274710	0.44	274204	11.58

A.3 Metaheurísticas: GRASP, Busca Tabu, FANT

Prob.	QAPLib	GRASP	T.(seg)	Tabu	T.(seg)	FANT	T.(seg)
Bur26a	5426670	**5426670**	14.32	5429780	7.65	**5426670**	12.62
Bur26b	3817852	**3817852**	14.12	3817902	8.00	**3817852**	12.92
Bur26c	5426795	**5426795**	14.05	5427199	7.66	**5426795**	12.71
Bur26d	3821225	**3821225**	14.31	3821515	7.80	**3821225**	12.88
Bur26e	5386879	**5386879**	14.50	5387118	7.82	**5386879**	12.82
Bur26f	3782044	**3782044**	14.03	3782479	7.78	**3782044**	12.77
Bur26g	10117172	**10117172**	14.79	10117903	7.79	**10117172**	12.83
Bur26h	7098658	**7098658**	14.26	7099530	7.76	**7098658**	12.78
Chr12a	9552	**9552**	0.88	**9552**	1.44	**9552**	1.26
Chr12b	9742	**9742**	0.83	**9742**	1.44	**9742**	1.26
Chr12c	11156	**11156**	0.90	**11156**	1.47	**11156**	1.27
Chr15a	9896	9936	1.90	10119	2.40	9916	2.47
Chr15b	7990	**7990**	1.79	8583	2.42	**7990**	2.47
Chr15c	9504	9811	1.82	10435	2.37	**9504**	2.48
Chr18a	11098	11130	3.40	13535	3.55	**11098**	4.24
Chr18b	1534	**1534**	3.24	1541	3.54	**1534**	4.24
Chr20a	2192	**2192**	4.90	2804	4.42	2194	5.78
Chr20b	2298	2443	4.84	2688	4.43	2411	5.79
Chr20c	14142	**14142**	4.53	16195	4.44	**14142**	5.84
Chr22a	6156	6297	6.73	6377	5.46	6198	7.73
Chr22b	6194	6362	6.99	6565	5.51	6276	7.80
Chr25a	3796	4295	10.99	4347	7.14	3933	11.46
Els19	17212548	**17212548**	4.75	17304919	4.03	**17212548**	5.19
Esc16a	68	**68**	1.87	**68**	2.84	**68**	3.06
Esc16b	292	**292**	1.90	**292**	2.81	**292**	2.94
Esc16c	160	**160**	1.98	**160**	2.78	**160**	3.02
Esc16d	16	**16**	1.96	**16**	2.76	**16**	3.01
Esc16e	28	**28**	1.81	**28**	2.79	**28**	3.00
Esc16f	0	**0**	1.59	**0**	0.01	**0**	1.57
Esc16g	26	**26**	1.83	**26**	2.74	**26**	2.97
Esc16h	996	**996**	2.03	**996**	2.73	**996**	2.96
Esc16i	14	**14**	1.72	**14**	2.73	**14**	2.96
Esc16j	8	**8**	1.77	**8**	2.73	**8**	2.94
Esc32a	130	133	21.83	141	12.11	**130**	23.85
Esc32b	168	**168**	22.34	**168**	12.05	**168**	23.91
Esc32c	642	**642**	21.57	**642**	12.06	**642**	23.86
Esc32d	200	**200**	20.72	**200**	12.03	**200**	23.74
Esc32e	2	**2**	18.08	**2**	12.02	**2**	23.44
Esc32g	6	**6**	18.33	**6**	12.26	**6**	23.65
Esc32h	438	**438**	21.98	**438**	12.00	**438**	23.96
Esc64a	116	**116**	258.85	**116**	51.88	**116**	191.69
Esc128	64	**64**	3490.06	**64**	206.86	**64**	1519.85
Had12	1652	**1652**	0.83	**1652**	1.43	**1652**	1.25
Had14	2724	**2724**	1.46	**2724**	2.03	**2724**	1.99
Had16	3720	**3720**	2.33	**3720**	2.73	**3720**	2.97
Had18	5358	**5358**	3.51	**5358**	3.55	**5358**	4.27
Had20	6922	**6922**	5.14	**6922**	4.44	**6922**	5.80

Prob.	QAPLib	GRASP	T.(seg)	Tabu	T.(seg)	FANT	T.(seg)
Kra30a	88900	90250	20.89	92080	10.49	**88900**	19.65
Kra30b	91420	91535	20.82	92650	10.54	**91420**	19.67
Kra32	88700	89050	26.08	91640	12.11	**88700**	23.89
Lipa20a	3683	**3683**	5.17	3755	4.44	**3683**	5.88
Lipa20b	27076	**27076**	5.61	**27076**	4.39	**27076**	5.95
Lipa30a	13178	**13178**	22.54	13391	10.62	**13178**	19.81
Lipa30b	151426	**151426**	25.21	**151426**	10.55	**151426**	19.75
Lipa40a	31538	31901	62.59	31819	19.27	31709	46.81
Lipa40b	476581	**476581**	70.38	558217	19.31	**476581**	46.93
Lipa50a	62093	62707	141.20	62789	30.62	62383	91.31
Lipa50b	1210244	**1210244**	159.97	1421219	30.61	**1210244**	91.38
Lipa60a	107218	108140	277.17	108195	44.45	108101	157.63
Lipa60b	2520135	**2520135**	311.69	3005575	44.36	**2520135**	157.76
Lipa70a	169755	171063	489.86	171195	61.10	170998	249.81
Lipa70b	4603200	**4603200**	549.50	5517936	60.46	**4603200**	248.95
Lipa80a	253195	254967	797.10	255089	79.20	254943	372.43
Lipa80b	7763962	**7763962**	895.05	9389545	79.13	**7763962**	371.38
Lipa90a	360630	362950	1230.11	363146	100.99	362970	530.50
Lipa90b	12490441	**12490441**	1377.29	15115774	102.32	13799684	529.84
Nug12	578	**578**	0.83	**578**	1.44	**578**	1.26
Nug14	1014	**1014**	1.42	1015	2.07	**1014**	2.00
Nug15	1150	**1150**	1.79	**1150**	2.37	**1150**	2.43
Nug16a	1610	**1610**	2.29	**1610**	2.74	**1610**	2.99
Nug16b	1240	**1240**	2.30	1246	2.77	**1240**	2.99
Nug17	1732	**1732**	2.87	1734	3.14	**1732**	3.55
Nug18	1930	**1930**	3.53	1938	3.53	**1930**	4.23
Nug20	2570	**2570**	5.25	2588	4.44	**2570**	5.80
Nug21	2438	**2438**	6.35	2445	4.91	**2438**	6.73
Nug22	3596	**3596**	7.61	3608	5.46	**3596**	7.72
Nug24	3488	**3488**	10.01	3532	6.57	**3488**	10.06
Nug25	3744	**3744**	11.71	3762	7.15	**3744**	11.41
Nug27	5234	**5234**	15.51	5283	8.44	**5234**	14.27
Nug28	5166	5174	17.41	5220	9.08	**5166**	15.99
Nug30	6124	6131	22.47	6176	10.48	6126	19.70
Rou12	235528	**235528**	0.88	**235528**	1.44	**235528**	1.27
Rou15	354210	**354210**	2.07	355432	2.38	**354210**	2.47
Rou20	725522	727033	5.72	738517	4.46	726157	5.79
Scr12	31410	**31410**	0.84	**31410**	1.45	**31410**	1.26
Scr15	51140	**51140**	1.90	**51140**	2.40	**51140**	2.46
Scr20	110030	**110030**	5.32	110842	4.41	**110030**	5.81
Sko42	15812	15916	76.17	15964	21.08	15819	53.80
Sko49	23386	23522	134.77	23662	29.08	23418	85.72
Sko56	34458	34680	220.41	34845	38.08	34517	127.58
Sko64	48498	48828	362.04	49083	50.16	48606	190.26
Sko72	66256	66655	559.41	66960	63.84	66452	271.03
Sko81	90998	91627	861.87	91994	81.47	91184	385.89
Sko90	115534	116326	1277.48	116900	100.59	115956	528.51
Sko100a	152002	152855	1883.63	153649	125.74	152552	729.27
Sko100b	153890	154793	1881.49	155649	124.90	154464	728.78
Sko100c	147862	148823	1899.42	149144	125.21	148277	728.36
Sko100d	149576	150564	1897.36	151000	125.66	150118	729.02