UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Uma Implementação Paralela Híbrida para o Problema do Caixeiro Viajante Usando Algoritmos Genéticos, GRASP e Aprendizagem por Reforço

João Paulo Queiroz dos Santos

Orientador: Prof. Dr. Jorge Dantas de Melo

Co-orientador: Prof. Dr. Adrião Duarte Dória Neto

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da UFRN (área de concentração: Engenharia de Computação) como parte dos requisitos para obtenção do título de Mestre em Ciências.

Número de ordem PPgEE: M227
Natal, RN, março de 2009

Divisão de Serviços Técnicos

Catalogação da publicação na fonte. UFRN / Biblioteca Central Zila Mamede

Santos, João Paulo Queiroz dos.

Uma implementação paralela híbrida para o problema do caixeiro viajante usando algoritmos genéticos, GRASP e aprendizagem por reforço / João Paulo Queiroz dos Santos. – Natal, RN, 2009

59 f.

Orientador: Jorge Dantas de Melo

Co-orientador: Adrião Duarte Dória Neto

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Programa de Pós-Graduação em Engenharia Elétrica.

1. Metaheurísticas GRASP – Dissertação. 2. Algoritmos genéticos – Dissertação. 3. Q-learning – Dissertação. 4. Sistemas paralelos e distribuídos – Dissertação. I. Melo, Jorge Dantas de. II. Dória Neto, Adrião Duarte. III. Universidade Federal do Rio Grande do Norte. IV. Título.

RN/UF/BCZM CDU 004.023(043.3)

Uma Implementação Paralela Híbrida para o Problema do Caixeiro Viajante Usando Algoritmos Genéticos, GRASP e Aprendizagem por Reforço

João Paulo Queiroz dos Santos

Dissertação de Mestrado aprovada em 06 de março de 2009 pela banca examinadora composta pelos seguintes membros:

Prof. Dr. Jorge Dantas de Melo (orientador) . . . . . DCA/UFRN

Prof. Dr. Adrião Duarte Dória Neto (co-orientador) . . . . . DCA/UFRN

Prof. Dr. - Ing. Manoel Firmino de Medeiros Júnior (examinador) DCA/UFRN

Prof. Dr. Ricardo Alexsandro de Medeiros Valentim (examinador) . . IFRN-RN

A Deus toda honra toda glória.

O único lugar onde êxito vem antes de trabalho é no dicionário.

Agradecimentos

Agradeço primeiramente a Deus, por me conceder o privilégio de galgar este título tão desejado.

Aos meus pais por toda estrutura e apoio que dispensaram por toda minha vida.

A minha amada, companheira de todas as horas Danielli Helena, por me encorajar e incentivar em todos momentos difíceis desta árdua caminhada, você é o amor da minha vida.

Aos meus familiares de alguma forma (sempre se tem uma forma) ajudaram-me nesta caminhada.

Ao pessoal que fazem parte do Laboratório de Sistemas Inteligentes (LABSI), em especial a Rafael e Lima, por ajudarem a compor este trabalho.

Agradecer em especial aos professores Jorge Dantas e Adrião Dória, pelo apoio dispensado para que este trabalho torna-se possível.

Resumo

As metaheurísticas são técnicas conhecidas para a resolução de problemas de otimização, classificados como NP-Completos e vêm obtendo sucesso em soluções aproximadas de boa qualidade. Elas fazem uso de abordagens não determinísticas que geram soluções que se aproximam do ótimo, mas no entanto, sem a garantia de que se encontre o ótimo global. Motivado pelas dificuldades em torno da resolução destes problemas, este trabalho propôs o desenvolvimento de métodos paralelos híbridos utilizando a aprendizagem por reforço e as metaheurísticas GRASP e Algoritmos Genéticos. Com a utilização dessas técnicas em conjunto, objetivou-se então, contribuir na obtenção de soluções mais eficientes. Neste caso, ao invés de utilizar o algoritmo *Q-learning* da aprendizagem por reforço, apenas como técnica de geração das soluções iniciais das metaheurísticas, este também aplicado de forma cooperativa e competitiva com o Algoritmo Genético e o GRASP, em uma implementação paralela.

Neste contexto, foi possível verificar que as implementações realizadas neste trabalho apresentaram resultados satisfatórios, tanto na parte de cooperação e competição entre os algoritmos *Q-learning*, GRASP a Algoritmos Genéticos, quanto na parte de cooperação e competição entre grupos destes três algoritmos. Em algumas instâncias foi encontrado o ótimo global; quando não encontrado, conseguiu-se chegar bem próximo de seu valor. Neste sentido foi realizada uma análise do desempenho da abordagem proposta e verificou-se um bom comportamento em relação aos quesitos que comprovam a eficiência e o *speedup* (ganho de velocidade com o processamento paralelo) das implementações realizadas.

Palavras-chave:Metaheurísticas GRASP, Algoritmos Genéticos, *Q-learning*, Sistemas Paralelos e Distribuídos

Abstract

The metaheuristics techiniques are known to solve optimization problems classified as NP-complete and are successful in obtaining good quality solutions. They use non-deterministic approaches to generate solutions that are close to the optimal, without the guarantee of finding the global optimum. Motivated by the difficulties in the resolution of these problems, this work proposes the development of parallel hybrid methods using the reinforcement learning, the metaheuristics GRASP and Genetic Algorithms. With the use of these techniques, we aim to contribute to improved efficiency in obtaining efficient solutions. In this case, instead of using the *Q-learning* algorithm by reinforcement learning, just as a technique for generating the initial solutions of metaheuristics, we use it in a cooperative and competitive approach with the Genetic Algorithm and GRASP, in an parallel implementation.

In this context, was possible to verify that the implementations in this study showed satisfactory results, in both strategies, that is, in cooperation and competition between them and the cooperation and competition between groups. In some instances were found the global optimum, in others theses implementations reach close to it. In this sense was an analyze of the performance for this proposed approach was done and it shows a good performance on the requeriments that prove the efficiency and *speedup* (gain in speed with the parallel processing) of the implementations performed.

Index terms:GRASP Metaheuristics, Genetic Algorithm, *Q-learning*, Parallel and Distributed Systems

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Símbolos e Abreviaturas

α	Coeficiente de aprendizagem
α_{LRC}	paramêtro de escolha comprimento da LRC
ε	Critério de diversificação ou intensificação
γ	Fator de desconto
AG	Algoritmos Genéticos
AR	Aprendizagem por Reforço
BT	Busca Tabu
CED	*Constrained Economic Dispatch*
dist.	distância
DT	Diferenças Temporais
F.O.	Função Objetivo
GRASP	*Greedy Randomized Adaptive Search Procedure*
JSS	*Job Shop Scheduling*
LRC	Lista Restrita de Candidatos
MC	Monte Carlo
MPI	*Message Passing Interface*
NP	Não Polinomial
PCV	Problema do Caixeiro Viajante
PD	Programação Dinâmica

PDM Processo de Decisão Markoviano

prox. proximidade

QV Quantização Vetorial

TSP *Traveling Salesman Problem*

TSPLIB Biblioteca de instâncias para o Problema do Caixeiro Viajante

UFRN Universidade Federal do Rio Grande de Norte

VNS *Variable Neighborhood Search*

Capítulo 1

Introdução

Modelar e resolver problemas complexos do mundo em que vivemos não é uma tarefa trivial, pois existem inúmeras situações em que é impossível se construir um modelo detalhado para o problema, dada sua elevada complexidade. Por outro lado, um processo de simplificação de tal modelo pode causar perdas de informações relevantes que podem comprometer a sua qualidade. Além da dificuldade inerente à construção de modelos para tais problemas, uma característica que os acompanha durante a fase de resolução é a necessidade de processamento computacional de grande porte, o que, na maioria das vezes, leva tais problemas a serem considerados intratáveis. Nesse contexto, inúmeros pesquisadores têm se dedicado ao desenvolvimento de técnicas que visam facilitar a modelagem e, principalmente, a resolução destes problemas [Lima Júnior et al. 2001].

Uma abordagem bastante utilizada na obtenção de soluções para os problemas citados acima tem feito uso de técnicas chamadas metaheurísticas [Sevkli e Aydin 2006], que são estratégias baseadas em procedimentos heurísticos, aplicáveis principalmente aos problemas de otimização e que produzem um processo simplificado de busca estocástica no espaço de solução. Entretanto, este processo simplificado de busca não garante que a solução encontrada seja a solução ótima do problema. Diferentes problemas de grande importância prática têm sido resolvidos com sucesso, dentre os quais, pode-se citar:

- Roteamento para atendimento médico [Pacheco et al. 2008]
- Sequenciamento de DNA [Blazewicz et al. 2004]
- Matriz energética [Eghbal et al. 2007]
- Engenharia de Software [Clarke et al. 2003]
- Projetos de chips VLSI [Faroe et al. 2001]

Basicamente, uma metaheurística é constituída de duas etapas, a saber: geração aleatória de uma ou mais soluções no espaço de busca (etapa de diversificação da busca) e

na melhoria desta(s) solução(ões) usando um procedimento de busca local (etapa de intensificação da busca). A maneira como é(são) gerada(s) a(s) solução(ões) incial(is), os procedimentos de busca local utilizados e a alternância entre as etapas de intensificação e diversificação estabelecem as diferenças entre as diversas metaheurísticas. A diversificação é utilizada para permitir a fuga da solução dos chamados mínimos locais, enquanto a intensificação é utilizada para melhorar a qualidade da solução localmente, em busca do ótimo global.

A qualidade das soluções geradas durante a fase de diversificação está diretamente ligada à qualidade das soluções obtidas após a fase de intensificação. Para problemas complexos, esta qualidade da solução inicial nem sempre é fácil de ser obtida, levando assim à constante investigação de novas abordagens para garantir este aspecto das metaheurísticas. No grupo de pesquisas em Sistemas Inteligentes da UFRN vem sendo investigada a possibilidade da utilização de técnicas de aprendizagem de máquina para melhorar a qualidade das soluções iniciais [Lima Júnior et al. 2001]. Os trabalhos têm investigado a utilização da aprendizagem por reforço [Sutton e Barto 1998] como alternativa para o problema da qualidade nas soluções geradas nas metaheurísticas. Trata-se de uma técnica de aprendizagem baseada na interação entre agente e ambiente, onde percepções do ambiente são transformadas em ações, sendo estas recompensadas ou punidas de acordo com as suas conseqüências sobre o ambiente. São normalmente utilizadas na solução de problemas de decisão markovianos, embora possam ser adaptadas para uma vasta classe de problemas de otimização.

A grande demanda de processamento referida anteriormente é outro aspecto a ser considerado. Uma ação possível para atender essa demanda de processamento é a utilização de arquiteturas dotadas de capacidade de processamento paralelo, capazes de aumentar em algumas ordens de grandeza a potência de processamento disponível nas arquiteturas monoprocessadas usuais. A utilização do processamento paralelo impõe o desenvolvimento de novos algoritmos e abre possibilidades para a exploração de aspectos do problema não abordáveis nas arquiteturas usuais, tais como, concorrência, cooperação e competição [Foster 1995].

1.1 Motivação

Com a complexidade dos problemas do mundo real e, motivado pelas dificuldades em torno da resolução destes problemas, além do êxito conseguido pelas técnicas já citadas, este trabalho propôs o desenvolvimento de métodos paralelos híbridos, utilizando a aprendizagem por reforço e as metaheurísticas GRASP e Algoritmos Genéticos. Com a

utilização dessas técnicas em conjunto, buscou-se contribuir para uma melhor eficiência na obtenção das soluções de problemas de otimização combinatória. Nesse caso, ao invés de utilizar o algoritmo *Q-learning* da aprendizagem por reforço, apenas como técnica de geração das soluções iniciais das metaheurísticas, analizar sua utilização de forma cooperativa e competitiva com o Algoritmo Genético e o GRASP, em uma implementação paralela.

1.2 Objetivos

Como em toda abordagem para resolução de problemas, as metaheurísticas possuem suas dificuldades, que estão diretamente ligadas ao dilema intensificação ou diversificação (ou *exploration/exploitation*): trata-se da busca de equilíbrio entre intensificar a busca local ou explorar o espaço de soluções em busca de novas opções. Sem uma boa apreciação deste dilema o algoritmo pode facilmente ficar preso um mínimo local. Sendo assim, este trabalho tem como objetivos:

- Estudar as potencialidades do uso conjunto do algoritmo *Q-learning* da aprendizagem por reforço com as metaheurísticas GRASP e Algoritmo Genético para resolução de problemas de otimização combinatória, buscando estabelecer um bom compromisso entre as etapas de intensificação e diversificação.
- Realizar uma implementação paralela de forma cooperativa e competitiva das três técnicas citadas e analisar sua(s) contribuição(ões) na aceleração e na qualidade da solução do problema.
- Utilizar o Problema do Caixeiro Viajante simétrico como *benchmark* para testar o desempenho da implementação paralela.

1.3 Estado da arte

Uma limitação encontrada nos procedimentos de busca local é a sua complexidade exponencial. O problema pode ser formulado da seguinte forma: dada uma instância de um problema de busca local e uma solução inicial s, existe uma solução ótima local s' que pode ser obtida a partir de s em um número polinomial de passos de busca local ?. Em Fischer (1994), o autor estabelece que a solução desta questão está em NP[1] (Não Polinomial). Logo, o uso do processamento paralelo pode ser uma alternativa interessante para

[1]De acordo com Wikipédia(http://pt.wikipedia.org/wiki/NP_(complexidade)), um problema é de classe NP (do inglês *Non-Deterministic Polynomial time*) se sua solução pode ser verificada em um tempo polinomial por uma máquina de Turing não-determinística

amenizar esta dificuldade e permitir um aumento significativo no tamanho dos problemas tratados. Deve-se observar ainda que há algum tempo, o uso de processamento paralelo em metaheurísticas vem sendo estudado [Duni et al. 2002].

Os algoritmos utilizados podem ser classificados de acordo com o percurso realizado no espaço de busca (simples ou múltiplo) e no mecanismo de execução (síncrono ou assíncrono). Em algoritmos de percurso simples, apenas um caminho no espaço de busca é explorado com a execução simultânea de várias das etapas da busca local. Em algoritmos de percursos múltiplos várias soluções são avaliadas de forma simultânea, seja de forma iterativa ou de forma independente. Em algoritmos paralelos de execução síncrona, uma ou mais etapas do algoritmo são efetuadas no mesmo instante por todas as tarefas, enquanto na execução assíncrona cada tarefa é processada de forma independente.

Neste trabalho foram estudados os algoritmos de busca de percursos múltiplos assíncronos baseados nas metaheurísticas GRASP e Algoritmos Genéticos e na Aprendizagem por Reforço. Como serão utilizados vários algoritmos diferentes em paralelo, optou-se por denominar a aplicação resultante de *híbrida*. O termo híbrido tem sido usado em outros contextos, seja porque o algoritmo sequencial paralelizado já tem essa denominação de híbrido [Aziz e Boussakta 2000], seja pelo uso de diferentes objetivos na busca paralela [Berger e Barkaoui 2004]. Os algoritmos paralelos baseados nos percursos múltiplos também são chamados algoritmos *cooperativos* [Toulouse et al. 2004] e não seguem a mesma concepção dos algoritmos paralelos baseados em decomposição dos dados ou decomposição funcional clássicos [Foster 1995].

Seguindo a abordagem cooperativa, podem-se citar os trabalhos de Ribeiro e Rosseti (2007) e Pardalos et al. (1995) que analisam diferentes implementações da metaheurística GRASP e os trabalhos de Albuquerque et al. (2004) e Gordon e Whitley (1993) sobre implementações paralelas de Algoritmos Genéticos. No que diz respeito à aprendizagem por reforço, as implementações paralelas podem ser divididas em duas categorias: os problemas de projeto de agentes únicos e os problemas de projeto multi-agentes. O caso tratado neste trabalho é de projeto de agente único, voltado para a solução do problema do caixeiro viajante e neste contexto pode-se citar os trabalhos de Kushida et al. (2006), que explora a decomposição do espaço dos estados e Grounds e Kudenko (2007) que utilizam um aprendizado cooperativo para o projeto do agente.

1.4 Organização do trabalho

Este trabalho está organizado da seguinte forma: no capítulo 2, apresenta-se uma descrição de alguns conceitos de otimização combinatória e uma descrição do problema

do caixeiro viajante. No capítulo 3, são tratados os conceitos básicos das metaheurísticas GRASP e Algoritmo Genético, enquanto que no capítulo 4, abordam-se conceitos sobre processos de decisão markovianos e apredizagem por reforço, com enfoque no algoritmo *Q-learning*. No capítulo 5, apresenta-se a abordagem paralela cooperativa e competitiva e, no capítulo 6, os resultados dos experimentos feitos na execução do método proposto. No capítulo 7 são discorridas as conclusões relacionadas à dissertação desenvolvida.

Capítulo 2

Otimização Combinatória e o Problema do Caixeiro Viajante

Neste capítulo apresentam-se conceitos gerais sobre otimização combinatória e uma descrição do problema a ser tratado, a saber, o problema do caixeiro viajante.

2.1 Otimização Combinatória

O conceito de otimização combinatória teve seu surgimento com a concepção da pesquisa operacional em torno da II Guerra Mundial. O objetivo inicial da pesquisa operacional era a melhoria das estratégias militares. Entretanto, no período pós-guerra, ela foi difundida nas empresas e indústrias americanas. A otimização combinatória tem como objetivo maximizar ou minimizar uma função definida sobre certo domínio finito. Em termos formais, considerando-se um problema de minimização P, este pode ser estabelecido como: dado um conjunto finito $E = \{1, 2, ..., n\}$, o conjunto de soluções possíveis $F \subseteq 2^E$ *de* P e uma função objetivo $f : 2^E \rightarrow \mathbb{R}$, deseja-se encontrar uma solução ótima $S^* \in F$ tal que $f(S^*) \leq f(S)$, $\forall\, S \in F$, onde $f(S)$ é a função objetivo. Para um problema específico, os conjuntos E e F, bem como a função $f(S)$, precisam ser definidas apropriadamente.

Para estes problemas, algoritmos de busca exatos, tais como separação-e-avalição (*branch-and-bound*) e programação dinâmica podem resultar numa quantidade exaustiva de enumerações. Devido a isto, tais algoritmos são apropriados apenas para instâncias do problema de tamanho moderado, face ao crescimento exponencial do número de elementos de F em função do aumento de n. Por isso, utilizam-se normalmente algoritmos de busca baseados em heurísticas para encontrar soluções (não necessariamente ótimas) para estes problemas. As heurísticas permitem a exploração de uma região reduzida do espaço de soluções, onde se presume, com base na heurística, que boas soluções podem

ser encontradas.

Os problemas abaixo são exemplos de otimização combinatória:

- Problema do Caixeiro Viajante;
- Roteamento de Veículos;
- Alocação de Salas em Escolas (*Timetabling*);
- Escalonamento de Tarefas em Máquinas;
- Localização de Facilidades;
- Projeto de Circuitos Integrados, dentre outros

Existem pelo menos duas abordagens distintas para se resolver problemas de otimização combinatória, as quais serão descritas nas subseções a seguir:

2.1.1 Abordagem exata

Segundo Bernardi (2001), algoritmos exatos em otimização combinatória são algoritmos enumerativos, que varrem o espaço de soluções e buscam aquela que apresenta o melhor valor para a função objetivo. Estes algoritmos distinguem-se quanto a abrangência em:

- **Enumeração explícita:** onde todo o espaço de soluções é examinado. Devido à característica de explosão combinatória dos problemas, esta abordagem só pode ser aplicada na resolução de problemas de instâncias pequenas.
- **Enumeração implícita:** onde apenas parte do espaço de soluções é examinado. Tal espaço é particionado de modo a eliminar inúmeras soluções possíveis, através da utilização de limites (*bounds*).

São exemplos de abordagens exatas para problemas de otimização combinatória:

- **Programação dinâmica:** A programação dinâmica é uma técnica de solução recursiva de problemas, baseada no fato que alguns problemas podem ser decompostos em sub-problemas mais simples e cuja solução pode ser obtida de forma recursiva a partir das soluções dos sub-problemas iniciais [Bellman 2003].
- **Uso de relaxações:** Esta técnica consiste em resolver uma seqüência de problemas "*fáceis*" obtidos a partir de relaxações do problema original. Com a solução de tais problemas, procura-se obter limites para o valor da solução ótima de P [Espejo e Galvão 2002]. São exemplos de relaxações:
  - Relaxação *lagrangeana*

  - Relaxação *surrogate*
  - Relaxação *lagrangeana-surrogate*

- ***Branch-and-Bound*:** É um método que busca a solução ótima realizando operações de ramificação, chamadas *branching*, que dividem um problema em dois subproblemas similares. Cada sub-problema trabalha com uma sub-região diferente do espaço de solução. A resolução dos subproblemas também inclui operações de ramificação (ou seja, o procedimento é repetido recursivamente). Se o espaço de soluções é muito grande, a busca pela solução se torna inviável devido ao tempo gasto para executar o algoritmo. Este problema é tratado utilizando *bounding*, que é um modo de encontrar limites para a solução ótima dentro de uma região viável [Balas e Toth 1985].

2.1.2 Abordagem aproximada

Os algoritmos aproximados tornam-se viáveis em função da quantidade de soluções do problema que cresce exponencialmente com sua dimensão. Dentre as diferentes abordagens heurísticas para a obtenção de soluções aproximadas, podem-se citar:

- **Algoritmos probabilísticos:** são algoritmos que se caracterizam pelo fato de que, no momento de tomada de decisão, escolhe-se seguir um caminho de ação aleatório ao invés de escolher uma dentre um conjunto de alternativas prévias. Ao contrário dos algoritmos determínisticos, os algoritmos probabilísticos podem conseguir soluções diversas, para os mesmos dados de caracterização do problema, quando executados várias vezes. [Prado 2005].
- **Metaheurísticas:** São métodos projetados tendo como base as características das soluções ou as propriedades do problema e possuem uma complexidade bem menor em relação aos algoritmos exatos. De forma geral, as metaheurísticas produzem soluções viáveis de boa ou excelente qualidade, mas isso sem a garantia do ótimo. As Metaheurísticas são estratégias de alto nível que guiam heuristicas mais específicas, a fim de encontrar soluções viáveis para problemas específicos. No desenvolvimento de uma heurística, esta geralmente é dependente do problema específico e exige a necessidade de adaptações para a utilização em outros problemas [Blum e Roli 2003] e [Milano e Roli 2004]. A seguir, relacionam-se algumas das metaheurísticas que mais se destacam:
  - Algoritmos Genéticos
  - Busca Tabu

- Colônia de Formigas
- GRASP
- *Simulated Annealing*
- Nuvem de Partículas (*Particle Swarm*)

As metaheurísticas que serão utilizadas neste trabalho serão algoritmos genéticos e GRASP, que serão descritas no capítulo seguinte.

2.2 Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (PCV) (do inglês *Traveling Salesman Problem* (TSP)) tem sido largamente estudado nos últimos 50 anos. Neste período, tem servido de *benchmark* para muitos algoritmos de otimização combinatória desde as técnicas clássicas a abordagens baseadas em busca tabu, redes neurais e algoritmos genéticos. No que segue, será feita uma rápida apresentação do problema e das dificuldades para a sua solução. Para maiores detalhes do que será discutido aqui, recomenda-se a leitura de Johnson e McGeoch (2003), que serviu de base para esta seção.

2.2.1 Definição do problema

O problema do caixeiro viajante pode ser definido como segue: tem-se um dado conjunto $c_1, c_2, ..., c_N$ de *cidades* e, para cada par (c_i, c_j) de cidades distintas, uma *distância* $d(c_i, c_j)$. O objetivo é encontrar um ordenamento π das cidades que minimize a quantidade:

$$\sum_{i=1}^{N-1} d(c_{\pi(i)}, c_{\pi(j)}) + d(c_{\pi(N)}, c_{\pi(1)}) \tag{2.1}$$

Esta quantidade é definida como *comprimento do ciclo* e corresponde ao comprimento da volta que o caixeiro irá efetuar ao visitar cada uma das cidades na ordem especificada pela permutação, retornando no final para a cidade inicial. Embora de formulação simples, o PCV tem despertado a atenção dos pesquisadores porque, mesmo após décadas de estudo, ele ainda não foi completamente resolvido. Além disso, ele pode ser aplicado a uma larga classe de problemas de nosso cotidiano.

Com base na sua formulação inicial, diferentes variações do problema têm sido propostas. Neste trabalho, o interesse está centrado na resolução do PCV *simétrico*, caracterizado pelo fato de que $d(c_i, c_j) = d(c_j, c_i)$. Caso esta restrição não seja satisfeita, o

problema é dito *assimétrico*. No PCV *métrico* todas as distâncias são simétricas e satisfazem à desigualdade triangular, ou seja, $d(c_i, c_k) \leq d(c_i, c_j) + d(c_j, c_k)$. Além disso, as cidades correspondem a pontos em um dado espaço e as distâncias são calculadas por uma métrica definida neste espaço. No caso das cidades pertecerem a um plano, a métrica utilizada é a distância euclidiana e o PCV é denominado *euclidiano*.

2.2.2 Complexidade do PCV

Um resultado bem estabelecido na literatura é que o PCV é *NP*-Completo no caso geral e *NP*-Difícil em alguns casos especiais. Nesta seção, estes conceitos serão discutidos para um melhor entendimento do que significam estas afirmações. Para isto, considerar-se-á um *algoritmo* como sendo um conjunto de instruções que, quando executadas em uma ordem específica, irá resolver um certo problema. Um problema é considerado *fácil* se ele pode ser resolvido por um algoritmo que se executa em um tempo polinomial (algoritmo polinomial). Da mesma forma, ele é considerado *difícil* se ele não pode ser resolvido em tempo polinomial. O algoritmo polinomial tem a característica de apresentar o seu tempo de execução limitado por uma função polinomial do tamanho da instância (número de cidades).

A teoria da complexidade divide os algoritmos em classes, que correspondem ao conjunto de todos os algoritmos que podem ser executados em uma certa quantidade de tempo e usando uma certa quantidade de recursos computacionais. Ela normalmente, trata dos problemas de decisão, que correspondem àqueles cuja solução é *sim* ou *não*. Embora possa parecer restritivo, na realidade, todos os problemas podem ser formulados como tal. No caso do PCV, o problema de decisão associado pode ser definido, tendo com base dois algoritmos: PCV_SOLUÇÃO, que possui como entrada uma descrição do PCV e PCV_DECISÃO, chamado por PCV_SOLUÇÃO que retorna uma resposta *sim* se existe um ciclo com comprimento menor que um dado B e *no*, caso contrário. Suponha que PCV_SOLUÇÃO se executa em tempo polinomial sem considerar o tempo de execução de PCV_DECISÃO. Suponha ainda que se conhece o valor do ciclo de comprimento mínimo, dado por C. PCV_SOLUÇÃO resolve o problema tomando duas cidades quaisquer e associando um custo infinito à distância entre elas. Após isto, invoca PCV_DECISÃO perguntando se existe um ciclo de comprimento C. Caso a resposta seja afirmativa, ele elimina a aresta que liga as duas cidades escolhidas e caso a resposta seja negativa, ele inclui a aresta no ciclo de comprimento ótimo. Repetindo este procedimento para todas as arestas existentes entre as cidades ele resolve o problema. Uma vez que existem, no pior caso, $\vartheta(n^2)$ arestas, PCV_SOLUÇÃO irá se executar em tempo polinomial se PCV_DECISÃO

também se executa em tempo polinomial.

As duas classificações formais de problemas de decisão estabelecidas pela teoria da complexidade são:

- Classe P: corresponde à classe de problemas que podem ser resolvidos por algoritmos determinísticos em tempo *P*olinomial. Em termos de problemas de decisão, isto corresponde ao fato de que uma resposta sim (ou não) estará disponível em um tempo polinomial.
- Classe NP: corresponde à classe de problemas que podem ser resolvidos por algoritmos *N***ão**-determinísticos em tempo *P*olinomial.

Uma relação importante entre as classes P e NP é que $P \subseteq NP$, que estabelece um algoritmo determinístico como sendo um caso especial de um algoritmo não-determinístico. Outro aspecto a considerar é que a verificação de pertinência à classe NP não é tão simples quanto a pertinência à classe P. Normalmente, isto é verificado com base em três princípios que se encontram descritos em Zambito (2008).

Um problema B é considerado NP-difícil se, para todos problemas $B' \in NP$ existe uma redução em tempo polinomial para B. Se qualquer problema NP-difícil $X \in NP$ pode ser transformado para B em tempo polinomial tal que uma solução de B também é solução de X, então B é considerado NP-completo. Uma característica importante da classe NP-completo é que caso seja provado que $B \in P$, então $P = NP$, o que significa que todo problema em NP pode ser resolvido em tempo polinomial. Da mesma forma, provando-se que $B \notin P$ nenhum problema NP-completo pode ser resolvido em tempo polinomial no pior caso. Este é um dos fatos que contribuem para a grande popularidade do problema do caixeiro viajante na análise de algoritmos.

2.2.3 Aplicações do PCV

Existem muitas aplicações do PCV a problemas de nossa vida cotidiana. O mais direto deles é o problema de roteamento. Pode ser formulado em sistemas de transporte, percurso de máquinas de ferramenta, *lay-out* de circuitos, etc. Outras aplicações tratam de fabricação de *chips* VLSI e cristalografia de raios-X [Johnson e McGeoch 2003].

Novas aplicações podem ser formuladas utilizado-se múltiplos caixeiros viajantes. Neste caso, cada cidade deve ser visitada apenas uma vez por qualquer um dos m caixeiros de uma dada companhia. Se o caixeiro j é utilizado, deve ser pago um valor d_j a ele. Cada caixeiro realiza um subciclo de tal forma que a combinação de todos eles estabelece que cada cidade seja visitada uma única vez. O problema a ser resolvido é estabelecer quantos

caixeiros devem ser utilizados de tal forma que a distância percorrida seja mínima assim como os custos de utilização.

2.3 Conclusão

Neste capítulo foram apresentados os principais conceitos ligados à otimização combinatória, uma abordagem do problema do caixeiro viajante com uma definição, a complexidade do PCV e aplicações do PCV para problemas do mundo real.

Capítulo 3

Metaheurísticas GRASP e Algoritmos Genéticos

Neste capítulo será apresentada, de forma resumida, uma breve fundamentação teórica que se torna necessária para uma melhor compreensão das metaheurísticas utilizadas na solução do problema apresentado no capítulo anterior.

3.1 Algoritmos Genéticos

Os Algoritmos Genéticos (AG) fazem parte de um gênero de modelos computacionais que têm inspiração na teoria da evolução proposta por Darwin. Com esses algoritmos, podemos modelar uma solução para um problema específico em uma estrutura de dados como a de um cromossomo e aplicar operadores que recombinam estas estruturas preservando informações críticas. O Algoritmo Genético começa a ser implementado com uma população (comumente gerada de forma aleatória) de cromossomos. Estas estruturas são então avaliadas para gerar reproduções de forma que cromossomos que tenham uma melhor aptidão tenham maiores chances de se reproduzirem do que os que representam uma menor aptidão. Segundo Haupt e Haupt (1998) os Algoritmos Genéticos são técnicas de busca e otimização baseadas nos princípios da genética e seleção natural. Os AG permitem a uma população composta de vários indivíduos de evoluir sob determinadas regras de seleção, levando-a a uma situação que maximiza o *fitness* i.e., a uma maior adequação à solução do problema. O método foi desenvolvido por John Holland entre as décadas de 1960 e 1970 e popularizado por seu aluno, David Goldberg, que foi capaz de resolver um problema de controle de transporte em gasodutos. O primeiro trabalho de Holland foi resumido em seu livro *Adaptation in Natural and Artificial System* e constitui-se em uma tentativa de estabelecer um paradigma para os AG. O trabalho de De Jong (1975) mostra o uso dos AG para problemas de otimização e traz o primeiro esforço concreto

para encontrar parâmetros otimizados de execução.

Nos AG são empregados os conceitos de população e reprodução, com os indivíduos da população representando soluções que são avaliadas e selecionadas de acordo com sua aptidão (*fitness*). Essa aptidão é calculada por uma função de avaliação específica de cada problema. Os indivíduos são selecionados para reprodução, que ocorre através de recombinação de partes de cada indivíduo da população, gerando novas soluções, que serão avaliadas e tem-se a repetição do processo durante um número de gerações definido como um parâmetro do algoritmo. A construção de um Algoritmo Genético consiste, basicamente, nas etapas listadas a seguir:

- Geração da população inicial de forma geral é aleatória com n indivíduos. Cada indivíduo é uma codificação apropriada de uma solução do problema. O valor de n é um dos parâmetros do AG que deve ser escolhido com bastante critério. Se o valor for muito pequeno, o algoritmo tende a convergir rapidamente, sem garantia de um bom resultado. Por outro lado, um valor muito alto para n pode comprometer o tempo de execução do AG.
- Avaliação da qualidade de cada solução (indivíduo da população), através de uma função de adequação, normalmente chamada adequação (do inglês: *fitness*). Após esta etapa, os indivíduos da população são ordenados em função de sua adequação.
- Na população avaliada, são aplicados operadores genéticos na ordem abaixo:
  - **Seleção:** Baseado nos valores da função de adequação os indivíduos são selecionados para comporem o conjunto de *pais* da próxima geração. Um método bastante comum para realizar a seleção é a utilização de uma roleta *viciada* ou tendenciosa que possibilita que cada indivíduo seja selecionado em função da sua adequação, com maior probabilidade de escolha para os indivíduos melhor avaliados. Outra forma de seleção é utilizando critérios elitistas, onde os melhores indivíduos são sempre escolhidos, sem necessidade de uso de roleta. A quantidade de indivíduos que serão utilizados como pais e sua presença ou não na geração seguinte são definidos no algoritmo e estão ligados à estratégia de intensificação ou diversificação a ser utilizada.
  - **Cruzamento (*Crossover*):** Permite que seja feita a recombinação de estruturas genéticas de dois indivíduos (pais) da população, criando novos indivíduos diferentes daqueles existentes na geração atual. Para que isto aconteça, o operador escolhe de forma aleatoria dois indivíduos da população dos *pais* e troca partes de seus materiais genéticos, resultando em cromossomos *filhos* que serão inseridos na população em substituição (ou não) a seus *pais*. Para realizar

a operação de cruzamento, são definidos pontos na estrutura do cromossomo para a troca do material genético. O número de pontos de cruzamento e sua localização são parâmetros de projeto, variando de um problema para outro.

– **Mutação:** Como em todas as metaheurísticas, o AG corre o risco de convergir prematuramente e ficar preso a um mínimo local. Uma forma de evitar que isso aconteça é com a utilização do operador mutação, que corresponde a uma pequena perturbação na configuração de alguns cromossomos, permitindo com isto a exploração em outras regiões do espaço de busca do problema. Da mesma forma que para o operador de cruzamento, a mutação necessita de um parâmetro de controle, a taxa de mutação, que deve também ser escolhida criteriosamente.

No algoritmo 3.1 está representado o pseudo-código de um Algoritmo Genético padrão:

```
 1: procedure Genético (cruzamento, mutação, tamPopulação, maxGeração)
 2: geração da população inicial pop com tamPopulação cromossomos
 3: avaliação da adequação de cada cromossomo presente na população pop
 4: for i=0 to maxGeração do
 5:     pop ← seleção (pop)
 6:     Escolha de acordo com o operador de seleção em pop dois cromossomos PAI_1 e
        PAI_2
 7:     if probabilidade cruzamento then
 8:         pop ← Cruzamento (PAI_1, PAI_2);
 9:     end if
10:     if probabilidade mutação then
11:         pop ← Mutação (cromossomo);
12:     end if
13: end for
14: return melhor solução em pop
```

Algoritmo 3.1: Algoritmo Genético padrão

Maiores detalhes sobre os Algoritmos Genéticos podem ser encontrados em Haupt e Haupt (1998) e Melanie (1998).

3.2 GRASP

A *Greedy Randomized Adaptive Search Procedure* (GRASP) é uma metaherística que foi proposta por Feo e Resende (1995). Na visão de Maciel (2005) o GRASP é um processo iterativo multipartida, em que cada iteração consiste de duas fases: construção

e busca local. A fase de construção gera uma solução inicial viável, cuja vizinhança será investigada até que um ótimo local seja encontrado. Este procedimento é repetido por várias iterações e a melhor solução obtida ao longo de todas as iterações é fornecida como resultado.

A cada iteração da fase de construção, que é incremental, um conjunto de elementos candidatos é formado por todos os elementos que ainda não foram incorporados à solução parcial em construção, sem que com isso comprometa a sua viabilidade. Para a seleção do próximo elemento a ser incorporado, realiza-se a avaliação de todos os elementos candidatos estabelecendo uma lista ordenada dos elementos. A avaliação está associada ao aumento ou diminuição incremental na função objetivo devido à incorporação deste elemento na solução em construção. Isto conduz à criação de uma Lista Restrita de Candidatos (LRC) cujo número de elementos é parametrizado pelo algoritmo. Na escolha de um elemento da LRC, utiliza-se um critério aleatório. O ordenamento guloso, a adaptabilidade do tamanho da LRC e o critério de escolha aleatório de seus elementos são diretamente representados na denominação da metaheurística.

Um dos pontos mais importantes na heurística GRASP é a escolha do comprimento da LRC, representado pelo parâmetro α_{LRC}. A influência desta escolha sobre a qualidade da solução construída é estudada em Ribeiro e Resende (2003). Após a construção da solução inicial, utiliza-se um procedimento de busca local para a melhoria da solução, dentro de uma vizinhaça especificada. Existem várias formas de definição desta vizinhança, dependendo do problema de otimização que se pretende resolver. Para o problema do caixeiro viajante, normalmente são utilizados critérios baseados em *trocas*, tal como $2-opt$ e $3-opt$: partindo-se da solução corrente, a cada etapa de busca local analisam-se todas as combinações decorrentes da troca de posição entre duas (ou três) cidades, escolhendo-se a solução que produz o melhor valor da função objetivo [Johnson e McGeoch 2003]. A metaheurística GRASP, como todas as outras, tem suas vantagens e desvantagens, conforme explicitado na tabela 3.1.

O algoritmo 3.2 apresenta o padrão para a metaheurística GRASP, em que se tem a fase da construção da solucão inicial randomizada gulosa e logo após a busca local, na qual se tenta melhorar a solução inicial. Finalmente, tem-se o armazenamento da melhor solução.

No algoritmo 3.3, tem-se a fase de construção do GRASP, em que a solução inicial é desenvolvida de forma incremental, a partir da lista restrita de canditatos.

No algoritmo 3.4 têm-se a fase de busca local para melhoria da solução inicial em sua vizinhança.

Para maiores detalhes sobre a metaheurística GRASP apresentada, podem ser encon-

Vantagens
1. De simples implementação (algoritmo guloso e busca local) 2. Ajuste nos poucos parâmetros (restritividade da lista de candidatos e número de iterações)
Desvantagens
1. Necessita de soluções iniciais boas: como se baseia apenas na mudança aleatória destas de uma iteração para outra, cada iteração se beneficia da qualidade da solução inicial; 2. Não utiliza a memória das informações coletadas durante a etapa de busca.

Tabela 3.1: Vantagens e desvantagens da metaheurística GRASP

```
1: procedure GRASP(maxIterações, α_LRC)
2: CarregaInstância();
3: for i=0 to maxIterações do
4:     solução ← ConstruçãoGulosaRandômica(α_LRC)
5:     solução ← BuscaLocal (solução, Vizinhança(solução))
6:     AtualizaSolução(solução, melhorSolução)
7: end for
8: return melhorSolução
```

Algoritmo 3.2: Metaheurística GRASP

```
1: procedure ConstruçãoGulosaRandômica(α_LRC)
2: solução ← 0
3: while solução não completa do
4:     ConstruirLRC (LRC)
5:     s ← SelecionaElementoAleatório(LRC, α_LRC)
6:     solução ← solução + s
7: end while
8: return solução
```

Algoritmo 3.3: Fase de construção da metaheurística GRASP

```
1: procedure BuscaLocal(solução,Vizinhança(solução))
2: solução ← 0
3: while solução local não for ótima do
4:     Encontrar melhor solução s ∈ Vizinhança(solução) solução ← s
5: end while
6: return solução
```

Algoritmo 3.4: Busca local da metaheurística GRASP

trados em Feo e Resende (1995).

3.3 Conclusão

Neste capítulo foi apresentada uma fundamentação teórica necessária do GRASP e do Algoritmo Genético. Para uma melhor compreensão do método proposto neste trabalho. No capítulo 5 serão apresentadas as versões implementadas no trabalho.

Capítulo 4

Aprendizagem por Reforço

Neste capítulo apresentam-se os fundamentos da aprendizagem por reforço, que é uma ferramenta utilizada na definição de estratégias em processos de decisão markovianos. Além disso, é apresentada uma formulação do problema do caixeiro viajante do ponto de vista da aprendizagem por reforço.

4.1 Processos de Decisão Markovianos

Um processo de decisão sequencial pode ser representado pela figura 4.1 abaixo [Puterman 2005]:

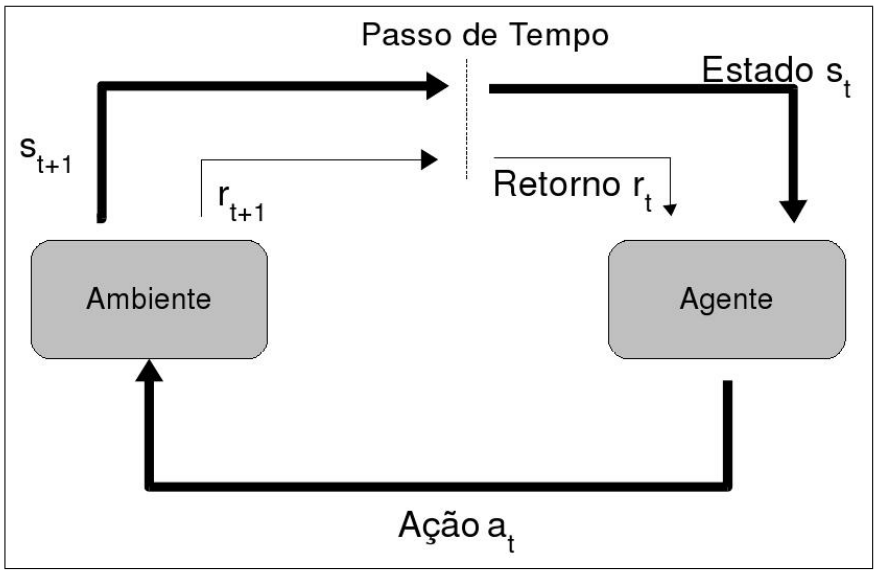

Figura 4.1: Processo de decisão sequencial

Esta figura pode ser interpretada da seguinte forma: a cada instante de tempo, o agente observa o estado do ambiente e, com base nesta informação, escolhe e pratica uma ação.

A ação produz dois resultados: o agente recebe um retorno imediato e o sistema evolui para um novo estado, de acordo com uma determinada distribuição de probabilidade. No instante seguinte, o agente depara-se com o mesmo problema, só que em um estado diferente e com um outro conjunto de ações para realizar a sua escolha.

Um processo de decisão sequencial pode ser modelado a partir dos seguintes componentes:

- Um conjunto de instantes de decisão $T = \{0,1,2,...,N\}$, $N \leq \infty$;
- Um conjunto S de estados do ambiente, que representam as percepções do agente. Este conjunto pode ser discreto (finito ou infinito) ou contínuo;
- Um conjunto de ações possíveis, resultado da união de todas as ações que podem ser executadas pelo agente em qualquer estado $A = \bigcup_{A(s)}$, $\forall s \in S$. Este conjunto também pode ser discreto (finito ou infinito) ou contínuo;
- Um conjunto de recompensas ou custos (retornos) $r_t(s,a)$, dependente dos estados e ações;
- Um conjunto de probabilidades de transição $p_t(.|s_{t-1},a_{t-1},...,s_0,a_0)$, dependente dos estados e ações.

Uma *política* π permite ao agente definir uma estratégia de escolha das ações em qualquer estado futuro. Uma *regra de decisão* $d_t(s_t)$ especifica a ação a ser escolhida em um dado instante particular. Ela pode depender apenas do estado atual ou de todos os estados e ações prévios. Uma política é uma seqüência de regras de decisão $\pi = \{d_1,d_2,...,d_N\}$, $N < \infty$. O *problema de decisão sequencial* é escolher, *a priori*, uma política que otimize a seqüência de retornos que o agente irá obter.

Um *Processo de Decisão Markoviano (PDM)* é um processo de decisão sequencial para o qual os conjuntos de ações possíveis, os retornos e as probabilidades de transição dependem somente do estado e da ação corrente e não dos estados visitados e ações escolhidas previamente. Quando isto ocorre, diz-se que o processo possui a propriedade markoviana, que pode ser expressa como:

$$p_t(s_{t+1} = s'|s_t,a_t,...,s_0,a_0) = p_t(s_{t+1} = s'|s_t,a_t) \tag{4.1}$$

O problema do caixeiro viajante pode ser modelado como um problema de decisão markoviano da seguinte maneira:

- Um conjunto de instantes de decisão $T = \{1,2,...,N\}$, onde N corresponde ao número de cidades que devem ser visitadas;

- Um conjunto $S = c_1, c_2, ..., c_N$ de estados cada um deles correspondendo a uma das N cidades a serem visitadas;
- Um conjunto de ações possíveis:

$$A(c_k) = \left\{c_1, ..., c_{i-1}, c_{i+1}, ..., c_{j-1}, c_{j+1}, ..., c_{k-1}, c_{k+1}, ..., c_N\right\},\ i > j > k \quad (4.2)$$

  que corresponde às cidades que podem ser visitadas a partir de c_k. Neste conjunto, as cidades c_i e c_j foram excluídas como exemplo de cidades que já foram visitadas antes de atingir-se a cidade c_k;
- Um conjunto de recompensas (retornos) $r_{ij}(c_i, c_j)$, função da distância entre as cidades c_i e c_j. O retorno deve refletir uma premiação que favoreça a escolha da cidade c_j mais perto de c_i;
- Um conjunto de probabilidades de transição $p_{ij}(c_j|c_i, c_j)$, correspondendo à probabilidade de se alcançar a cidade c_j, estando-se em c_i e escolhendo como ação *ir para* c_j. Os valores de p_{ij} são:

$$p_{ij}(c_j|c_i, c_j) = \begin{cases} 0, & \text{se } c_j \notin A(c_i) \text{ e,} \\ 1, & \text{se } c_j \in A(c_i) \end{cases} \quad (4.3)$$

Uma política ótima para este problema deve estabelecer qual a cidade que deve ser visitada a partir de cada c_i, $i = 1, 2, ..., N$, de tal forma que a soma dos retornos seja a melhor possível. Deve ser observado neste caso que o problema possui um horizonte de tempo finito, com conjuntos de estados e ações discretos e também finitos.

4.2 Aprendizagem por Reforço

Uma parte considerável do conhecimento que todos os seres vivos adquirem ao longo das suas vidas provém da aprendizagem. A idéia que está na base da Aprendizagem por Reforço (AR) é aquela que nos ocorre imediatamente quando pensamos na natureza da aprendizagem, que esta no princípio de que aprendemos interagindo com o ambiente. Um exemplo que podemos citar é de uma criança, principalmente nos primeiros anos de vida, pela não existência de um *professor* ou *crítico* na sua aprendizagem, pois seu aprendizado é através da tentativa e erro.

No entendimento de Sutton e Barto (1998), a Aprendizagem por Reforço (AR) consiste no aprendizado do mapeamento de estados em ações, de modo que um valor numérico de recompensa seja maximizado. A princípio, o sistema de aprendizagem por reforço não precisa conhecer as ações que devem ser tomadas, mas deve descobrir quais ações o

levam a obter a maior soma de valores de retorno, considerando-se aqui um problema de maximização.

Em problemas de AR, o agente toma decisões em função do estado no ambiente e do *retorno total esperado* que será obtido a longo prazo, partindo-se do estado atual. Mais especificamente, o agente e o ambiente interagem em uma seqüência de passos de tempo discretos ($t = 0,1,2,3,...$). A cada passo de tempo t, com base no estado s_t, $s_t \in S$ o agente seleciona uma ação $a_t \in A(s_t)$. Como conseqüência de sua ação, o agente recebe uma recompensa, que consiste de um valor numérico $r_{t+1} \in \mathbb{R}^*$, e o ambiente é levado a um novo estado s_{t+1}.

No processo de tomada de decisão do agente, o retorno total esperado desempenha um papel importante. Sua expressão matemática, para uma dada política π, é dada pela equação 4.4. Tomando-se como exemplo o problema do caixeiro viajante, o retorno total esperado associado a uma dada cidade c_i, representa o custo de um ciclo partindo desta cidade.

$$R_N^{\pi}(s_0 = s) = r_1(s_0, a_0) + r_2(s_1, a_1) + r_3(s_2, a_2) + ... + r_N(s_{N-1}, a_{N-1}) \tag{4.4}$$

Como não se conhece *a priori* o valor do retorno total, sua estimação torna-se o problema a ser resolvido pela aprendizagem por reforço. Os métodos desenvolvidos para a sua solução têm como base a programação dinâmica, os métodos de Monte Carlo e as diferenças temporais [Sutton e Barto 1998]. De maneira geral, todos são voltados para a avaliação de uma dada política π e sua subsequente melhoria, adotando-se um procedimento iterativo. Garantias de convergência são estabelecidas para cada método. Um dos algoritmos de maior sucesso para o estabelecimento da melhor política é o *Q-Learning*, que será descrito a seguir.

4.2.1 *Q-learning*

Como abordado no problema da aprendizagem por reforço existe um agente que interage com o ambiente, buscando estimar a ação de melhor retorno total esperado para cada estado, com o objetivo de estabelecer assim uma política ótima π^* de atuação.

Em Sutton e Barto (1998) pode-se verificar que quando não se dispõe de um modelo do ambiente com o qual o agente interage, o uso da Programação Dinâmica (PD) para se obter a política ótima torna-se inviável. Mas, alguns algoritmos de AR não precisam da modelagem completa do ambiente, ou seja, não precisam ter conhecimento da matriz de probabilidades de transição e valores esperados do sinal de reforço para todos os estados e ações possíveis. Neste caso, podem ser utilizados os métodos de Monte Carlo (MC) ou

de Diferenças Temporais (DT).

O algoritmo *Q-learning* é uma técnica baseada em diferenças temporais, desenvolvido por Watkins (1989) e é considerada uma das mais importantes contribuições em AR, uma vez que a sua convergência para valores ótimos de $Q(s,a)$ (estimativa do retorno total esperado, chamada *função de valor estado-ação*) não depende da política que está sendo utilizada. A expressão de atualização do valor de Q no algoritmo *Q-learning*, fundamenta-se na equação 4.5.

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r_t(s_t,a_t) + \gamma \max_a Q(s_{t+1},a_{t+1}) - Q(s_t,a_t) \right] \tag{4.5}$$

onde $s_t = s$ é o estado atual, $a_t = a$ é a ação realizada no estado s_t, $r_t(s_t,a_t)$ é o sinal de reforço (retorno ou recompensa) recebido após executar a_t em s_t, s_{t+1} é o estado seguinte, γ é o fator de desconto $(0 \leq \gamma < 1)$ e α, $(0 < \alpha < 1)$ é a taxa de aprendizagem. A função $Q(s,a)$ é o valor associado ao par *estado-ação* (s,a) e representa quão boa é a escolha dessa ação na otimização do retorno total esperado.

Um aspecto a ser observado quando da obtenção de $Q(s_t,a_t)$ é a estratégia de diversificação ou intensificação (*exploration/explotation*). Quando da atualização do valor descrito em 4.5 deve-se escolher uma dentre as várias ações possíveis de serem tomadas no estado s_t, conforme pode ser visto na figura 4.2.

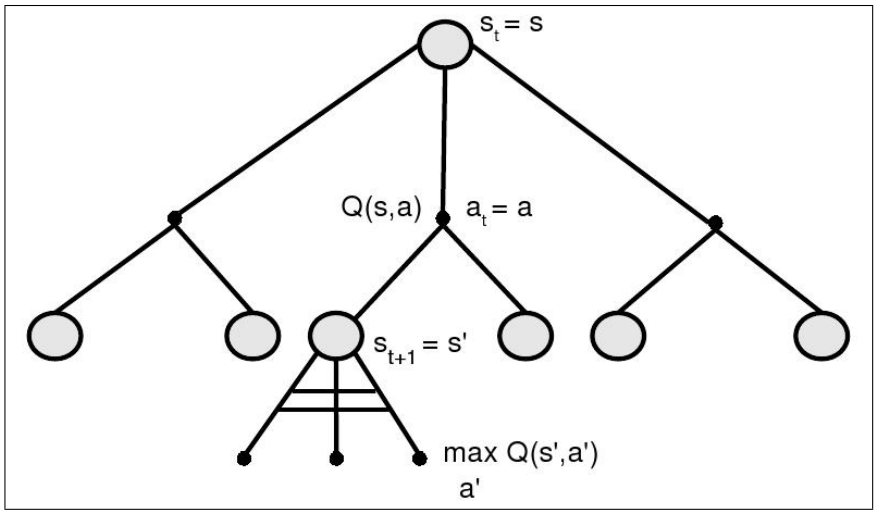

Figura 4.2: Diagrama de *back-up* relativo à escolha dos pares estado-ação

Essa escolha deve permitir que todos os pares estado-ação sejam visitados um número suficientemente grande de vezes para garantir a convergência do algoritmo. Isto irá permitir que todo o espaço $S \times A_s$ seja explorado. Entretanto, algumas das ações escolhidas

levarão a valores que não indicam tais ações como candidatas para a otimização do retorno total esperado. Como este fato não é conhecido *a priori*, deve-se estabelecer um compromisso entre visitar mais vezes as boas ações (intensificação) e explorar eventualmente ações que podem levar à descorberta de novas políticas ainda melhores que as existentes (diversificação). Uma boa estratégia para garantir a intensificação ou diversificação é o uso de uma política ε-gulosa. Tal política pode ser definida da seguinte forma: seja

$$a^* = arg \max_a Q(s,a) \tag{4.6}$$

a ação correspondente ao maior valor Q no estado s. Então, o par *estado-ação* a ter seu valor atualizado será associado à ação escolhida de acordo com a política ε-gulosa:

$$\pi(s) = \begin{cases} a^*, & \text{com probabilidade } 1-\varepsilon, \\ qualquer\ \ outra, & \text{com probabilidade } \varepsilon \end{cases} \tag{4.7}$$

O *Q-learning* foi um dos primeiros algoritmos de aprendizagem por reforço a possuir fortes provas de convergência, apresentadas em Watkins (1989), que mostrou que se cada par (s,a) for visitado um número suficientemente grande de vezes, então a função de valor $Q(s,a)$ irá convergir com probabilidade igual a um para seu valor ótimo Q^*, desde que α seja suficientemente pequeno. O algoritmo 4.1 apresenta de forma simplificada o *Q-learning*

```
1: procedure Q-learning (α, ε, γ)
2: Inicialize Q(s,a) arbitrariamente
3: loop
4:    Repita para cada etapa do episódio
5:    Inicialize s
6:    loop
7:       Repita para cada passo do episódio
8:       Selecione a de acordo com a regra ε-gulosa
9:       Observe os valores de r(s,a) e s_{t+1} = s'
10:      Q(s,a) ← Q(s,a) + α [ r(s,a) + γ max_{a_{t+1}} Q(s',a_{t+1}) − Q(s,a) ]
11:      s ← s'
12:   end loop
13: end loop
```

Algoritmo 4.1: Algoritmo *Q-learning*

Maiores detalhes sobre o algoritmo *Q-learning* podem ser encontrados em Sutton e Barto (1998) e Watkins (1989).

4.3 Conclusão

Neste capítulo foi apresentada uma fundamentação teórica dos principais conceitos ligados à Aprendizagem por Reforço com ênfase no algoritmo *Q-learning*, para uma melhor compreensão do método proposto no capítulo seguinte deste trabalho.

Capítulo 5

Abordagem paralela cooperativa e competitiva para integração dos algoritmos

5.1 Introdução

Os computadores são usados em muitas áreas de conhecimento como a ciência, a medicina e a engenharia. Em diversas aplicações, como previsão do tempo ou montagem de uma cena do próximo filme de sucesso, pode-se usar o poder computacional para tornar cada vez mais detalhadas as simulações do comportamento da atmosfera terrestre ou a representação dos quadros de imagem. Outras aplicações, como telemetria de dados vindos do espaço ou sequenciamento de DNA requerem um poder computacional ainda maior, além daquele disponível nas arquiteuras convencionais. Para tornar disponível o poder computacional necessário nestes problemas de grandes dimensões, surgiu o processamento paralelo [Mattson et al. 2004]. A seguir, serão comentadas algumas das aplicações do processamento paralelo.

No artigo proposto por Momose et al. (2004), é apresentada uma abordagem paralela do algoritmo de aprendizagem competitiva para um rápido projeto de *codebook* em espaços particionados. O artigo descreve a compressão de imagens com perdas de dados, que é feita através da Quantização Vetorial (QV). Para minimizar as perdas, deve-se encontrar um *codebook* ótimo onde haja a menor distorção possível devido à compressão com perdas. O algorimo de Kohonen (algoritmo de aprendizagem não supervisionada) é utilizado para achar esse *codebook* e, em uma das fases deste algoritmo, a de competição, no qual é gasto o maior tempo de processamento para a resolução do problema, utiliza-se o processamento paralelo.

No artigo de Ongsakul et al. (2004) é proposta uma Busca Tabu (BT) em paralelo

para resolver o *Constrained Economic Dispatch* (CED), que é usado para determinar um escalonamento ótimo de unidades de geração de energia elétrica, de modo a satisfazer a demanda de carga com o mínimo de custo operacional sujeito a restrições. A paralelização da busca tabu para o CED foi construída no espaço de busca local (vizinhança), que é decomposto em vários sub-vizinhos de igual tamanho para balancear a carga entre os processadores. Uma estratégia de seleção competitiva é utilizada para propagar a melhor solução encontrada até o momento entre todos os sub-vizinhos, que será empregada como solução inicial em cada iteração do algoritmo. Uma outra abordagem foi feita por Sevkli e Aydin (2007), que utilizam o algoritmo de *Variable Neighborhood Search* (VNS) no problema do *Job Shop Scheduling* (JSS).

5.2 Interface de comunicação para a troca de informações

Para um melhor entendimento, considere o diagrama esquemático apresentado na figura 5.1, que mostra a estrutura de interação entre os algoritmos.

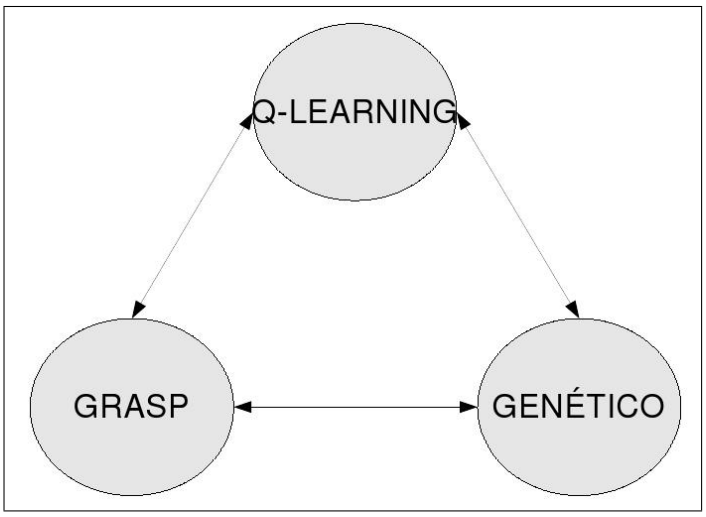

Figura 5.1: Cooperação entre os algoritmos: *Q-learning*, GRASP, Genético

Conforme abordado nos capítulos anteriores, os algoritmos genéticos trabalham com populações de soluções, o GRASP fornece uma solução ótima local e o *Q-Learning* uma tabela de Q-valores que permite a construção de soluções partindo-se de qualquer ponto da tabela de Q-valores. Todos os algoritmos envolvidos são iterativos, ou seja, a qualidade de suas soluções tende a melhorar com o aumento do número de iterações. Para

se estabelecer a comunicação entre o *Q-Learning* e os demais algoritmos, procede-se da seguinte forma:

- Usando a tabela de Q-valores atual, gera-se uma ou mais soluções da seguinte forma: escolhe-se uma cidade de partida s do PCV e consulta-se a tabela de valores $Q(s,.)$. Para o melhor (melhor custo, no caso do problema) valor $Q(s,a) = \max_a Q(s,.)$, escolhe-se a cidade a como sendo a seguinte. Repete-se o processo até que todas as cidades tenham sido visitadas uma vez apenas. Por simplicidade, considere-se que apenas uma solução é gerada;
- Para o GRASP, esta solução será utilizada como solução inicial no lugar daquela obtida na fase de construção e será melhorada utilizando-se a busca local, em uma nova iteração do algoritmo, mas só será utilizada como solução inicial, se o custo da sua solução for menor que o custo da solução do indivíduo mais apt o do Algoritmo Génetico.
  que o melhor indivíduo do Algortimo Genético;
- No caso do algoritmo genético, esta solução será inserida em uma nova população do algoritmo e processada da mesma forma que os demais indivíduos da mesma população.

Para estabelecer-se a comunicação entre o algoritmo genético e os demais algoritmos, procede-se da seguinte forma:

- A cada iteração do algoritmo, escolhe-se um ou mais indivíduos da população dentre aqueles com maior valor de adequabilidade. Por simplicidade, considere-se que apenas um indivíduo é escolhido e seja R_t o custo do ciclo associado;
- Para o GRASP, segue-se o mesmo procedimento, ou seja, esta solução será utilizada como solução inicial no lugar daquela que é obtida na fase de construção e será melhorada utilizando-se a busca local, em uma nova iteração do algoritmo, mas só será utilizada como solução inicial, se o custo da solução do indivíduo mais apto for menor, que o custo da solução gerada pelo *Q-learning*;
- Para o *Q-learning*, deve-se proceder uma inserção da informação disponibilizada na tabela de Q-valores do algoritmo. Para tanto, deve ser relembrado que um valor $Q(s,a)$ representa uma estimativa do retorno total esperado que será obtido estando no estado s e escolhendo-se a ação a. No caso do PCV, este valor representa uma estimativa do custo do ciclo partindo de s e tendo como próxima cidade visitada a. Da mesma forma, a solução produzida pelo algoritmo genético possui um custo

associado ao ciclo. Assim, partindo-se desta solução, pode-se estabelecer uma série de pares (s,a), correspondentes à ordem na qual as cidades foram visitadas e atualizar seus valores $Q(s,a)$ da seguinte forma:

$$Q(s,a) = \beta(Q(s,a) + R_t),\ 0 < \beta < 1 \tag{5.1}$$

Para estabelecer uma comunicação entre o GRASP e os demais algoritmos, procede-se da seguinte forma:

- A cada iteração do algoritmo, toma-se a solução obtida através da busca local. Por simplicidade, seja R_t o custo do ciclo associado;
- Para o algoritmo genético, esta solução será inserida em uma nova população do algoritmo e processada da mesma forma que os demais indivíduos da população.
- Para o *Q-Learning*, usa-se o mesmo procedimento descrito no item anterior e representado pela equação 5.1

Para uma melhor percepção da interação entre os algoritmos, a figura 5.2 mostra um diagrama de sequência com os algoritmos implementados, e o relacionamento entre os algoritmos.

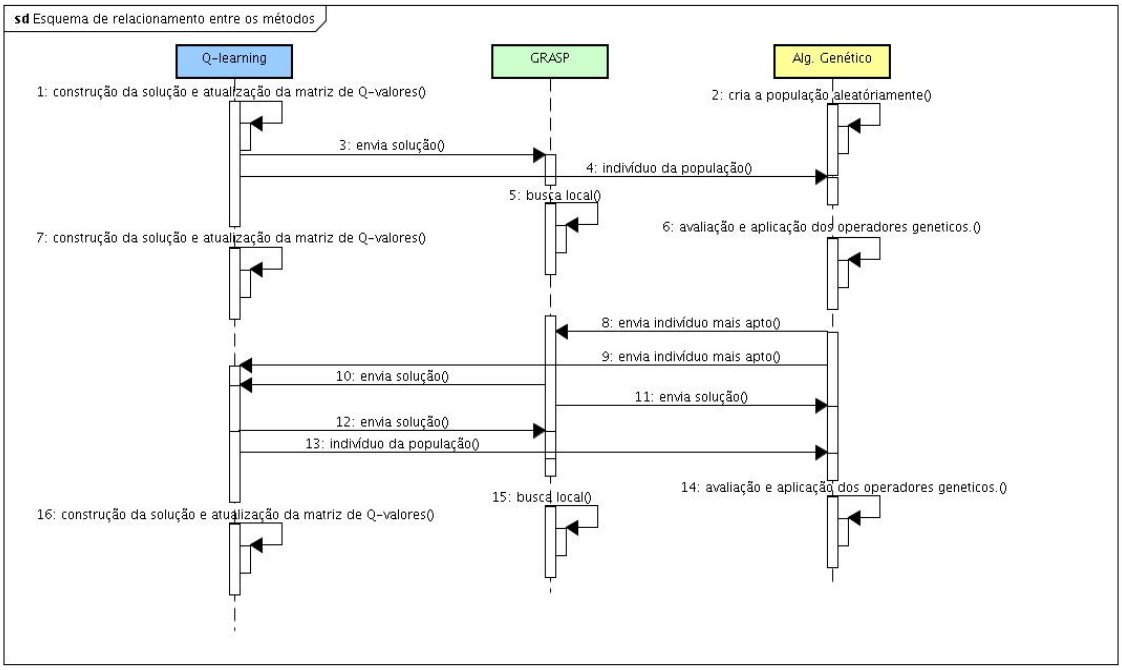

Figura 5.2: Esquema de relacionamento entre os métodos

Deve-se observar que, numa aplicação paralela, cada um dos algoritmos constituirá uma tarefa independente que será executada no seu ritmo próprio. Isto implica que a comunicação entre as tarefas não poderá ocorrer de forma síncrona, sob pena de uma das tarefas ter a sua execução bloqueada enquanto aguarda o envio dos resultados de uma outra. Neste trabalho a implementação de comunicações assíncronas entre as tarefas não pode ser verificada a contento, uma vez que as funções para comunicação assíncrona disponibilizadas pela biblioteca de comunicação não tiveram um desempenho aceitável. Para solucionar o problema, procedeu-se de forma experimental, conforme descrito abaixo:

- Para cada instância testada do problema, verificou-se o tempo de processamento de uma iteração de cada algoritmo, a saber: para o algoritmo genético (AG), tratou-se do tempo gasto na avaliação da população corrente de indivíduos e na geração de uma nova população; para o GRASP, tratou-se do tempo gasto na da busca do ótimo local dada uma solução inicial, utilizando-se a vizinhança *2-opt*; e, para o *Q-Learning* no tempo gasto na construção de uma nova solução (episódio) e atualização da matriz de *Q*-valores;
- De posse destes tempos, verificou-se que o AG era aquele com maoir tempo de processamento por iteração. Assim sendo, dividiu-se este valor pelos tempos de iteração dos outros algoritmos de forma a obter-se o número de iterações equivalentes para cada um deles, ou seja, quantas iterações do GRASP e do *Q-Learning* correspondiam a uma iteração do AG;
- Da mesma forma, verificou-se que o GRASP era o de algoritmo com menor tempo de processamento por iteração. Uma vez que a cada iteração este algoritmo precisa de uma solução inicial e os algoritmos AG e *Q-Learning* só podem informar suas melhores soluções nos momentos de comunicação (para servir como novas soluções iniciais), tornou-se necessário iniciar o GRASP com soluções aleatórias nas iterações realizadas entre os instantes de comunicação. Isto foi feito permutando-se aleatoriamente um certo número de cidades a partir de cada ótimo local obtido na iteração anterior, sendo este número de permutações um parâmetro de entrada do algoritmo paralelo híbrido;
- Finalmente, para se verificar como as comunicações influenciam o desempenho do algoritmo paralelo, optou-se por fixar como parâmetro de entrada o número de iterações do AG após as quais eram realizadas as comunicações. Desta forma, a execução paralela ficou sendo controlada pela execução do AG.

5.3 Interface de comunicação para a troca de informações entre grupos

Uma vez que cada tarefa paralela corresponde à execução de um dos algoritmos cooperativos e competitivos (em número de três), somente três processadores são necessários para a execução do algoritmo paralelo. Para tornar possível a utilização de um número maior de processadores, adotou-se o conceito de grupos, que consistem de conjuntos compostos por uma instância do AG, uma do GRASP e uma do *Q-learning*. Cada grupo possui os mesmos algoritmos definidos com *comportamentos* (conjunto de parâmetros de entrada) diferentes. No caso do AG de cada grupo, utilizou-se diferentes mecanismos de seleção, cruzamento e mutação; para o GRASP de cada grupo, diferentes vizinhanças para a realização da busca local e, para o *Q-learning* de cada grupo, diferentes taxas de aprendizagem e políticas ε-gulosa. Dada a exiguidade de tempo imposta no final da elaboração da dissertação, estes aspectos não puderam ser testados em sua totatlidade e serão omitidos neste trabalho.

O aspecto principal analisado na configuração do algoritmo em grupos foi a comunicação. Isto devido à introdução de um nível adicional de comunicação, agora entre grupos de algoritmos. Para um melhor entendimento desta comunicação em grupos, considere o diagrama esquemático apresentado na figura 5.3, que mostra a estrutura de interação entre os algoritmos em dois grupos, onde os arcos representam as possibilidades de comunicação. O resultado enviado por um grupo ao outro é aquele da melhor solução encontrada até o momento pelos algoritmos do grupo, ou seja, existe uma competição interna em cada grupo. Da mesma forma que no interior de cada grupo, os tempos de processamento por grupo são variáveis e existe a necessidade de uma comunicação assíncrona entre eles, o que não foi implementado pelas razões descritas anteriormente. Por simplicidade, optou-se neste trabalho por realizar esta comunicação inter-grupos no mesmo instante em que eram realizadas as comunicações intra-grupos, garantindo-se assim um menor tempo ocioso em cada um dos processadores.

5.4 Conclusão

Neste capítulo foi descrita a abordagem paralela utilizada para a implementação dos algoritmos GRASP, Algoritmo Genético e *Q-learning* de forma cooperativa e competitiva e a forma que foram implementadas as comunicações entre eles. Em uma abordagem mais complexa foi introduzido o conceito de grupos, como eles foram caracterizados e de que

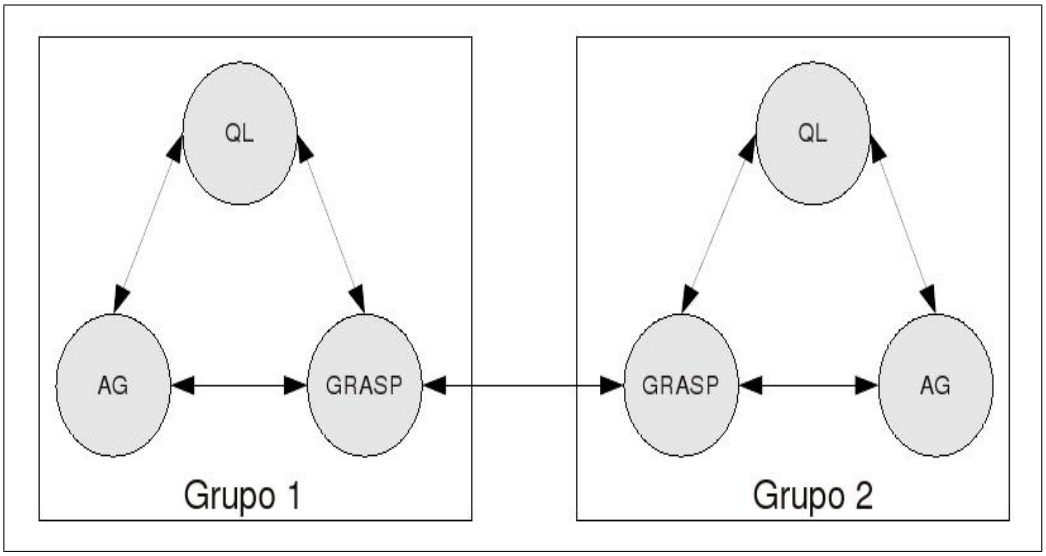

Figura 5.3: Cooperação entre os algoritmos: *Q-learning*, GRASP, Genético em grupos

forma foram feitas as comunicações entre grupos, tendo como objetivo uma melhora na qualidade da solução final.

Capítulo 6

Resultados experimentais

6.1 Introdução

Neste capítulo, são apresentados os resultados obtidos com a utilização do processamento paralelo na abordagem cooperativa e competitiva de integração dos algoritmos GRASP, Genético e *Q-learning* para o probelam do caixeiro viajante. Para tanto, foi desenvolvido um ambiente de programação permitindo a avaliação do desempenho paralelo, não apenas em termos dos parâmetros ganho e eficiência, mas também em termos da qualidade dos resultados.

6.2 Metodologia

Nesta seção trataremos da organização e utilização dos *hardwares* (equipamentos computacionais), nos quais promoveram a possibilidade do processamento paralelo, e a implementação de programas que fizeram uso desta arquitetura.

Foi utilizada na arquitetura computacional, uma rede dedicada de computadores, que formam um sistema de *cluster* do tipo *Beowulf*, que é o nome dado a um sistema específico de estações de trabalhos conectadas através de uma rede para o provimento de computação paralela. Esta arquitetura é composta por oito computadores que possuem a seguinte configuração: processador Intel Core 2 Duo com clock de 2.33 GHz, sendo os computadores com 2 GB de memória RAM e disco rígido de 80 GB, uma placa de rede padrão Gigabit Ethernet e velocidade de 1 GB/s. A rede é conectada por um *switch* 3Comm de 100 Mbps e contém sistema de proteção elétrica mantido por três *no-breaks*.

O *cluster* está dotado de um sistema operacional e um conjunto de bibliotecas necessárias ao desenvolvimento de aplicações paralelas. O sistema operacional utilizado nas máquinas é o GNU/Linux Ubuntu em sua versão 8.04. Para a comunicação utilizou-se

o pacote MPI[1] (*Message Passing Interface*), através do projeto *open source* "Open MPI" em sua versão 1.2.5 que é desenvolvido e mantido por um consórcio de academias, pesquisadores e parceiros da indústria. De modo sintético o MPI baseia-se em simular a existência de inúmeros programas diferentes trocando informações por um meio comum. Para o monitoramento de desempenho foi utilizado o Zabbix em sua versão 1.4.2. Para implementação dos algoritmos propostos, foi desenvolvido um programa utilizando-se a linguagem de programação C++, em função da familiaridade prévia do desenvolvedor com a linguagem, do material de suporte e das referências disponíveis. O programa incorpora características de programação orientada a objetos, o que facilita ainda mais a reutilização de códigos além de permitir melhores abstrações das estruturas necessárias ao projeto implementado.

Para o Problema do Caixeiro Viajante (PCV) utilizou-se a TSPLIB[2]. Esta biblioteca apresenta dezenas de instâncias para as mais diversas variantes do PCV. Desta forma, esta base tornou-se rapidamente uma referência mundial para testes na área da otimização combinatória. As instâncias (número de cidades que o caixeiro viajante deve percorrer, e as distâncias entre uma cidade e outra) utilizadas para testes neste trabalho, foram baseadas no tabalho de **?**, e foram: bays29, swiss42, berlin52, eil76, gr120, ch150, si175, a280;

6.3 Execução sequencial

Neste experimento foi realizada a execução do algoritmo paralelo de forma sequencial, onde todos os algoritmos foram executados na mesma máquina, ou seja, no mesmo nó de processamento. A tabela 6.1 apresenta os resultados obtidos com a execução serial em cada uma das oito instâncias avaliadas. Este experimento foi realizado para efeito comparativo com a execução paralela e para medição de ganho e eficiência em termos de tempo e de qualidade dos resultados que é a função objetivo (F.O.).

Instância	bays29	swiss42	berlin52	eil76	gr120	ch150	si175	a280
Valor da F.O.	2020	1284	7705	591	8789	9163	23795	4682
Tempo (seg.)	220	577	952	1390	3308	7082	9975	20300

Tabela 6.1: Execução serial das instâncias utilizadas

[1]MPI, sigla da expressão em inglês *Message Passing Interface*, com uma possível e tradução programação por passagem de mensagem. O *site* oficial do fórum está hospedado sob o domínio http://www.mpi-forum.org/

[2]As instâncias utilizadas foram obtidas do *site* do TSPLIB, que está hospedado sob o domínio http://ftp.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsplib.html

6.4 Execução por iterações ou paralelo

Os experimentos de execução paralela com uma comunicação periódica, em que cada algoritmo é executado em um nó de processamento, foram executados trinta (30) vezes no *cluster* computacional para cada uma de oito (8) instâncias do problema do caixeiro viajante.

A tabela 6.2 apresenta as configurações utilizadas para os parâmetros de execução dos algoritmos para esse conjunto de experimentos. O número de indivíduos da população do algoritmo genético foi de cem (100) elementos, a taxa de *crossover* igual a 0.7, a taxa de mutação igual a 0.2, enquanto que no algoritmo *Q-learning* os paramêtros foram ajustados da seguinte forma; $\alpha = 0.8$, $\varepsilon = 0.01$, $\gamma = 1$, a parametrização realizada das variáveis, tanto no Algoritmo Genético como no *Q-learning* foram retiradas do traballho de Lima Júnior et al. (2001), e o $\beta = 0.2$, que é o paramêtro de ajuste para a atualilização dos *Q*-valores, com as soluções vindas do GRASP e do Algoritmo Genético. Conforme explicitado no Capítulo 5, considerando-se que o tempo de execução de uma iteração do Algoritmo Genético (AG) é maior que o tempo de execução dos demais algoritmos aqui analisados, as quantidades de iterações do GRASP e do *Q-Learning* são maiores que a do AG, sendo que esta quantidade é expressa nas colunas (QL / AG) e (GRASP / AG) da tabela, indicando o número de iterações dos algoritmos *Q-Learning* e GRASP correspondentes a uma iteração do AG. De modo mais simples, na instância bays29 em uma iteração do AG, o *Q-Learning* executa oito vezes, e o GRASP executa vinte oito vezes. Deve-se observar que estas quantidades são funções das instâncias consideradas do problema, sendo maiores para as instâncias menores. O índice Aleatório indica o número de permutações realizadas na solução corrente do GRASP para definir a nova solução inicial nas iterações realizadas entre os instantes de comunicação, conforme explicitado no Capítulo 5. Já o valor associado à coluna Comunicação indica o número de iterações do AG realizadas entre duas comunicações sucessivas, ou seja, a periodicidade com que os algoritmos trocam informações.

Os gráficos das figuras 6.1 a 6.8 apresentam para cada uma das instâncias os valores normalizados das funções objetivo, obtidos em função do teste realizado. Adicionalmente, estão mostradas as faixas de desvio padrão para percepção visual da qualidade da solução em função da média atingida em todos os experimentos.Os limites do eixo vertical de cada figura foram escolhidos entre os intervalos de 20% a mais e 20% a menos em relação ao valor normalizado da média da função objetivo em cada experimento.

A figura 6.1 destaca-se pelo resultado homogêneo em todas as execuções, neste caso a função objetivo alcançou o valor ótimo conhecido nas trinta execuções. As figuras 6.2

Instância	Nºde iterações (AG)	(QL/AG)	(GRASP/AG)	Aleatório	Comunicação
bays29	20.000	8	28	5	10
swiss42	40.000	7	18	5	20
berlin52	50.000	7	18	5	20
eil76	50.000	8	10	5	20
gr120	70.000	8	7	5	30
ch150	100.000	10	6	6	50
si175	100.000	12	5	5	50
a280	100.000	12	5	6	50

Tabela 6.2: Parâmetros de execução dos algoritmos

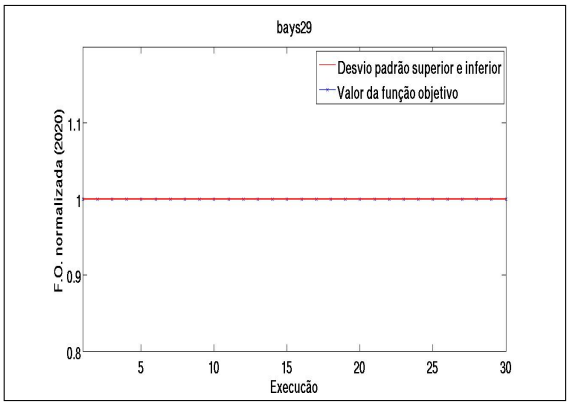

Figura 6.1: Desvio Padrão da bays29

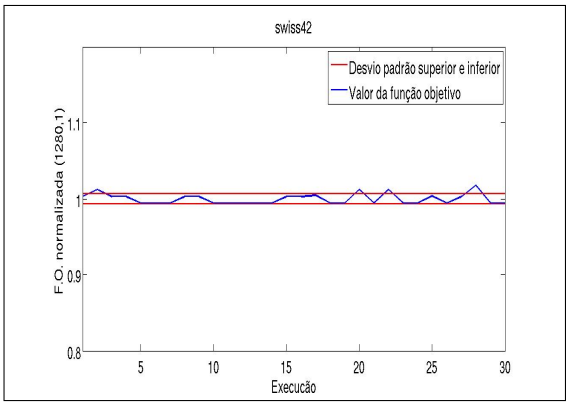

Figura 6.2: Desvio Padrão da swiss42

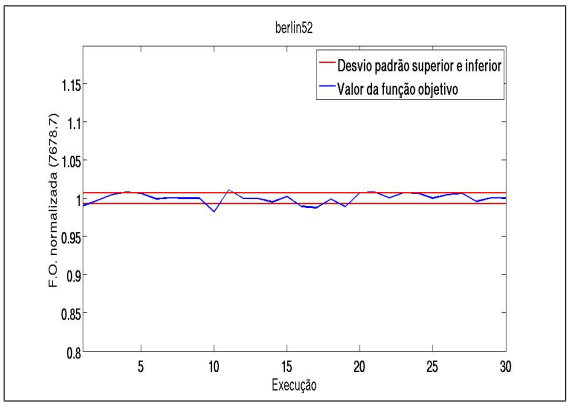

Figura 6.3: Desvio Padrão da berlin52

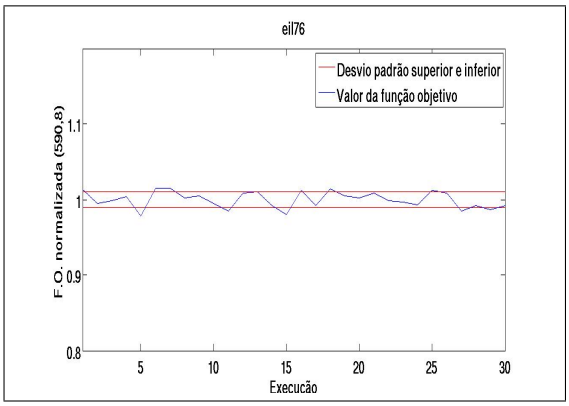

Figura 6.4: Desvio Padrão da eil76

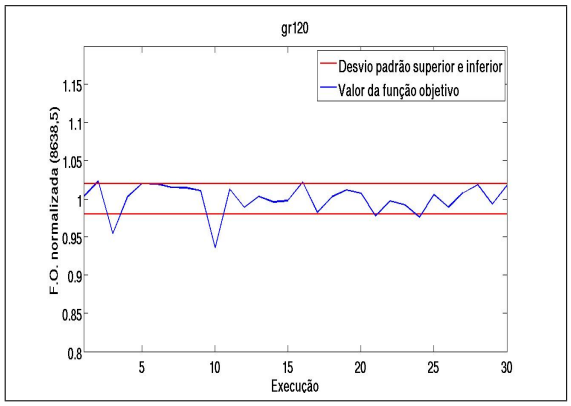

Figura 6.5: Desvio Padrão da gr120

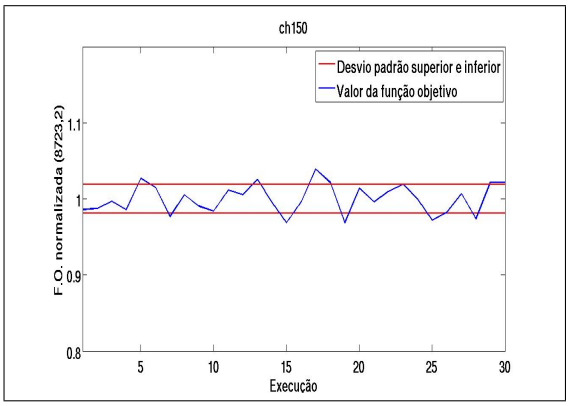

Figura 6.6: Desvio Padrão da ch150

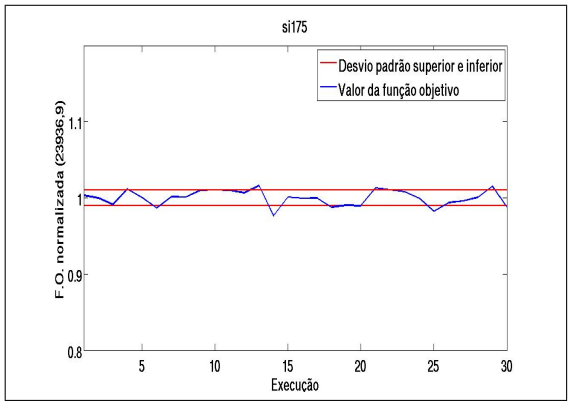

Figura 6.7: Desvio Padrão da si175

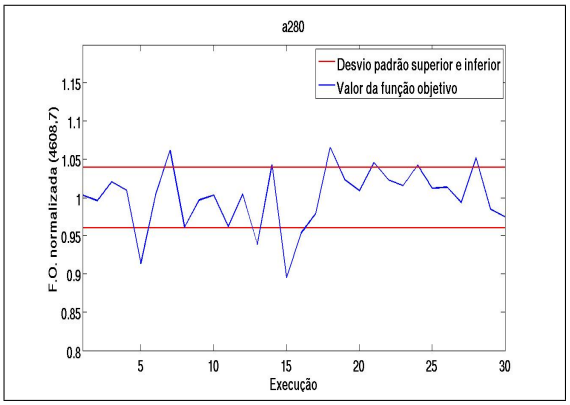

Figura 6.8: Desvio Padrão da a280

a 6.7 apresentam um comportamento bastante estável em todas as execuções, é possível confirmar através do valor dos desvios padrões que em nenhum dos casos foi superior a 2%. O comportamento menos homogêneo e de maior variância proporcional acontece na figura 6.8, provocado provavelmente pela característica complexa da instância e pelos parâmetros de execução selecionados empiricamente para os algoritmos implementados.

A tabela 6.3 apresenta uma compilação de dados estatísticos sobre esse conjunto de experimentos. A primeira linha apresenta os valores médios para a função objetivo obtidos com trinta execuções em cada instância analisada. A segunda linha apresenta o desvio padrão de cada instância em relação aos valores médios de função objetivos obtidos, a terceira linha apresenta a mesma informação sobre o desvio padrão porém percentualmente para melhor percepção da variação dos dados. A quarta linha apresenta o valor ótimo (melhor valor) da função objetivo encontrada na literatura e na base de dados para cada instância. A quinta linha apresenta a distância entre os valores médios obtidos e os valores ótimos conhecidos, o que é apresentado novamente na última linha em valores percentuais.

Instâncias	bays29	swiss42	berlin52	eil76	gr120	ch150	si175	a280
Média da F.O	2020	1280	7678,7	590,8	8638,5	8723,1	23936,9	4608,7
Desvio Padrão	0	8,7	52,1	6,3	168,6	164,7	240,6	183,4
Desvio padrão (%)	0	0,68	0,67	1,07	1,95	1,88	1	3,9
Valor ótimo da F.O.	2020	1273	7542	538	6942	6528	21407	2579
Dist. da média para F.O.	0	7,1	136,7	52,8	1696,5	2195,17	2529,9	2029,7
Dist. da média para F.O. (%)	0	0,5	1,8	9,8	24,4	33,6	11,8	78,7

Tabela 6.3: Dados estatísticos sobre os experimentos realizados por iterações

A figura 6.9 apresenta a distância normalizada entre os valores das funções objetivos obtidas com o experimento e os valores ótimos das funções objetivos encontradas na literatura para cada uma das distâncias. Nessa figura observa-se que os valores das ins-

tâncias bays29, swiss42, berlin52, eil76, gr120 e si175 apresentam valores próximos do valor normalizado, e que as instâncias ch150 e a280 se distanciam mais do valor ótimo normalizado. A instância a280 está um pouco fora do padrão dos resultados das demais instâncias, isso se deve ao refinamento adequado dos paramêtros, não sendo possível realiza-lo devido ao tempo reduzido imposto ao trabalho, e o seu tamanho que também pode influir neste resultado.

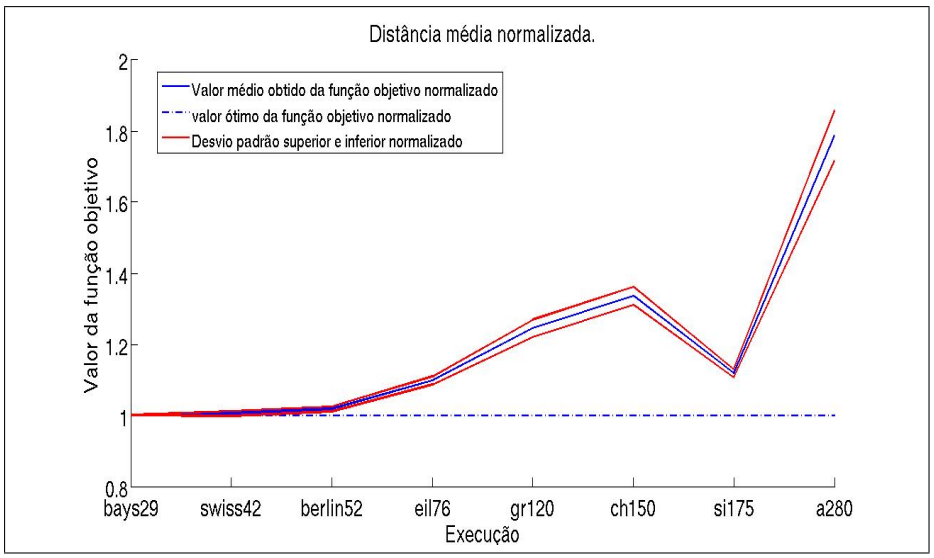

Figura 6.9: Distância normalizada entre os valores das F.O.'s obtidas e os da literatura

É possível interpretar a partir da última linha da tabela 6.3 que na média o resultado obtido a partir da execução dessa implementação para a instância 29 é 100% próximo do valor ótimo conhecido, para a instância 42 é 99,45% próximo do valor ótimo conhecido e assim consecutivamente conforme apresentado no gráfico da figura 6.10 e na tabela 6.4, onde os valores mais próximos de 100% representam melhores soluções. A distância de proximidade média máxima e mínima são obtidas através do desvio padrão percentual apresentado na tabela 6.3, o que indica que na média, mais da metade das instâncias avalidades apresentam um valor de função objetivo maior que 90% próximos do valor ótimo.

6.5 Execução limitada por tempo

Neste experimento o procedimento adotado para execução no *cluster* foi o mesmo do experimento paralelo ou por iteração da Seção 6.4, com a diferença na limitação no tempo de execução. Este foi escolhido como sendo a metade do tempo médio gasto com

	prox. média (%)	valor máximo de prox. (%)	valor mínimo de prox. (%)
bays29	100	100	100
swiss42	99.44	100	98.77
berlin52	98.19	98.87	97.51
eil76	90.19	91.26	97.51
gr120	75.56	77.51	73.62
ch150	66.37	68.26	64.48
si175	88.18	89.19	87.18
a280	21.30	25.28	17.32

Tabela 6.4: Porcentagem da média das F.O.'s encontradas

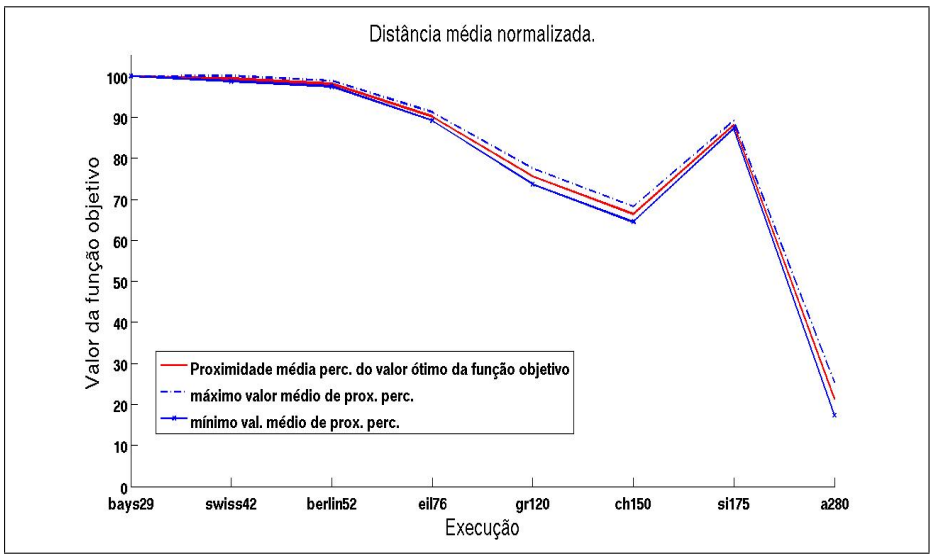

Figura 6.10: Porcentagem da média dos resultados obtidos em cada instância

a execução paralela padrão, independente da iteração que ele conseguir atingir. Cada instância neste caso foi executada trinta (30) vezes para obtenção de dados estatísticos.

A tabela 6.5 apresenta os valores obtidos com os experimentos. Comparando-os com os resultados obtidos na execução sem a limitação do tempo, observa-se que os resultados apesar de serem inferiores em qualidade, são muito próximos dos valores encontrados naquele experimento. Este comportamento foi o previsto, uma vez que o tempo de busca pelo mínimo foi reduzido.

Instâncias	bays29	swiss42	berlin52	eil76	gr120	ch150	si175	a280
Média	2020	1285,9	7751,9	598,9	8735,7	8896,1	24090,8	4732,9
Desvio padrão	0	11.0	92.8	6.4	175.4	222.5	179.6	138.3
Desvio padrão (%)	0.0	0.8	1.2	1.1	2	2.5	0.7	2.9
Ótimo	2020	1273	7542	538	6942	6528	21407	2579
Desvio do ótimo	0.0	12.9	209.9	60.9	1793.7	2368.1	2683.8	2153.9
Desvio (%) do ótimo	0	1,01	2,8	11,3	25,9	36,2	12,5	83,5
Tempo(seg.)	58	151	241	391	991	2252	3332	6513

Tabela 6.5: Dados estatísticos sobre o experimento com limite de tempo

A figura 6.11 apresenta as distâncias normalizadas entre os valores obtidos com o experimento e os valores ótimos encontrados na literatura, de onde percebe-se um comportamento semelhante ao experimento paralelo sem limitação no tempo.

6.6 Execução em grupos

Este experimento trata de uma abordagem cooperativa e competitiva em uma hierarquia maior conforme descrito no Capítulo 5. Foram realizadas dez (10) execuções em cada experimento. Cada experimento consta da colaboração de dois grupos paralelos, cada um constituído de componentes Algoritmos Genético, *Q-learning* e GRASP. O intuito foi o de avaliar se a colaboração em nível maior resultaria em um valor de função objetivo melhor do que os obtidos com as abordagens anteriores, o que de fato aconteceu e com uma redução dos parâmetros das colunas (QL / AG) ou (GRASP / AG) da tabela 6.6. A tabela 6.6 apresenta as configurações utilizadas para os parâmetros de execução

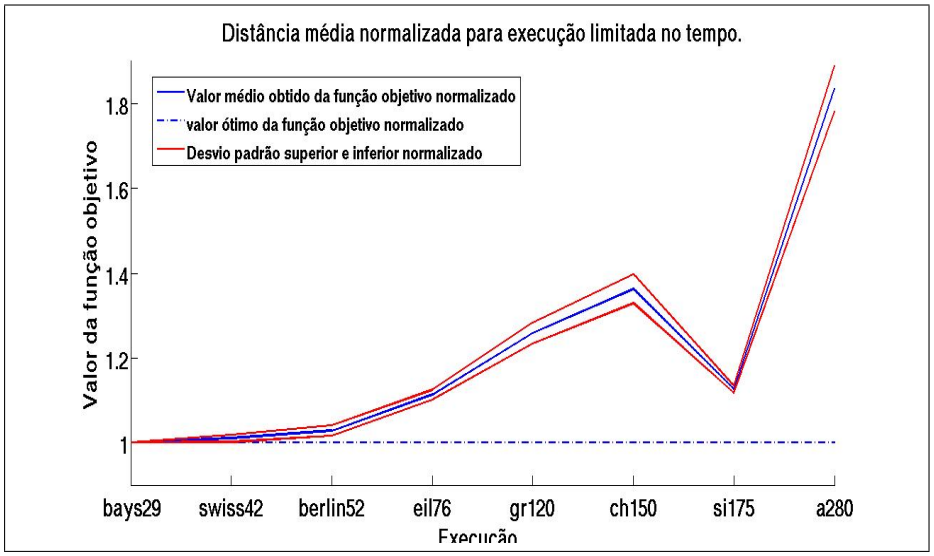

Figura 6.11: Normalização entre valores obtidos com o experimento e valores ótimos

dos algoritmos para esse conjunto de experimentos. No caso dos parâmetros próprios de cada algoritmo estes foram os mesmos do experimento anterior (taxa de mutação do AG, taxa de aprendizagem do *Q-learning*, etc.), exceto β que foi diversificado, com o intuito de verificar sua influência das inserções das soluções do AG e GRASP na tabela de *Q*-valores do *Q-learning*. Foram utilizados os seguintes valores: para o grupo 1 o β = 0.2 e para o grupo 2 o β = 0.3.

Instância	Nºde iterações (AG)	(QL/AG)	(GRASP/AG)	Aleatório	Comunicação
bays29	20.000	5	20	5	10
swiss42	40.000	5	16	5	20
berlin52	50.000	5	13	5	20
eil76	50.000	6	8	5	20
gr120	70.000	7	7	5	30
ch150	100.000	8	6	6	50
si175	100.000	8	5	5	50
a280	100.000	10	5	6	50

Tabela 6.6: Parâmetros de execução dos algoritmos em grupos

Quando comparada a tabela 6.2 com a tabela 6.6, nota-se uma redução dos parâmetros das colunas (QL / AG) ou (GRASP / AG). Isso se deve ao fator de comunicação entre os grupos, enquanto que a execução por iterações existiam três nós de processamento, na execução em grupos existiam seis nós de processamento, para que o tempo de processamneto tanto o grupo 1 quanto o grupo 2, fossem adequados para a comunicação entre eles, e

não ficassem ociosos, foi necessária uma nova parametrização dos algoritmos envolvidos, onde teve uma maior redução nas instâncias menores, enquanto nas instâncias maiores sua redução foi menor. Uma consequência decorrente disso foi, alcançar a mesma qualidade de solução da arquitetura anterior, em menos iterações do *Q-learning* e GRASP, devido a maior diversidade.

A tabela 6.7 apresenta os resultados obtidos neste conjunto de experimentos. Assim como nas tabelas 6.3 e 6.5 anteriores apresenta-se na tabela 6.7, os dados estatísticos calculados sobre os valor da função objetivo alcançada, e seus desvios padrões absolutos e percentuais, o melhor valor ótimo conhecido e a distância entre os valores ótimos conhecidos e os obtidos em termos absolutos e percentuais.

Instâncias	bays29	swiss42	berlin52	eil76	gr120	ch150	si175	a280
Média	2020	1273	7695.9	589.4	8560	8674.3	23789	4519
Desvio padrão	0	0	43.05	5.8	157.08	181.3	202.3	131.8
Desvio padrão (%)	0	0	0.56	0.98	1.81	2.1	0.85	2.9
Ótimo	2020	1273	7542	538	6942	6528	21407	2579
dist. do ótimo	0	0	153.9	51.4	1618	2146.3	2381.7	1940
dist. do ótimo (%).	0	0	2.1	9.5	23.3	32.8	11.1	75.2

Tabela 6.7: Dados estatisticos obtidos na execução em grupo

Na mesma tabela 6.7 observa-se que além da instância bays29, a instância swiss42 já obtém na média um valor com distância zero em relação ao valor ótimo, ou seja, em todas as execuções do experimento ele atinge o melhor valor conhecido da solução. Isto se reflete também nas curvas de valor de função objetivo e desvio padrão normalizados em relação ao valor ótimo conhecido mostrados na figura 6.12.

6.7 Avaliação de desempenho

A avaliação de desempenho realizada nessa seção diz respeito apenas aos dados de processamento e tempo, não levando em conta a qualidade dos resultados. Na comparação dos resultados incluem-se os dados referentes aos tempos de execução dos experimentos seriais (utilizando apenas uma máquina), paralelos (utilizandos três máquinas

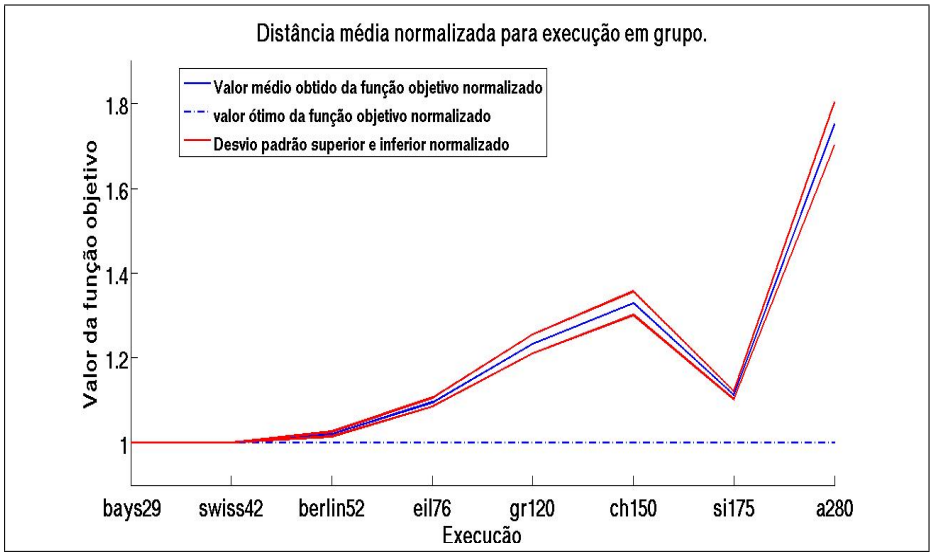

Figura 6.12: Normalização entre valores obtidos com o experimento em grupo e valores ótimos

por experimento) e os paralelos em grupos (utilizando seis máquinas por experimento). Na tabela 6.8 estão os valores dos tempos em segundos de execução para cada uma das instâncias em cada um dos tipos de teste. A mesma informação é expressa graficamente nas Figuras 6.13 e 6.14, incluídas aqui para melhor percepção visual.

Instâncias	bays29	gr42	gr52	eil76	gr120	gr150	gr175	a280
Tempo Serial (seg.)	220	577	952	1390	3308	7082	9975	20300
Tempo Paralelo (seg.)	112	302	470	790	2030	4500	6660	13050
Tempo Grupo (seg.)	115	307	464	750	2041	4237	5487	13413

Tabela 6.8: Tempo de execução das implementações realizadas

Em todos os casos o tempo de execução serial é maior do que os tempos de execução paralelo e paralelo em grupo. O experimento paralelo em grupo, apresenta um tempo semelhante a execução paralela simples, com a redução de tempo de processamento facilmente perceptível em todas as instâncias, mas a execução em paralelo por iterações e]por grupos, tem uma melhor performance na instância a280.

As medidas de avaliação de desempenho comumente utilizadas em softwares paralelos [Foster 1995] são o *speedup* e a eficiência. O *speedup* é obtido pela expressão 6.1.

$$speedup = \frac{T_s}{T_p} \tag{6.1}$$

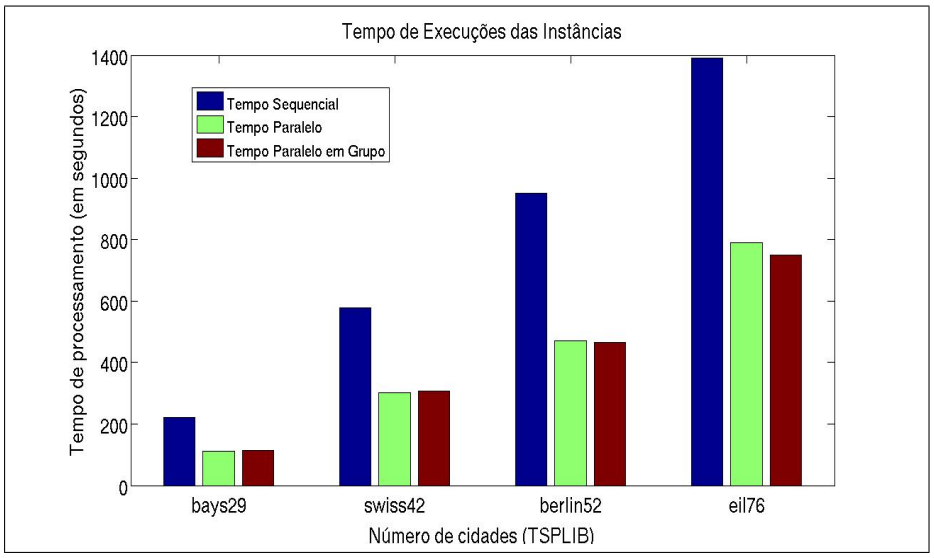

Figura 6.13: Tempo de execução das implementações

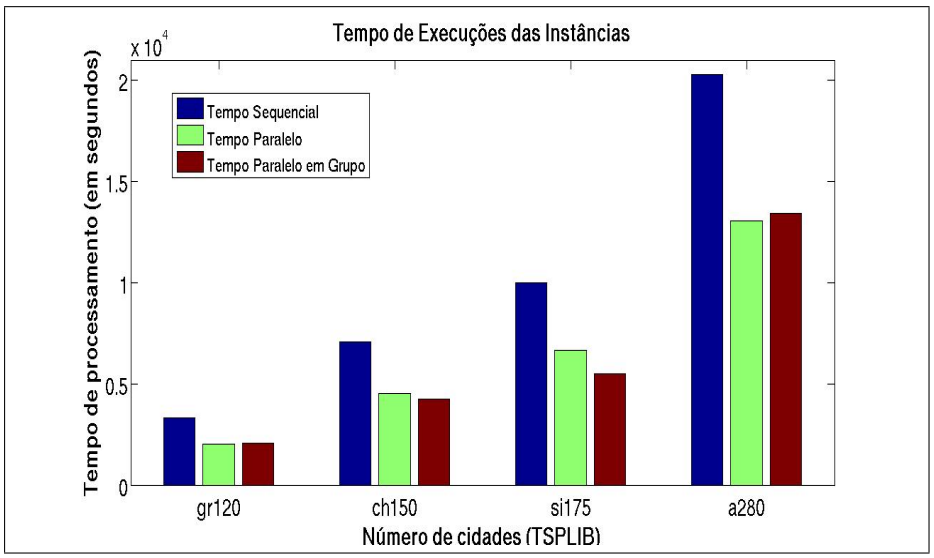

Figura 6.14: Tempo de execução das implementações

onde T_s refere-se ao melhor tempo de execução serial do algoritmo e T_p ao tempo de execução paralelo. Nesta trabalho utilizou-se como tempo de execução paralelo T_p, o tempo médio obtido nas 30 (trinta) execuções realizadas em cada instância. A eficiência pode ser calculada como mostra a expressão 6.2.

$$eficiencia = \frac{T_s}{pT_p} \tag{6.2}$$

onde, novamente, T_s é o tempo serial, T_p o tempo paralelo e p é o número de nós processadores utilizados no experimento. Sendo assim, o valor de p é igual a três nos experimentos paralelos e igual a seis nos experimentos paralelos em grupo.

A tabela 6.9 apresenta os valores de *speedup*, onde mostra-se o quanto as implementações paralelas foram mais rápidas que a sequencial, e a eficiência que mede a qualidade da implementação em paralelo, onde foi realizado para cada um dos experimentos em cada uma das instâncias avaliadas. Como não é possível avaliar as características de uma instância apenas pelo seu número de cidades, não se considera que o aumento do número de cidades produza uma crescimento linear no tempo de processamento, pois vários fatores devem ser considerados como, por exemplo, os valores dos parâmetros de cada algoritmo e a disposição das cidades.

Instâncias	bays29	swiss42	berlin52	eil76	gr120	ch150	si175	a280
Speedup Paralelo	1.96	1.91	2.02	1.73	1.62	1.57	1.49	1.55
Eficiência Paralelo	0.65	0.63	0.67	0.57	0.54	0.52	0.49	0.51
Speedup Grupo	1.91	1.87	2.05	1.85	1.62	1.67	1.81	1.51
Eficiência Grupo	0.31	0.31	0.34	0.30	0.27	0.27	0.30	0.25

Tabela 6.9: *Speedup* e eficiência da implementações

A figura 6.15 apresenta as curvas que representam o *speedup* paralelo e paralelo em grupo para efeito de comparação. A figura 6.16 apresenta as curvas de eficiência do experimento paralelo e paralelo em grupo. Observa-se que o custo para manutenção de um bom *speedup* para todas as instâncias, que apresenta-se quase constante, é exatamente uma ligeira redução de eficiência, percebido na figura 6.16 devido a adição de nós de processamento ao experimento paralelo em grupo.

6.8 Avaliação coletiva dos resultados

As tabelas 6.10 e 6.11 apresentam uma comparação qualitativa entre todos os experimentos realizados neste trabalho. Na primeira, tabela 6.10, são apresentados para cada

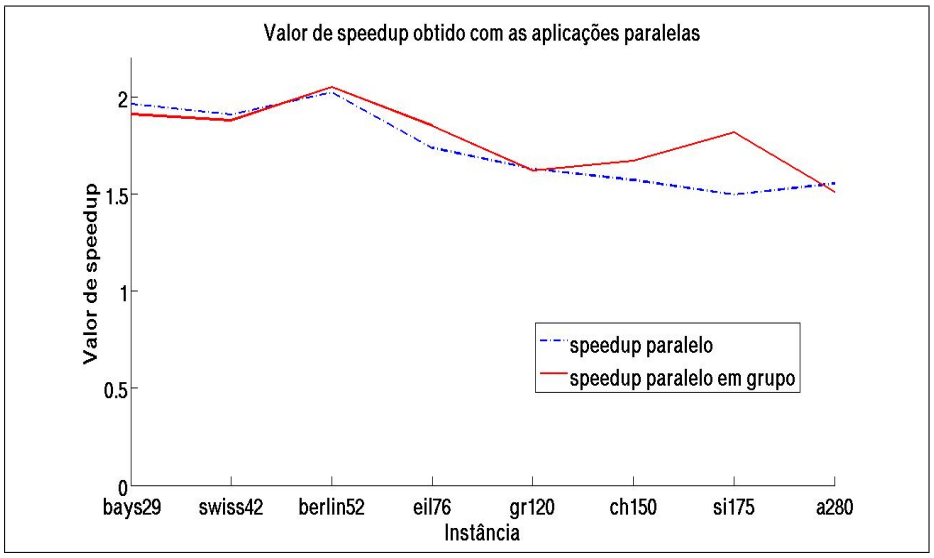

Figura 6.15: Speedup das implementações

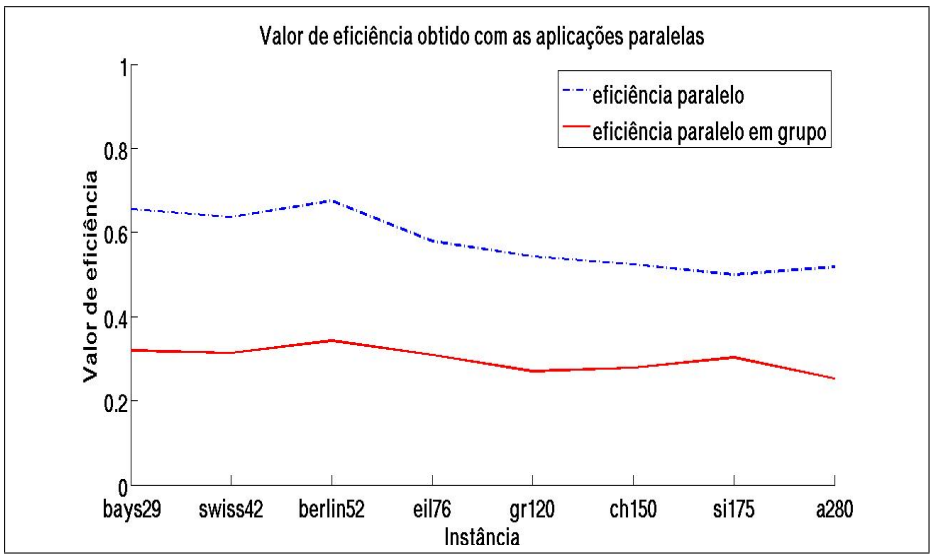

Figura 6.16: Eficiências das implementações

experimento: serial, paralelo, paralelo limitado no tempo e paralelo em grupo, e os valores médios obtidos. Na última linha observa-se o melhor valor obtido na literatura.

Instâncias	bays29	swiss42	berlin52	eil76	gr120	ch150	si175	a280
Serial	2020	1284	7705	591	8789	9163	23795	4682
Paralelo	2020	1280	7678	590	8638	8723	23936	4608
Paralelo limitado	2020	1285	7751	598	8735	8896	24090	4732
Grupo	2020	1273	7695	589	8560	8674	23788	4519
Ótimo	2020	1273	7542	538	6942	6528	21407	2579

Tabela 6.10: Comparações entre os valores obtidos da F.O., obitidas nas implementações realizadas

Já na tabela 6.11, apresenta-se a distância percentual entre os valores médios obtidos em cada experimento e o valor ótimo conhecido. A partir dessas tabelas 6.10 e 6.11 é possível observar que há certa proximidade entre os dados obtidos em cada tipo de experimento em relação a instância avaliada. Como esperado os valores da execução paralela limitada pelo tempo tiveram leve redução da qualidade comparada às outras execuções, porém no pior caso a diferença é de cerca de 8% em relação a outra instância quando comparadas ao valor ótimo conhecido, caso da instância a280.

Instâncias	bays29	swiss42	berlin52	eil76	gr120	ch150	si175	a280
Serial (%)	0	0.86	2.16	9.8	26.6	40.3	11.1	81.5
Paralelo (%)	0	0.55	1.80	9.6	24.4	33.6	11.8	78.6
Parelelo limitado (%)	0	0.94	2.77	11.1	25.8	36.2	12.5	83.4
Parelelo em grupo (%)	0	0	2.02	9.4	23.3	32.8	11.1	75.2

Tabela 6.11: Distância percentual entre os valores médios nas implementações realizadas

A partir desses dados cria-se o gráfico da figura 6.17. Neste gráfico valores mais próximos de zero são melhores, pois correspondem a distâncias menores entre os valores obtidos e os valores ótimos conhecidos. É possível perceber que o experimento que melhor contribui na média para solução das respectivas instâncias é o experimento de comunicação paralela em grupo, sendo a única excessão a instância berlin52, em que a execução paralela sem limitação no tempo conseguiu um desempenho ligeiramente superior.

6.9 Conclusão

Os resutados computacionais apresentados neste capítulo demonstram que a abordagem cooperativa e competitiva proposta neste trabalho obtiveram resultados satisfatórios,

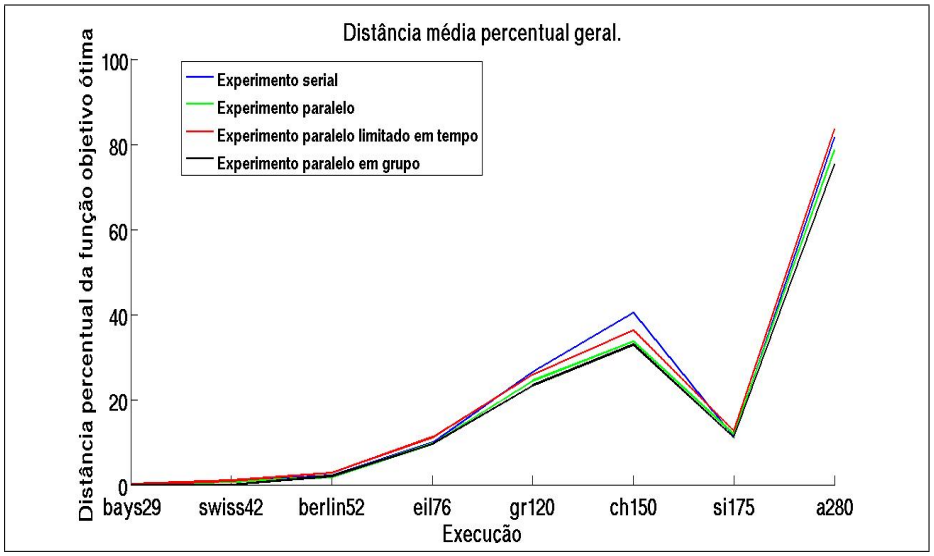

Figura 6.17: Valores médios obtidos em cada experimento e o valor ótimo conhecido

tanto na parte de cooperação e competição entre eles, quanto na parte de cooperação e competição entre grupos, na qual em duas instâncias testadas, a bays29 e a swiss42, sempre que executadas, foi encontrado o ótimo global, o restante das instâncias conseguiram chegar bem próximo do ótimo conforme analisado. Foi realizada uma análise do desempenho da abordagem proposta e verificou-se um bom comportamento em relação aos quesitos que comprovam a eficiência e o *speedup* das implementações realizadas.

Capítulo 7

Conclusões

Foi apresentada neste trabalho uma investigação da utilização do processamento paralelo para integração dos algoritmos *Q-learning*, GRASP e Algoritmo Genético de forma cooperativa e competitiva.

Foram apresentados os principais conceitos ligados a otimização combinatória, uma abordagem do problema do caixeiro viajante, com uma definição, a complexidade do Problema do Caixeiro Viajante (PCV) e aplicações do PCV para problemas do mundo real. Apresentou-se uma fundamentação teórica necessária do GRASP, Algoritmo Genético e Aprendizagem por Reforço com ênfase no algoritmo *Q-learning*, para uma melhor compreensão do método proposto neste trabalho. A abordagem paralela utilizada foi descrita para a implementação dos algoritmos GRASP, Algoritmo Genético e *Q-learning* de forma cooperativa e competitiva e o procedimento realizado para implementar as comunicações entre eles, foi usada uma abordagem de equiparação no tempo de processamento dos três algoritmos, para ficassem o menor tempo ocioso possível. Em uma abordagem mais complexa foi introduzido o conceito de grupos, como eles foram caracterizados e a forma de comunicação foi a mesma que a abordagem anterior, só que em uma hierarquia maior que são os grupos, tendo como objetivo uma melhora na qualidade da solução final.

Os resutados computacionais demonstram que a abordagem cooperativa e competitiva proposta neste trabalho obtiveram resultados satisfatórios, sendo o melhor desempenho alcançado pela abordagem em grupos, devido a sua maior diversificação no espaço de busca, seguido pela abordagem por iterações e a abordagem limitada no tempo. Em algumas instâncias foram encontradas o ótimo global, que foram as instâncias de menor porte, como a instância de bays29 e swiss42. Quando não encontrado conseguiu-se chegar bem próximo de seu valor. Realizou-se uma análise do desempenho da abordagem proposta e verificou-se um bom comportamento em relação aos quesitos que comprovam a eficiência e o *speedup* das implementações realizadas.

7.1 Trabalhos Futuros

Como o trabalho foi desenvolvido em um estilo arquitetônico modular, tatnto nos componentes (algoritmos usados) como nas comunicações, torna-se possível a extensão do mesmo em abordagens não utilizadas neste trabalho, como por exemplo:

- Executar testes computacionais com instâncias maiores (de grande porte) PCV, a fim de proporcionar uma verificação do comportamento da arquitetura proposta.
- Investigar a multiparametrização.
- Implementar as comunicações assíncronas
- Estabelecer modelos de comportamentos da execução dos algoritmos para guiar a multiparametrização.
- Abordar o problema de funções multiobjetivos
- Inserir outras metaheurísticas: Busca Tabu, Nuvem de Particulas etc...

Referências Bibliográficas

Albuquerque, Ana Claudia M. L., Jorge D. Melo e Adrião D. Dória Neto (2004), *Evolvable Machines: Theory & Practice*, Vol. 161, Springer-Verlag, New York, capítulo Evolutionary Computation and Parallel Processing Applied to the Design of Multilayer Perceptrons, pp. 181–203.

Aziz, M. e S. Boussakta (2000), 'A hybrid parallel algorithm for digital image filtering applications', *Electronics, Circuits and Systems, 2000. ICECS 2000. The 7th IEEE International Conference on* **1**(2), 591–594.

Balas, E. e P. Toth (1985), 'Branch and bound methods. in the traveling salesman problem', (1), 361–401.

Bellman, Richard (2003), *Dynamic Programming*, 1ª edição, Dover Publications.

Berger, Jean e Mohamed Barkaoui (2004), 'A parallel hybrid genetic algorithm for the vehicle routing problem with time windows', *Computers & Operations Research* **31**(12), 2037–2053.

Bernardi, Reinaldo de (2001), Aplicando a técnica de times assíncronos na otimização de problemas de empacotamento unidimensional, Dissertação de mestrado, Universidade de São Paulo, USP, São Paulo, SP.

Blazewicz, Jacek, Marek Figlerowicz, Przemyslaw Jackowiak, Dariusz Janny, Dariusz Jarczynski, Marta Kasprzak, Maciej Nalewaj, Bartosz Nowierski, Rafal Styszski, Lukasz Szajkowski e Pawel Widera (2004), Parallel DNA sequence assembly, *em* 'Proceedings of the Fifth Mexican International Conference in Computer Science', Institute of Electrical and Electronics Engineers, Montreal, Que, pp. 378 – 382.

Blum, Christian e Andrea Roli (2003), 'Metaheuristics in combinatorial optimization: Overview and conceptual comparison', *ACM Computing Surveys (CSUR)* **35**, 268 – 308.

Clarke, J., J.J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper e M. Shepperd (2003), Reformulating software engineering as a search problem, *em* 'IEE Proceedings Software', Institute of Electrical and Electronics Engineers, pp. 161 – 175.

De Jong, K. A. (1975), Analysis of the behavior of a class of genetic adaptive systems, Tese de doutorado, University of Michigan, Ann Arbor, USA.

Duni, S., P.M. Pardalos e M.G.C. Resende (2002), Parallel metaheuristics for combinatorial optimization, *em* R.Corrêa, I.Dutra, M.Fiallos e F.Gomes, eds., 'Models for Parallel and Distributed Computation - Theory, Algorithmic Techniques and Applications', Kluwer Academic Publishers, New York, pp. 179–206.

Eghbal, Mehdi, E.E. El-Araby, Naoto Yorino e Yoshifumi Zoka (2007), Application of metaheuristic methods to reactive power planning: a comparative study for GA, PSO and EPSO, *em* 'Proceedings of International Conference on Systems, Man and Cybernetics', Institute of Electrical and Electronics Engineers, Montreal, Que, pp. 3755 – 3760.

Espejo, Luis Gonzalo Acosta e Roberto D. Galvão (2002), 'O uso das relaxações lagrangeana e *surrogate* em problemas de programação inteira', *Pesquisa Operarional (online)* **22**, 387–402.
URL: *www.scielo.br/scielo.php?script=sci_arttext&pid=S0101-74382002000300006&lng=pt&nrm=iso*

Faroe, Oluf, David Pisinger e Martin Zachariasen (2001), Local search for final placement in VLSI design, *em* 'Proceedings of International Conference on Computer Aided Design', Institute of Electrical and Electronics Engineers, Jose, CA, pp. 565 – 572.

Feo, Thomas A. e Maurício G.C. Resende (1995), '*Greedy Randomized Adaptive Search Procedures*', *Journal of Global Optimization* **6**, 109–134.

Fischer, S. T. (1994), 'A note on the complexity of local search problems', *Information Processing Letters* (2), 69–75.

Foster, Ian (1995), *Designing and Building Parallel Programs*, Addison Wesley.
URL: *http://www-unix.mcs.anl.gov/dbpp/*

Gordon, V. Scott e Darrell Whitley (1993), Serial and parallel genetic algorithms as function optimizers, *em* 'Proceedings of the 5th International Conference on Genetic Algorithms', San Francisco, CA, USA, pp. 177 – 183.

Grounds, Matthew e Daniel Kudenko (2007), Parallel reinforcement learning with linear function approximation, *em* 'Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems', New York, NY, USA, pp. 1–3.

Haupt, Randy L. e Sue Ellen Haupt (1998), *Practical genetic algorithms*, 2ª edição, A Wiley-Interscience publication, Hoboken, New Jersey.

Johnson, David S. e Lyle A. McGeoch (2003), *Local Search in Combinatorial Optimization*, Princeton University Press, capítulo The Traveling Salesman Problem: a case study in local optimization.

Kushida, M., K. Takahashi, H. Ueda e T. Miyahara (2006), 'A comparative study of parallel reinforcement learning methods with a PC cluster system', *Intelligent Agent Technology, 2006. IAT '06. IEEE/WIC/ACM International Conference on* pp. 416–419.

Lima Júnior, Francisco C., Jorge D. de Melo e Adrião D. D. Neto (2001), Using *Q-Learning* algorithm for initialization of the GRASP *Metaheuristic* and *Genetic Algorithm*, *em* 'Proceedings of International Joint Conference on Neural Networks', Institute of Electrical and Electronics Engineers, Orlando, Flórida, pp. 12–17.

Maciel, André C. M. (2005), Heurísticas para o problema do caixeiro viajante branco e preto, Dissertação de mestrado, Universidade Federal Fluminense, UFF, Niteroí, RJ.

Mattson, Timothy G., Beverly A. Sanders e Berna L. Massingill (2004), *Patterns for Parallel Programming (Software Patterns Series)*, Addison-Wesley Professional.

Melanie, Mitchell (1998), *An Introduction to Genetic Algorithms*, 1ª edição, MIT Press, Cambridge, Massachusetts.

Milano, Michela e Andrea Roli (2004), 'Magma: A multiagent architecture for metaheuristics', *IEEE Transactions on Systems, Man, and Cybernetics - Part B* **33**(2).

Momose, S., K. Sano, K. Suzuki e T. Nakamura (2004), Parallel competitive learning algorithm for fast codebook design on partitioned space, *em* 'IEEE International Conference on Cluster Computing', pp. 449 – 457.

Ongsakul, W., S. Dechanupaprittha e I. Ngamroo (2004), 'Parallel tabu search algorithm for constrained economic dispatch', *Generation, Transmission and Distribution, IEE Proceedings-* **151**(2), 157–166.

Pacheco, Joaquín A., Silvia Casado e Jesús F. Alegre (2008), 'Heuristic solutions for locating health resources', *Intelligent Systems, IEEE* **23**, 57–63.

Pardalos, Panos M., Leonidas S. Pitsoulis, T. Mavridou e Mauricio G. C. Resende (1995), Parallel search for combinatorial optimization: Genetic algorithms, simulated annealing, tabu search and GRASP, *em* 'Proceedings of the Second International Workshop on Parallel Algorithms for Irregularly Structured Problems', London, UK, pp. 317–331.

Prado, José Augusto Soares (2005), Análise experimental do quicksort probabilístico com gerador de números pseudo-aleatórios penta-independente, Dissertação de mestrado, Universidade Federal do Paraná, UFPR, Curitiba, PR.

Puterman, Martin L. (2005), *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley-Interscience.

Ribeiro, Celso C. e Isabel Rosseti (2007), 'Efficient parallel cooperative implementations of GRASP heuristics', *Parallel Computing* **33**, 21–35.

Ribeiro, Celso C. e Maurício G.C. Resende (2003), '*Greedy Randomized Adaptive Search Procedures*', *Kluwer Academic Publishers* pp. 219–249.

Sevkli, Mehmet e Emin M. Aydin (2007), 'Parallel variable neighbourhood search algorithms for job shop scheduling problems', *IMA Journal of Management Mathematics* **18**(2), 117–133.

Sevkli, Mehmet e M. Emin Aydin (2006), A variable neighbourhood search algorithm for job shop scheduling problems, *em* 'Proceedings of 6th European Conference Evolutionary Computation in Combinatorial Optimization', Springer, Budapest, Hungary, pp. 261–271.

Sutton, Richard S. e Andrew G. Barto (1998), *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA.

Toulouse, Michel, Teodor Gabriel Crainic e Brunilde Sansó (2004), 'Systemic behavior of cooperative search algorithms', *Parallel Computing* **30**(1), 57–79.

Watkins, Christopher J. C. H. (1989), Learning from delayed rewards, Tese de doutorado, Cambridge University, Cambridge, England.

Zambito, Leonardo (2008), www.cse.yorku.ca, Página na internet, York University, Ontario, CA.
URL: *www.cse.yorku.ca/~aaw/Zambito/TSP_Survey.pdf*

Apêndice A

Listagem dos resultados experimentais

A.1 Resultados experimentais da abordagem paralela cooperativa e competitiva em grupo

Nº	TSPLIB							
Iterações	bays29	swiss42	berlin52	eil76	gr120	ch150	si175	a280
1	2020	1273	7676	575	8631	8851	24160	4634
2	2020	1273	7673	587	8548	8574	23833	4510
3	2020	1273	7673	592	8348	8539	24013	4669
4	2020	1273	7596	587	8770	8336	23806	4688
5	2020	1273	7736	589	8657	8987	23528	4497
6	2020	1273	7720	594	8624	8572	23441	4472
7	2020	1273	7690	597	8283	8854	23707	4458
8	2020	1273	7716	587	8710	8718	23680	4252
9	2020	1273	7721	592	8381	8580	23885	4630
10	2020	1273	7758	594	8648	8732	23834	4380
Média	2020	1273	7695,9	589,4	8560	8674,3	23788,7	4519
Tempo (seg.)	115	307	464	750	2041	4237	5487	13413

Tabela A.1: Resultados computacionais para a abordagem paralela cooperativa e competitiva em grupos

A.2 Resultados experimentais da abordagem paralela cooperativa e competitiva

Nº	TSPLIB							
Iterações	bays29	swiss42	berlin52	eil76	gr120	ch150	si175	a280
1	2020	1284	7604	599	8665	8601	24017	4623
2	2020	1296	7657	588	8838	8611	23934	4590
3	2020	1284	7716	590	8245	8695	23729	4704
4	2020	1284	7742	593	8662	8601	24219	4655
5	2020	1273	7727	578	8814	8961	23952	4210
6	2020	1273	7673	600	8809	8855	23611	4624
7	2020	1273	7682	600	8767	8519	23973	4894
8	2020	1284	7681	592	8764	8768	23958	4429
9	2020	1284	7679	594	8733	8638	24181	4594
10	2020	1273	7542	588	8086	8582	24190	4624
11	2020	1273	7765	582	8746	8823	24167	4434
12	2020	1273	7677	596	8542	8772	24095	4630
13	2020	1273	7676	597	8667	8941	24318	4325
14	2020	1273	7641	586	8603	8680	23372	4805
15	2020	1284	7695	579	8620	8451	23961	4126
16	2020	1284	7596	598	8829	8685	23922	4395
17	2020	1286	7585	586	8486	9062	23935	4511
18	2020	1273	7670	599	8661	8909	23645	4909
19	2020	1273	7596	594	8740	8447	23710	4715
20	2020	1296	7736	592	8703	8845	23689	4650
21	2020	1273	7740	596	8448	8689	24252	4819
22	2020	1296	7684	590	8614	8808	24191	4715
23	2020	1273	7735	589	8572	8892	24131	4680
24	2020	1273	7727	587	8433	8719	23921	4804
25	2020	1285	7679	598	8688	8476	23520	4665
26	2020	1273	7716	596	8545	8569	23791	4672
27	2020	1284	7728	582	8708	8781	23841	4579
28	2020	1303	7647	586	8801	8493	23952	4849
29	2020	1273	7684	583	8577	8911	24302	4539
30	2020	1273	7682	586	8790	8911	23628	4492
Média	2020	1280,07	7678,73	590,8	8638,53	8723,17	23936,9	4608,7
Tempo (seg.)	112	302	470	800	2030	4500	6660	13050

Tabela A.2: Resultados computacionais para a abordagem paralela cooperativa e competitiva

A.3 Resultados experimentais da abordagem paralela co-operativa e compopetitiva com limite de tempo

Nº	TSPLIB							
Iterações	bays29	swiss42	berlin52	eil76	gr120	ch150	si175	a280
1	2020	1285	7774	591	8719	8658	23950	4745
2	2020	1273	7804	604	8626	8976	23902	4837
3	2020	1303	7714	604	8466	8365	23919	4618
4	2020	1273	7542	596	8947	8981	24254	4918
5	2020	1273	7778	602	8632	8756	24041	4709
6	2020	1284	7673	598	8702	8978	24023	4866
7	2020	1273	7756	607	8951	8929	24350	4605
8	2020	1294	7832	590	8753	9127	24188	4728
9	2020	1297	7777	603	8697	8984	23808	4951
10	2020	1306	7738	595	8657	8758	24157	4772
11	2020	1284	7711	611	8969	8866	24209	4730
12	2020	1294	7757	597	8518	8478	24206	4603
13	2020	1285	7959	579	8485	9106	24274	4793
14	2020	1284	7734	601	8697	8625	23921	4832
15	2020	1297	7909	595	8744	8803	24148	4930
16	2020	1307	7796	599	8722	8944	23820	4693
17	2020	1273	7618	606	8598	8842	23981	4741
18	2020	1306	7795	600	8874	9160	24344	4654
19	2020	1273	7784	600	8294	9016	24316	4339
20	2020	1273	7788	606	8701	8912	24175	4479
21	2020	1286	7750	600	8719	9168	24238	4791
22	2020	1284	7917	592	8901	8964	24274	4463
23	2020	1284	7760	606	8921	9157	24275	4793
24	2020	1284	7709	600	8628	9207	24038	4748
25	2020	1301	7690	592	8647	8934	23885	4811
26	2020	1284	7670	594	8748	9052	23792	4897
27	2020	1284	7542	608	8801	8405	23936	4720
28	2020	1286	7845	596	9115	9082	23797	4753
29	2020	1274	7778	600	8801	8625	24284	4632
30	2020	1273	7657	595	9040	9026	24221	4838
Média	2020	1285,9	7751,9	598,9	8735,77	8896,13	24090,87	4732,97
Tempo (seg.)	58	151	241	391	991	2252	3332	6513

Tabela A.3: Resultados computacionais para a abordagem paralela coop. e competitiva com limite de tempo