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

Algoritmo Q-learning como Estratégia de Exploração e/ou Explotação para as Metaheurísticas GRASP e Algoritmo Genético.

Francisco Chagas de Lima Júnior

Orientador: Prof. Dr. Jorge Dantas de Melo

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

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

Número de ordem PPgEE: 0041
Natal, RN, 11 de maio de 2009

Divisão de Serviços Técnicos

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

Lima Júnior, Francisco Chagas de.

Algoritmo Q-learning como estratégia de exploração e/ou explotação para metaheurísticas GRASP e algoritmo genético / Francisco Chagas de Lima Júnior. - Natal, RN, 2009.

110 p.

Orientador: Jorge Dantas de Melo

Co-orientador: Adrião Duarte Dória Neto

Tese (Doutorado) - Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Programa de Pós-Graduação em Ciências e Engenharia Elétrica.

1. Algoritmos genéticos - Tese. 2. Metaheurísticas - Tese. 3. Q-learnig - Tese. 4. Problema do caixeiro viajante simétrico. I. Melo, Jorge Dantas de. II. Dória Neto, Adrião Duarte. III. Universidade Federal do Rio Grande do Norte. IV. Título.

RN/UF/BCZM CDU 004.421(043.2)

Algoritmo Q-learning como Estratégia de Exploração e/ou Explotação para as Metaheurísticas GRASP e Algoritmo Genético.

Francisco Chagas de Lima Júnior

Tese de Doutorado aprovada em 20 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. Carlos Henrique Costa Ribeiro ..... DTC/ITA

Prof. Dr. Gerardo Valdisio Rodrigues Viana ..... DC/UFC

Prof Dr. Dario José Aloise ..... DEP/UFRN

"Confia no Senhor de todo o teu coração e não te vanglorie no teu próprio conhecimento." (Pv. 3:5), "...porque Deus resiste aos soberbos mas aos humildes concede a sua graça." (1 Pe 5:5d)

Dedico este trabalho à minha esposa Sandra, e aos meus filhos, Débora, Sadraque e Ana Clara, aos quais peço sinceras desculpas pelos momentos em família aos quais não pude estar presente em função do desenvolvimento deste trabalho.

Agradecimentos

A gratidão é um sentimento nobre que permite que o ser humano demonstre o reconhecimento pelas ações, palavras e atitudes de seus semelhantes, e que nos faz lembrar que não se pode conquistar nenhuma vitória lutando sozinho. Meus sinceros agradecimentos:

A Deus por está sempre comigo, por ter guiado meus passos nessa jornada e por ter me concedido mais esta conquista.

A meus pais, Francisco Chagas de Lima e Terezinha Maria de Lima, pela educação informal que me concederam e pelo contínuo estímulo à busca pela educação formal.

À minha esposa, Sandra Lima, uma mulher forte que sempre me apoiou e me deu forças para continuar, mesmo quando eu por ventura desanimava.

Aos meus orientadores, Jorge Dantas de Melo e Adrião Duarte Dória Neto, pela dedicada orientação, pela segurança transmitida e pela amizade construída durante todo o curso.

À Universidade do Estado do Rio Grande do Norte pelo investimento em minha capacitação docente.

À Faculdade de Ciências e Tecnologia Mater Christi, nas pessoas do Chanceler Emerson Azevedo e da Diretora geral Maria Auxiliadora Tenório Pinto de Azevedo, pela confiança em meu potencial e pelo investimento em minha capacitação docente.

Aos meus chefes imediatos Sebastião Alves Filho e Cicília Raquel Maia Leite, pela forma atenciosa e prestativa com que sempre atenderam minhas solicitações .

Aos meus colegas do Departamento de Informática, DI/FANAT-UERN, pela compreensão e pelo apoio durante o processo de liberação de minhas atividades acadêmicas.

A todos os meus colegas do Laboratório de Sistemas Inteligentes, pelas opiniões sinceras e debates em torno da temática deste trabalho, ações que contribuíram de forma grandiosa para o desenvolvimento do mesmo.

Aos meus amigos Marcelino Pereira dos Santos Silva, Augusto Moreira de Oliveira e Francisco Reriton de Almeida Moura pelas correções e preciosas dicas na escrita dos artigos em língua Inglesa.

Aos colegas de “república” Adriano de Andrade Bresolin, Rafael Marrocos Magalhães, Ricardo de Sousa Brito, Moisés Dantas dos Santos e Kelson Rômulo Teixeira Aires, gran-

des amigos com os quais compartilhei despesas, dificuldades e alegrias.

As professoras Heliana Bezerra Soares e Ana Maria Guerreiro, que sempre muito prestativas, contribuíram de forma importante com correções e análises dos textos.

Ao professor Rommel Wladimir de Lima, grande amigo e parceiro de capacitação com o qual sempre compartilhei as angústias, alegrias e dificuldades desta jornada.

Aos meus cunhados Nivaldo Vitorino de Melo e Francisca das Chagas de Melo Costa, pelo constante apoio, e por todas as vezes que cuidaram da minha família, quando precisei me ausentar em função deste trabalho.

Ao professor Dario José Aloise e sua esposa Valmira Aloise, grandes amigos e exemplos que me orientam tanto na vida como academicamente.

A todos do Departamento de Engenharia de Computação e Automação e do Programa de Pós-graduação em Engenharia Elétrica e de Computação, que contribuíram de forma direta ou indireta com este trabalho.

A todos que torceram e/ou oraram por este projeto.

Resumo

Técnicas de otimização conhecidas como metaheurísticas têm obtido sucesso na resolução de problemas classificados como NP - Árduos. Estes métodos utilizam abordagens não determinísticas que geram soluções próximas do ótimo sem, no entanto, garantir a determinação do ótimo global. Além das dificuldades inerentes à complexidade que caracteriza os problemas NP-Árduos, as metaheurísticas enfrentam ainda o dilema de exploração/explotação, que consiste em escolher entre intensificação da busca em uma região específica e a exploração mais ampla do espaço de soluções. Uma forma de orientar tais algoritmos em busca de melhores soluções é supri-los de maior conhecimento do problema através da utilização de um agente inteligente, capaz de reconhecer regiões promissoras e/ou identificar em que momento deverá diversificar a direção de busca, isto pode ser feito através da aplicação de Aprendizagem por Reforço. Neste contexto, este trabalho propõe o uso de uma técnica de Aprendizagem por Reforço - especificamente o Algoritmo *Q-learning* - como uma estratégia de exploração/explotação para as metaheurísticas GRASP *(Greedy Randomized Adaptive Search Procedure)* e Algoritmo Genético. Na implementação da metaheurística GRASP proposta, utilizou-se o *Q-learning* em substituição ao algoritmo guloso-aleatório tradicionalmente usado na fase de construção. Tal substituição teve como objetivo melhorar a qualidade das soluções iniciais que serão utilizadas na fase de busca local do GRASP, e, ao mesmo tempo, suprir esta metaheurísticas de um mecanismo de memória adaptativa que permita a reutilização de boas decisões tomadas em iterações passadas e que evite a repetição de decisões não promissoras. No Algoritmo Genético, o algoritmo *Q-learning* foi utilizado para gerar uma população inicial de alta aptidão, e após um determinado número de gerações, caso a taxa de diversidade da população seja menor do que um determinado limite L, ele é também utilizado em uma forma alternativa de operador de cruzamento. Outra modificação importante no algoritmo genético híbrido é a proposta de um processo de interação mutuamente cooperativa entre o os operadores genéticos e o Algoritmo *Q-learning*. Neste processo interativo/cooperativo o algoritmo *Q-learning* recebe uma atualização adicional na matriz dos Q-valores com base na solução elite da população corrente. Os experimentos computacionais apresentados neste trabalho consistem em comparar os resultados obtidos com a implementação de versões tradicionais das metaheurísticas citadas, com aqueles obtidos utilizando os métodos híbridos propostos. Ambos os algoritmos foram aplicados com sucesso ao problema do caixeiro viajante simétrico, que por sua vez, foi modelado como um processo de decisão de Markov.

Palavras-chave: Metaheurística GRASP, Algoritmos Genéticos, Algoritmo Q-learning, Problema do Caixeiro Viajante.

Abstract

Techniques of optimization known as metaheuristics have achieved success in the resolution of many problems classified as NP-Hard. These methods use non deterministic approaches that reach very good solutions which, however, don't guarantee the determination of the global optimum. Beyond the inherent difficulties related to the complexity that characterizes the optimization problems, the metaheuristics still face the dilemma of exploration/exploitation, which consists of choosing between a greedy search and a wider exploration of the solution space. A way to guide such algorithms during the searching of better solutions is supplying them with more knowledge of the problem through the use of a intelligent agent, able to recognize promising regions and also identify when they should diversify the direction of the search. This way, this work proposes the use of Reinforcement Learning technique - *Q-learning* Algorithm - as exploration/exploitation strategy for the metaheuristics GRASP (Greedy Randomized Adaptive Search Procedure) and Genetic Algorithm. The GRASP metaheuristic uses *Q-learning* instead of the traditional greedy-random algorithm in the construction phase. This replacement has the purpose of improving the quality of the initial solutions that are used in the local search phase of the GRASP, and also provides for the metaheuristic an adaptive memory mechanism that allows the reuse of good previous decisions and also avoids the repetition of bad decisions. In the Genetic Algorithm, the *Q-learning* algorithm was used to generate an initial population of high fitness, and after a determined number of generations, where the rate of diversity of the population is less than a certain limit L, it also was applied to supply one of the parents to be used in the genetic crossover operator. Another significant change in the hybrid genetic algorithm is the proposal of a mutually interactive cooperation process between the genetic operators and the *Q-learning* algorithm. In this interactive/cooperative process, the *Q-learning* algorithm receives an additional update in the matrix of Q-values based on the current best solution of the Genetic Algorithm. The computational experiments presented in this thesis compares the results obtained with the implementation of traditional versions of GRASP metaheuristic and Genetic Algorithm, with those obtained using the proposed hybrid methods. Both algorithms had been applied successfully to the symmetrical Traveling Salesman Problem, which was modeled as a Markov decision process.

Keywords: GRASP Metaheuristic, Genetic Algorithm, Q-learning Algorithm, Travelling Salesman Problem.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Símbolos e Abreviaturas

AG: Algoritmo Genético

AR: Aprendizagem por Reforço

D: Matriz de Distâncias entre as Cidades do *PCV*

DT: Diferença Temporal

L: Limite de Controle da Taxa de Diversificação Utilizado nos Algoritmos Genéticos

LC: Lista de Todos os Elementos Candidatos a Compor uma Solução. Utilizada na Metaheurística GRASP

LRC: Lista Restrita de Candidatos. Utilizada na Metaheurística GRASP

MC: Métodos de Monte Carlo

MaxG: Número Máximo de Gerações Utilizado nos Algoritmos Genéticos

NEp: Número de Episódios Utilizados no Algoritmo *Q-learning*

NMax: Número Máximo de Iterações na Metaheurística GRASP

PCV: Problema do Caixeiro Viajante

PD: Programação Dinâmica

PDM: Processo de Decisão de Markov

PQA: Problema Quadrado de Alocação

$Q*$: Função Estado-Ação Ótima

TSPLIB: Biblioteca de Instâncias para o Problema do Caixeiro Viajante e Problemas Associados

Tc: Taxa de Cruzamento Utilizado nos Algoritmos Genéticos

Tm: Taxa de Mutação Utilizado nos Algoritmos Genéticos

Tp: Tamanho da População Utilizado nos Algoritmos Genéticos

α: Parâmetro Ajustável Utilizado na Fase Construtiva da Metaheurística GRASP

α_q: Coeficiente de Aprendizagem Utilizado no Algoritmo Q-learning

ε: Parâmetro que regula o critério guloso na política ε-gulosa

γ: Taxa de Desconto Utilizado no Algoritmo Q-learning

Capítulo 1

Introdução

Modelar e resolver problemas complexos do mundo real não é uma tarefa trivial, pois existem situações em que, ou é impossível se construir um modelo dada sua complexidade, ou em que o processo de simplificação de tal modelo causa perdas de informações relevantes que podem comprometer a exatidão, e conseqüentemente a qualidade da solução obtida.

Além da dificuldade inerente à construção de modelos para estes problemas, uma característica que os acompanha durante a fase de resolução, é a alta complexidade de representação ou de processamento computacional para instâncias [1] de grande porte. A complexidade elevada de tais problemas, leva-os, na maioria das vezes, a serem considerados intratáveis [2]. Neste contexto muitos pesquisadores têm se dedicado ao desenvolvimento de técnicas que visam facilitar a modelagem e principalmente a resolução destes problemas.

Dentre as técnicas existentes para se obter soluções aproximadas de problemas intratáveis destacam-se as denominadas metaheurísticas, que são procedimentos destinados a encontrar uma boa solução, eventualmente a ótima, consistindo na aplicação, em cada passo, de uma heurística subordinada, a qual tem que ser modelada para cada problema específico [Resende et al. 2004]. As metaheurísticas se diferenciam das heurísticas convencionais por seu caráter estocástico que possibilita a fuga dos pontos de ótimo local. Os ótimos locais são responsáveis pela estagnação do processo de busca pela solução ótima.

As metaheurísticas buscam o equilíbrio entre momentos de diversificação no processo de busca pela solução (exploração) e momentos de intensificação da busca em regiões promissoras do espaço de solução (explotação), com o objetivo de percorrer o espaço de soluções viáveis da melhor forma possível.

O desenvolvimento de metaheurísticas encontra inspiração em diversas áreas do conhecimento, tais como Física e Biologia, e têm obtido sucesso quando aplicadas a problemas como: roteamento ou escalonamento de veículos [Backer et al. 2000], empacotamento de caixas em *containers* [Faroe et al. 2003], projetos de computadores e de *chips* VLSI [Wang & Chen 1995], alocação de trabalhadores ou máquinas a tarefas [Miller et al. 1999], logística de produção e transporte de petróleo [Barros 2001], sequenciamento

[1] Usa-se o termo instância (do inglês *instance*) para se referir à atribuição de valores às variáveis de entrada de um problema.

[2] Um problema é dito intratável quando não existe um algoritmo em tempo polinomial que o resolva

de DNA [Blazewicz et al. 2004], dentre outros.

Uma outra técnica que tem se destacado na resolução de problemas de alta complexidade, é um paradigma de aprendizagem de máquina, conhecido como Aprendizagem por Reforço (do inglês *Reinforcement Learning*). A aprendizagem por reforço baseia-se na capacidade do agente aprendiz obter conhecimento interagindo com o ambiente desconhecido no qual está inserido. Uma das grandes vantagens da aprendizagem por reforço é justamente a capacidade de, ao interagir com um ambiente desconhecido, o agente aprendiz evoluir através da identificação das características deste ambiente, dispensando a atuação de um professor como acontece na aprendizagem supervisionada.

Assim como as metaheurísticas, a aprendizagem por reforço tem provido aplicações em diversas áreas, tais como navegação de robôs [Gedson & Romero 1999], desenvolvimento de jogos eletrônicos inteligentes [Olson 1993], controle de tráfego veicular [Wiering 2000], gerência de fluxo em tráfego aéreo [Alves et al. 2006], informática na educação [Guelpeli et al. 2004], aplicações médicas [Ernst et al. 2006] e [Fonteneau et al. 2008], otimização combinatória [Zhang & Dietterich 1996], dentre outras.

Além do uso de metaheurísticas e aprendizagem por reforço isoladamente, alguns algoritmos híbridos também tem sido desenvolvidos e aplicados na resolução de problemas práticos de difícil solução, como em [Gambardella & Dorigo 1995], [Pettinger & Everson 2002], [Zhang 2007] e [S. Girgin n.d.].

A idéia proposta neste trabalho é modificar as versões tradicionais das metaheurísticas *Greedy Randomized Adaptive Search Procedure* - GRASP e Algoritmo Genético, gerando versões híbridas que utilizam aprendizagem por reforço.

1.1 Motivação

Desde o início da década de 1970 muita pesquisa tem sido feita com o objetivo de desenvolver modelos combinatórios para problemas NP-árduos [Gilbert & Pollak 1968], [Rothfarb et al. 1970], [Du et al. 1991], [Mateus et al. 2000], a maioria destes trabalhos são referentes a algoritmos heurísticos. Os algoritmos heurísticos são limitados pois não conseguem fugir de situações que leva o processo de busca a estagnar em um ótimo local. Como resposta ao anseio por heurísticas mais poderosas surgiram as metaheurísticas que podem ser vistas como heurísticas genéricas que são direcionadas à otimização global de um problema, podendo conter diferentes procedimentos heurísticos de construção e/ou busca local na solução a cada passo.

O principal desafio de qualquer metaheurística é estabelecer, durante a busca pela solução ótima, o equilíbrio entre os processo de exploração e explotação. Estabelecer este equilíbrio consiste em decidir adequadamente sobre a necessidade ou não de experimentar novas situações (explorar novas áreas do espaço de solução), em detrimento daquelas já vivenciadas, ou seja, o desafio é resolver o dilema entre intensificar a busca nas regiões,no momento, consideradas promissoras, ou explorar na expectativas de encontrar regiões melhores no futuro.

Assim como acontece com as metaheurísticas, as técnicas de aprendizagem por reforço enfrentam dificuldades semelhantes no que diz respeito a este mesmo dilema. Neste caso o dilema consiste em decidir quando se deve aprender e quando se deve tirar pro-

veito da informação até então obtida, é importante que esta decisão seja tomada de forma autônoma. Neste contexto, o agente "aprende" quando consegue executar de forma mais eficiente determinada tarefa em função de experiências obtidas. Um exemplo de algoritmo de aprendizagem por reforço que enfrenta este dilema é o conhecido algoritmo *Q-learning* [Watkins. 1989].

No algoritmo *Q-learning* a decisão entre explorar ou explotar é feita através de uma política (na aprendizagem por reforço uma política é um função que mapeia estados[3] em ações) que permite escolher entre agir baseado na melhor informação de que o agente dispõe no momento ou agir para obter novas informações sobre o problema que possam permitir melhor desempenho no futuro. Como ambos os comportamentos trazem, em momentos distintos, benefícios à solução do problema uma boa estratégia é mesclar momentos de exploração de novas situações e momentos de usufruir da experiência adquirida. [Sutton & Barto 1998].

O algoritmo *Q-learning* pode gerenciar a decisão de explorar ou tirar proveito, por exemplo, utilizando uma política denominada ε-gulosa. Esta política é definida no algoritmo por escolher a ação que possui o maior valor de utilidade do estado (critério guloso), com probabilidade $(1-\varepsilon)$ e de ação aleatória (critério estocástico) com probabilidade ε.

Pode-se observar que o algoritmo *Q-learning* fazendo uso da política $\varepsilon - gulosa$, além de tirar proveito do conhecimento do comportamento do problema, também atua de forma semelhante a um algoritmo guloso-aleatório[4], que tem seus níveis de "gula" e aleatoriedade regulados pelo valor de ε.

A partir desta observação surgiu a idéia de utilizar o algoritmo *Q-learning* como heurística gulosa-aleatória para a fase construtiva da metaheurística GRASP, e também como gerador de população inicial para um Algoritmo Genético.

Assim diante das dificuldades inerentes à resolução de problemas reais complexos, pelo sucesso das técnicas de resolução citadas anteriormente, e por entender, que tais técnicas utilizadas conjuntamente podem cooperar mutuamente para um bom desempenho computacional, propõe-se neste trabalho o desenvolvimento de métodos híbridos utilizando o algoritmo *Q-learning* como estratégia de explotação/exploração para as Metaheurística GRASP e Algoritmo Genético. A figura 1.1 apresenta uma visão geral da idéia proposta neste trabalho.

1.2 Objetivos

Conforme já mencionado, uma das dificuldades da maioria das metaheurísticas, é a deficiência em resolver o dilema da explotação/exploração, que consiste em decidir entre intensificar em uma direção de busca, ou explorar um universo mais amplo de soluções. A falta de critério na solução de tal dilema pode levar o método de busca a ficar preso em

[3] Os estados correspondem à forma como o agente percebe determinado aspecto do problema, enquanto as ações correspondem às escolhas disponíveis ao agente, de forma que, a escolha de uma ação acarreta numa mudança de estado.

[4] Um algoritmo guloso-aleatório caracteriza-se por gerar passo-a-passo uma solução para um problema de otimização, sorteando, dentre as melhores possibilidades, os elementos componentes de tal solução.

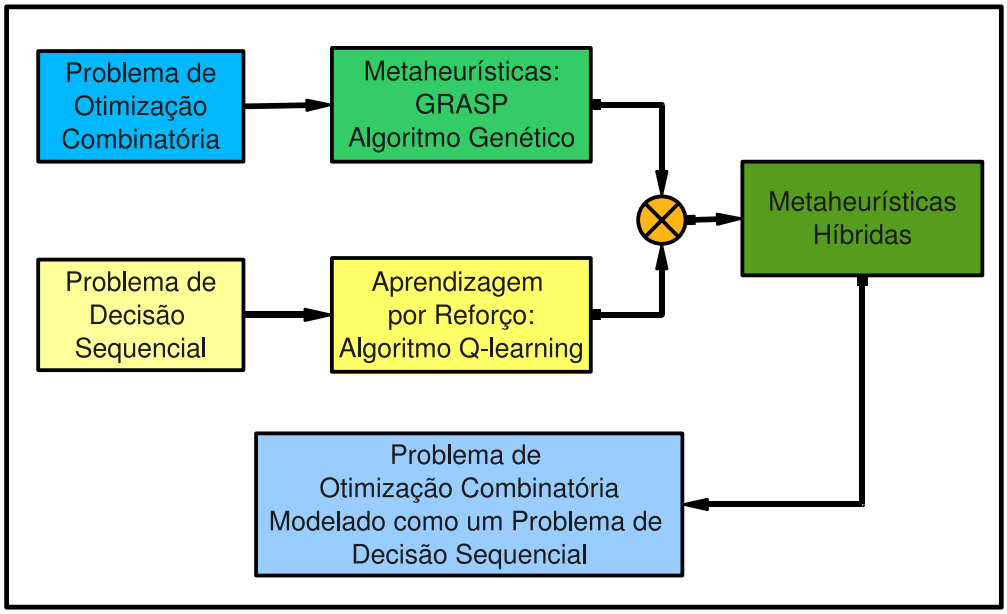


Figura 1.1: Visão geral da idéia proposta

um ponto de mínimo local. O sucesso da criação e teste de um algoritmo heurístico para um problema em particular é basicamente determinado por três fatores:

- Habilidade em usar experiência passada para criar novas soluções possíveis
- Uso de estratégia de explotação e/ou exploração
- Utilização de informações específicas do problema em foco.

Os três fatores listados anteriormente são características intrínsecas das técnicas de aprendizagem por reforço, isso representa um forte indício de que a utilização de tais técnicas pode vir a melhorar o desempenho das metaheurísticas abordadas neste trabalho.

A melhoria de uma metaheurística através da inserção de uma técnica de aprendizagem por reforço não é algo novo [Pettinger & Everson 2002], em geral, o que se deseja é que o método híbrido fuja das busca exaustivas e possibilite a geração inteligente de uma solução de boa qualidade.

Ao se propor um algoritmo heurístico espera-se que ele tome decisões que resultem, com o passar do tempo, em soluções melhores. A geração inteligente de uma solução para um problema de otimização, pode ser conseguido através de métodos híbridos utilizando um agente aprendiz que atue no ambiente [5] contexto do problema, de forma a facilitar a fuga de mínimos locais e a determinação do ótimo global [Gambardella & Dorigo 1995]. Fundamentado nestes pressupostos este trabalho tem como objetivos:

[5] O termo ambiente pode ser interpretado como um conjunto de informações que caracterizam a tarefa a ser executada pelo agente aprendiz. Em um problema de roteamento, por exemplo, o ambiente pode ser totalmente caracterizado através de um grafo ponderado (ou por sua matriz de distâncias).

Geral:

Melhorar o desempenho das metaheurísticas GRASP e Algoritmo Genético, através da utilização de uma estratégia inteligente baseada em aprendizagem por reforço, para geração de soluções iniciais de boa qualidade.

Específicos:

- Estudar as potencialidades do uso do algoritmo *Q-learning* como construtor de soluções iniciais de boa qualidade para a metaheurística GRASP. Para atingir esta meta propõe-se uma versão híbrida da metaheurística GRASP que utiliza a matriz dos q-valores (matriz gerada pelo algoritmo *Q-learning*) como mecanismo de memória adaptativa para a fase construtiva desta metaheurística.
- Verificar o desempenho de um Algoritmo Genético híbrido, que tem sua população inicial gerada pelo algoritmo *Q-learning* e que atua com o mesmo, em um processo cooperativo que atualiza, a cada geração, a matriz dos q-valores com base na solução elite da população corrente.
- Investigar a convergência do algoritmo *Q-learning* implementado nos métodos híbridos propostos, quando aplicado ao problema do caixeiro viajante simétrico, modelado como processo de decisão de Markov.
- Validar os resultados computacionais obtidos com os métodos propostos, através da análise estatística dos dados experimentais.

1.3 Estado da Arte

A idéia de utilizar aprendizagem por reforço para resolver problemas de otimização é quase tão antiga quanto a própria aprendizagem por reforço. Neste contexto, muitos pesquisadores têm desenvolvido métodos que utilizam aprendizagem por reforço em conjunto com outras técnicas tradicionais, na resolução de problemas de otimização.

Um exemplo de estudo deste tipo foi desenvolvido por Bellman (1962). Neste trabalho o autor propõe o uso de programação dinâmica aplicada ao problema do caixeiro viajante. Neste mesmo contexto outros exemplos interessantes podem ser citados: Crites & Barto (1996) propuseram a aplicação do algoritmo *Q-learning* em programação de elevadores. Neste trabalho, cada agente em uma etapa do algoritmo *AR* controla um carro de elevador particular, e o problema inteiro é resolvido cooperativamente. Zhang & Dietterich (1996) usaram o método TD(λ) para resolver um problema de *Job Shop Scheduling*. Singh & Bertsekas (1997) usaram aprendizagem por reforço para o problema de alocação de canal de comunicações.

Dentre outras publicações pertinentes, Scardua. et al. (2002), descrevem o uso de aprendizagem por reforço na obtenção de trajetórias ótimas e controle anti-balanço de um descarregador de navios, o problema de otimização tratado, pode ser encarado como um problema de decisão seqüencial em tempo discreto, no qual um controlador deve decidir, em cada época de decisão, qual a melhor ação a executar, considerando seu objetivo a longo prazo. A abordagem proposta modela o problema como um processo de decisão

de Markov, no qual, dado o elevado numero de possíveis estados do sistema o uso de programação dinâmica ou técnicas de aprendizado por reforço que utilizam representação tabular para a função valor se tornam proibitivos. Os autores propõem então, o uso de um algoritmo TD(0) juntamente com uma rede neural do tipo *perceptron* multi-camadas como um aproximador da função valor.

No que diz respeito à utilização de aprendizagem por reforço em conjunto com metaheurística destacam-se os seguinte trabalhos:

Gambardella & Dorigo (1995) desenvolveram uma técnica de otimização conhecida como *Ant System* (AS). A idéia básica deste algoritmo é utilizar uma "colônia de formigas" de forma cooperativa para localizar o menor circuito hamiltoniano em um grafo ponderado completo (o chamado Problema do Caixeiro Viajante - *PCV*), neste trabalho os autores interpretam o AS como um caso particular de uma técnica de aprendizagem por reforço distribuída e propõem, uma família de algoritmos (algoritmos Ant-Q) que fortalece a conexão entre aprendizagem por reforço, em particular o algoritmo *Q-learning*, e o método *Ant System*.

Outro trabalho bastante interessante utilizando *Ant Colony System* foi desenvolvido por Chang (2004), no qual o autor apresenta uma nova estratégia de exploração/explotação para aprendizagem por reforço baseada nesta metaheurística, denominada de *ANT-EE*. Neste trabalho o autor faz uso dos Q-valores do algoritmo *Q-learning* com base em estatísticas extraídas de múltiplas simulações de trajetórias, obtidas por um grupo de formigas (um agente aprendiz). Estas estatísticas são utilizadas para calcular o que o autor denomina de "feromônio fundido". O feromônio fundido é usado para atualizar a política de exploração do agente aprendiz, e funciona como uma medida de qualidade para tal política.

É importante observar que os trabalhos de Gambardela & Dorigo e de Chang, citados anteriormente, são bem parecidos (ambos utilizam os algoritmos Q-learning e Ant Colony System), entretanto existe uma diferença fundamental entre eles, enquanto no primeiro trabalho cada agente de aprendizagem é uma formiga, no segundo o agente aprendiz é representado por um grupo de formigas, e cada um destes grupos segue uma política de exploração diferente.

Miagkikh & Punch III (1999) propuseram o uso do algoritmo *Q-learning* em conjunto com um algoritmo genético aplicados à resolução do problema do caixeiro viajante assimétrico. No método proposto cada nova solução viável é gerada em parte pela replicação de fragmentos da melhor solução presente na população corrente do algoritmo genético, e a parte complementar da solução é preenchida usando valores convenientemente atribuídos pela estimativa dos q-valores fornecidos pelo *Q-learning*. O controle entre copiar os fragmentos da melhor solução e gerá-los utilizando os q-valores é determinado pelo parâmetro λ, o qual é uma fração do número total de atribuições realizadas, de forma que se $\lambda = 0$ todas as atribuições serão realizadas utilizando o algoritmo *Q-learning*. Os autores apresentaram resultados bastante competitivos quando comparados com outras técnicas de busca.

Ainda utilizando algoritmo genético Pettinger & Everson (2002) propuseram um algoritmo híbrido utilizando o método $Q(\lambda)$ para estimar os valores de utilidade dos pares estado-ação ao escolher um específico operador genético e os indivíduos da população

aos quais tais operadores serão aplicados. O método então denominado *Hybrid RL-GA* foi aplicado ao problema do caixeiro viajante e os resultados obtidos demonstraram que o método consegue aprender a discernir entre operadores eficazes e ineficazes.

Algumas publicações utilizando *Simulated Annealing* em conjunto com aprendizagem por reforço podem ser citadas: Atiya. et al. (2003) propõem um método de aprendizagem por reforço que utiliza o poder da metaheurística Adaptive Simulated Annealing (ASA), o método de Aprendizagem por reforço proposto pelos autores consiste da melhoria da função valor através do uso da metaheurística ASA.

Guo et al. (2004), propõem um algoritmo *Q-learning* diferenciado, denominado *AS-Q-learning* que utiliza o critério de Metropolis do *Simulated Annealing* com o objetivo de equilibrar os processos de explotação e exploração na busca da solução ótima. A estratégia proposta modifica o algoritmo *Q-learning* original substituindo a regra ε-gulosa geralmente utilizada na escolha de uma ação, por outra construída com base no critério de Metropolis.

Um outro trabalho nesta mesma linha foi publicado por Zhang (2007). Neste trabalho os autores propõe o desenvolvimento de um controlador inteligente para otimização de tráfego em redes *ATM*, que utiliza a metaheurística *Simulated Annealing* como regulador dos parâmetros de um dos seus componentes. O componente que tem seus parâmetro ajustados pelo *Simulated Annealing* é o responsável pela escolha das ações do agente de aprendizagem.

Ainda utilizando metaheurística e *AR*, foi publicado por Abramson & Harry (2003) um artigo que apresenta a metaheurística Busca Tabu como uma estratégia de exploração a ser utilizada conjuntamente com um método de aprendizagem por reforço conhecido por *Sarsa Learning Vector Quantization* (SLVQ). Neste trabalho o uso da metaheurística Busca Tabu tem como objetivo evitar a formação de ciclos de retorno às soluções precedentes, e conseqüentemente o inconveniente de ficar preso em um mínimo local.

A proposta do presente trabalho se diferencia das publicações citadas anteriormente principalmente na forma como utiliza o algoritmo *Q-learning* em conjunto com as metaheurísticas GRASP e Algoritmo Genético. A grande maioria dos trabalhos citados utilizam metaheurísticas a fim de melhorar o desempenho do algoritmo *Q-learning*, enquanto que no presente trabalho é proposto exatamente o inverso.

1.4 Principais Contribuições do Trabalho

- Proposta de utilização de Aprendizagem por Reforço na melhoria das metaheurísticas GRASP e Algoritmo Genético.
- Proposta de iteração mutuamente cooperativa entre Algoritmo Genético e o Algoritmo *Q-learning*.
- Implementação de uma versão híbrida da metaheurística GRASP com fase construtiva dotada de um mecanismo de memória capaz de repetir no futuro, as boas decisões tomadas no passado.
- Implementação de um método numérico para análise de convergência do algoritmo *Q-learning* aplicado ao problema do caixeiro viajante simétrico.

- Elaboração da análise estatística dos resultados experimentais dos métodos propostos, através de teste de hipótese e análise de sobrevivência.

1.5 Organização do Trabalho

Este trabalho está organizado da seguinte forma: No capítulo 2 é apresentada uma breve fundamentação teórica abrangendo conceitos de otimização combinatória, as diferentes abordagens de resolução existentes, e a descrição sucinta de alguns métodos exatos e aproximativos. Neste capítulo são apresentados ainda conceitos básicos de Processos de Decisão de Markov - *PDM* e Aprendizagem por Reforço, com enfoque especial no algoritmo *Q-learning*. No capítulo 3 são descritas em detalhes as metaheurísticas GRASP, GRASP Reativo e Algoritmos Genéticos que são o foco deste estudo. No capítulo 4 são descritos os métodos híbridos propostos, a metodologia de trabalho, bem como a modelagem do Problema do Caixeiro Viajante - *PCV*, como um problema de aprendizagem por reforço. O capítulo 5 apresenta os resultados experimentais obtidos e uma análise estatísticas dos mesmos. No capítulo 6 são apresentadas algumas observações, conclusões e perspectivas. Em anexo são disponibilizados a listagens de todos resultados obtidos nos experimentos realizados (Apêndice 6.0.3) e a lista das publicações relacionadas ao trabalho,(Apêndice A)

Capítulo 2

Fundamentação Teórica

2.1 Introdução

Muitos problemas práticos do cotidiano quando modelados recaem em problemas de otimização. Resolver estes problemas consiste em determinar um conjunto de parâmetros que otimizem o valor de uma função, na maioria das vezes, respeitando um conjunto de restrições. Se o conjunto de soluções viáveis para o problema modelado é discreto, o problema é dito um problema de otimização combinatória.

Uma forma trivial de resolver um problema de otimização combinatória é enumerar todas as soluções possíveis e dentre elas escolher aquela de menor custo. Entretanto, para qualquer problema de um tamanho minimamente interessante (e de utilidade prática), este método torna-se inviável, já que o número de soluções possíveis é muito grande. Em virtude desta abordagem trivial se tornar impraticável, surge a necessidade natural de se buscar métodos alternativos de solução, que forneçam uma resposta ao problema em tempo hábil. São exemplos destes métodos alternativos os algoritmos aproximativos probabilísticos denominados de metaheurísticas.

Muitos dos termos técnicos mencionados na introdução deste capítulo e nos capítulos posteriores, necessitam, de uma definição formal e de certa fundamentação teórica. Portanto, o texto apresentado nas seções a seguir terá este propósito.

2.2 Otimização Combinatória

A Otimização Combinatória surgiu com a concepção da Pesquisa Operacional no decorrer da II Guerra Mundial e seu objetivo era a melhoria dos armamentos e técnicas militares, sendo difundida nas indústrias e empresas públicas americanas após 1945. A Otimização Combinatória têm como objetivo maximizar ou minimizar uma função definida sobre certo domínio finito e discreto. Para isso, são listados os seus elementos e também testados se um dado elemento pertence ou não a esse domínio. O desafio é realizar os testes dentro de um domínio que trata milhares ou até milhões de possibilidades, gerando resultados de extrema qualidade, ou seja, eficientes e rápidos, a fim de apoiar a tomada de decisões.

Em um problema de otimização tem-se uma função objetivo e um conjunto de restrições (caso o problema apresente restrições), ambos relacionados às variáveis de decisão.

O problema pode ser de minimização ou de maximização da função objetivo. A resposta para o problema, ou seja, o ótimo global, será o menor (ou maior) valor possível para a função objetivo para o qual o valor atribuído às variáveis não viole nenhuma restrição. Em alguns casos, a alteração discreta da composição de uma solução não conduz a resultados melhores, ou seja, o processo de busca pela melhoria da solução estagna, parando assim numa solução que não representa o ótimo global - tais soluções são denominadas ótimo local. Formalmente um problema de otimização combinatória pode ser definido como segue: Dados um conjunto finito de soluções X e uma função $f : X \rightarrow \mathbf{R}$, no caso de minimização, o objetivo é encontrar, uma solução ótima $x^* \in X$ tal que $f(x^*) \leq f(x)$ $\forall x \in X$.

São exemplos bastante conhecidos de problemas de Otimização Combinatória:

- Problema do Caixeiro Viajante (*PCV*);
- Roteamento de Veículos;
- Alocação de Salas e Horários em Escolas (*Timetabling problem*);
- Escalonamento de Tarefas em Máquinas;
- Localização de Facilidades;
- Projeto de chips VLSI, dentre outros.

A resolução de um problema de otimização normalmente precisa passar por duas fases: a primeira consiste em transformar o problema em um modelo e, posteriormente, um algoritmo deve ser implementado para resolver o problema conforme modelado.

Existe pelo menos duas abordagens distintas para se resolver problemas de otimização combinatória, as quais serão descritas nas seções a seguir.

2.2.1 Abordagem Exata

Segundo Bernardi [Bernardi 2001], algoritmos exatos são algoritmos enumerativos, que varrem o espaço de soluções e buscam exaustivamente a solução que apresenta melhor característica em relação aos critérios de otimização. 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 conjuntos inteiros de soluções, através da utilização de limites (bounds).

São exemplos de abordagem exatas para problemas de otimização:

> **Programação Dinâmica** - A programação dinâmica é uma técnica que resolve problemas combinando as soluções dos sub-problemas, tal técnica conhecida como "dividir para conquistar" particiona o problema em sub-problemas independentes, resolve-os recursivamente, e então combina suas soluções para resolver o problema original. Um exemplo de aplicação desta técnica pode ser encontrado em [Belluzzo & Morabito 2005].

Relaxações - Dado um problema de otimização combinatória *P*, 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*. São exemplos de relaxações:

- Relaxação *Lagrangeana* [Akker et al. 2002];
- Relaxação *Surrogate* [Acosta & D. 2002];
- Relaxação *Lagrangeana-surrogate* [Senne & Lorena 2000].

Branch-and-Bound - É um método que baseia-se na idéia de desenvolver uma enumeração inteligente (implícita) dos pontos candidatos à solução ótima de um problema. O termo *branch* refere-se ao fato de que o método efetua partições no espaço das soluções, enquanto que o termo *bound* ressalta o fato que a prova da otimalidade da solução utiliza-se de limites calculados ao longo da enumeração. Um exemplo de aplicação deste método pode ser visto em [Wei-Chang Yeh 2004].

A grande dificuldade na resolução de problemas de otimização combinatória é que o número de possíveis soluções cresce com o tamanho da instância, em conseqüência disto, a utilização de métodos exatos torna-se inviável quando aplicados a problemas reais de grande porte. A decisão de utilizar um método exato na resolução de um problema de grande porte, depende da existência ou não de um algoritmo polinomial que o resolva.

Algoritmos Polinomiais

Algoritmos polinomiais caracterizam-se por apresentarem tempo de execução limitada por uma função polinomial do tamanho da instância do problema, ou seja, o algoritmo é polinomial se existe k tal que o consumo de tempo de processamento deste algoritmo seja da ordem de $O(n^k)$, onde n é o tamanho da instância para o problema. Por outro lado, existem os algoritmos exponenciais que caracterizam-se por apresentar aumento do tempo de processamento não proporcional ao tamanho da instância, isto é, a medida que o tamanho de sua entrada aumenta, o fator que estiver sendo analisado (tempo ou espaço) aumenta exponencialmente. A função de complexidade para algoritmos exponenciais é denotada por $O(c^n)$, sendo $c > 1$.

Exemplos de Algoritmos Polinomiais:

- Algoritmo de pesquisa binária ($O(logn)$)
- Algoritmo de pesquisa sequencial ($O(n)$)
- Algoritmo de ordenação por inserção ($O(n^2)$)
- Algoritmo de multiplicação de matrizes ($O(n^3)$)

A idéia de classificar um algoritmo como polinomial (ou de tempo polinomial) ou não, está fortemente ligada ao conceito de complexidade de algoritmos. Segundo Barbosa [Barbosa et al. 2001] a complexidade de um algoritmo consiste na quantidade de trabalho necessária para a sua execução, expressa em função das operações fundamentais, as quais variam de acordo com o algoritmo, e em função do volume de dados. As operações fundamentais são aquelas que, dentre as demais operações que compõe o algoritmo, expressam a quantidade de trabalho, portanto, estas são extremamente dependentes do problema e do algoritmo.

O conceito de algoritmo de tempo polinomial é muito importante para a definição de intratabilidade de problemas. Um problema é dito intratável se não existir um algoritmo de tempo polinomial que o resolva.

Classificação de Problemas

Neste trabalho todos os métodos propostos serão aplicados a um problema considerado intratável - o problema do caixeiro viajante simétrico. Portanto, são comuns durante todo o texto o uso de expressões como “Problemas NP-árduos”, “Problemas NP-completos”, “Intratabilidade de problemas”, dentre outros termos ligados à complexidade de problemas e algoritmos. Assim, em nome da clareza, serão apresentados nas seções a seguir, algumas definições básicas da teoria da NP-completude.

No estudo teórico da complexidade de algoritmos os problemas podem ser classificados da seguinte forma:

1. **Problemas de Decisão** O objetivo deste tipo de problema consiste em decidir (responder) “SIM” ou “NÃO” a uma questão específica.
2. **Problemas de Localização** Neste tipo de problema objetiva-se localizar uma determinada estrutura S que satisfaça um conjunto de propriedades pré-estabelecidas.
3. **Problemas de Otimização** Se as propriedades a que S deve satisfazer envolverem critérios de otimização, então o problema passa a ser visto como um problema de otimização.

Com base nesta classificação é possível formular um problema de decisão cujo objetivo é questionar se existe ou não, uma estrutura S satisfazendo a determinadas propriedades, solicitar a localização de tal estrutura, obedecendo critérios de otimimalidade, ou seja, existem tripas de problemas, um de Decisão, um de Localização e outro de Otimização que podem ser associados a todo problema de otimização.

Exemplo de Tripla associada ao Problema do Caixeiro Viajante

Considere o problema do caixeiro viajante. Seja G um grafo completo, tal que cada aresta e possui um peso $c(e) \geq 0$. Um percurso de caixeiro viajante é simplesmente um ciclo hamiltoniano de G. O peso de um percurso é a soma dos pesos das arestas que o compõe. No grafo completo da figura 2.1 um percurso de caixeiro viajante é, por exemplo, a, b, c, d, a, cujo peso é 16, enquanto que um percurso ótimo é a, b, d, c, a, de peso 11.

Neste exemplo os seguintes problemas associados podem ser formulados:

- **Problema de decisão**
  DADOS: Um grafo G e um inteiro $K > 0$;
  OBJETIVO: Verifique se existe em G um percurso de caixeiro viajante de peso K.

- **Problema de Localização**
  DADOS: Um grafo G e um inteiro $K > 0$;
  OBJETIVO: Localizar em G um percurso de caixeiro viajante de peso K.
- **Problema de Otimização**
  DADOS: Um grafo G;
  OBJETIVO: Localizar em G um percurso de caixeiro viajante ótimo.

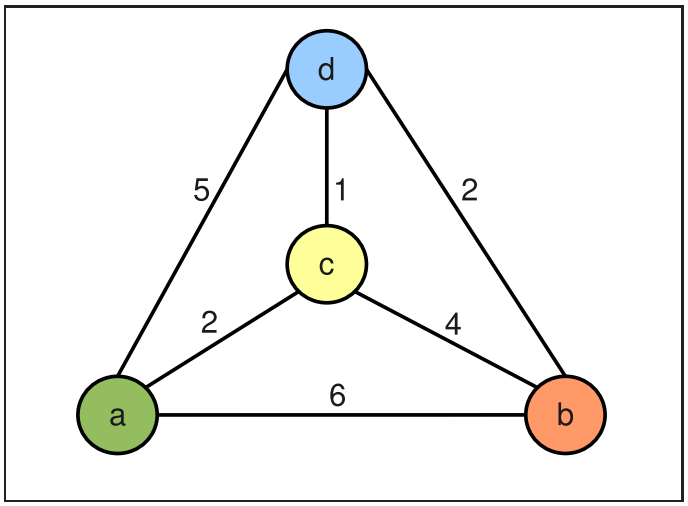


Figura 2.1: Grafo Completo: Problema do Caixeiro Viajante

Os três problemas associados do exemplo anterior estão obviamente relacionados, entretanto para definir a classe de problemas *NP* pode-se considerar apenas o problema de decisão. Uma justificativa para esta simplificação é que o problema de decisão é o mais simples dentre os três problemas associados, ou seja, dado um determinado modelo de problema, é menos custoso verificar a existência de uma estrutura S que satisfaça certas propriedades (problema de decidir se S existe ou não), do que localizar tal estrutura no modelo. Da mesma forma, um custo maior ainda seria o de localizar a estrutura S que satisfaça critérios de otimização. Isto significa que em termos de complexidade temos a seguinte hierarquia [1]:

$$\text{OTIMIZAÇÃO} >> \text{LOCALIZAÇÃO} >> \text{DECISÃO}$$

O que implica que, se em caso de problemas associados prova-se que o problema de decisão é intratável pode-se concluir que tal prova de intratabilidade, pode ser estendida aos outros dois problemas.

Classe de Problemas P

A classe de problemas *P* é a classe dos problemas de decisão que podem ser resolvidos por algoritmos (determinísticos) de tempo polinomial.

Classe de Problemas NP

Considere um problema de decisão $\P(D,Q)$, onde D representa o conjunto de dados de entrada para o problema e Q a questão (decisão) correspondente. Se $\P(D,Q)$ for computável, então necessariamente existe uma justificativa para a solução de $\P(D,Q)$. Esta justificativa pode ser representada por um determinado conjunto de argumentos, os quais,

[1] Leia-se $>>$ como "mais complexo que", em relação a hierarquia dos problemas associados

quando interpretados convenientemente, podem atestar a veracidade da resposta "SIM" ou "NÃO", dada ao problema.

Define-se então a classe de problemas *NP* como sendo aquela que compreende todos os problemas de decisão $¶_i(D_i, Q_i)$, para os quais existe uma justificativa à resposta "SIM" para $¶_i(D_i, Q_i)$, cujo passo de reconhecimento pode ser realizado por um algoritmo não-determinístico de tempo polinomial no tamanho da entrada de $¶_i(D_i, Q_i)$. A expressão "algoritmo não-determinístico de tempo polinomial" significa que tal algoritmo gera de forma probabilística uma solução candidata ao problema e verifica a resposta "SIM" em tempo polinomial.

Transformação Polinomial

Um problema de decisão $¶_1(D_1, Q_1)$ se transforma polinomialmente em outro problema de decisão $¶_2(D_2, Q_2)$ se, dada qualquer entrada D_1 para o problema $¶_1$, pode-se construir uma entrada D_2 para o problema $¶_2$ em tempo polinomial no tamanho da entrada D_1, tal que D_1 é uma instância "SIM" para $¶_1$ se e somente se D_2 é uma instância "SIM" para $¶_2$.

Problemas NP-Completos

Um problema de decisão $A \in NP$ é NP-Completo se todos os outros problemas de *NP* se transformam polinomialmente em *A*

Lema 1 *Um problema A é NP-completo se somente se:*
(1) A é NP
(2) Existir um problema B NP-completo transformável polinomialmente em A.

Problemas NP-Árduos

Problemas NP-árduos ou NP-difíceis: são problemas que satisfazem apenas o item (2) do Lema 1.

Todos os métodos propostos neste trabalhos serão aplicados a um problema de otimização combinatória bastante conhecido, o Problema do Caixeiro Viajante, o qual será descrito formalmente a seguir.

2.2.2 O Problema do Caixeiro Viajante - *PCV*

O Problema do Caixeiro Viajante - *PCV* (do inglês, *Traveling Salesman Problem - TSP*) é um dos mais conhecidos problemas de otimização combinatória, resolvê-lo consiste em determinar, em um grafo ponderado um circuito hamiltoniano de custo mínimo. O *PCV* pertence a classe de problemas que se acredita ser insolúvel em tempo polinomial, e por isso foi classificado como NP-Árduo por Karp [Karp 1975] e considerado intratável por Garey e Johnson [Garey & Johnson 1979].

Na prática o problema consiste em: dado um conjunto de cidades $C = \{c_1, c_2, c_3, ...c_n\}$, e a cada par de cidades (c_i, c_j) uma distância d_{ij} associada, determinar uma seqüencia de

visitas (uma permutação simples, ou uma rota) que inclua todas as cidades, e que retorne a cidade de origem com custo mínimo. O custo associado a uma rota é definido como o comprimento do ciclo, que corresponde ao valor relativo ao custo do caixeiro visitar cada uma das cidades na ordem especificada pela permutação, e em seguida retornar a cidade de origem.

De modo mais formal, o objetivo do *PCV* é encontrar o caminho hamiltoniano de menor custo, em um grafo $G = (V,A,W)$, em que V é o conjunto de vértices do grafo ($|V| = n, n \geq 3$), A o conjunto das arestas que ligam esses vértices e W o conjunto de pesos, onde $c_{ij} \in W$, $c_{ij} \geq 0$ é o custo associado a cada uma das arestas (i, j) de G. c_{ij} denota o custo do deslocamento da cidade c_i a cidade c_j, ou em outra palavras à distância d_{ij}.

A modelagem clássica para o *PCV* em variáveis inteira $0-1$, [Dantzig et al. 1954] é dada por:

$$Minimizar \quad Z = \sum_{j=1}^{n} \sum_{i=1}^{n} c_{ij} x_{ij} \tag{2.1}$$

Sujeito a:

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j \in V \tag{2.2}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i \in V \tag{2.3}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| \qquad \forall S \subset V \tag{2.4}$$

$$x_{ij} \in 0,1 \qquad \forall i, j \in V \tag{2.5}$$

onde a variável binária x_{ij} assume valor 1 se o arco $(i, j) \in A$ compõe a solução e 0 caso contrário. S é um sub-grafo do grafo G e $|S|$ é a cardinalidade do sub-grafo. As restrições 2.2 e 2.3 garantem que cada vértice será visitado uma única vez, enquanto que a restrição 2.4 garante a eliminação das sub-rotas. Esta modelagem matemática destaca o aspecto combinatório do *PCV*, logo percebe-se que solucionar o problema consiste em determinar uma combinação que respeite as restrições e que tenha custo mínimo.

Com base no problema originalmente proposto, existem diferentes formulações para o *PCV*, neste trabalho a formulação utilizada é a do *PCV* simétrico caracterizado pelo fato de $d_{ij} = d_{ji}$. Caso esta condição não seja satisfeita o *PCV* é dito assimétrico.

Além de ser um problema clássico, o *PCV* tem uma grande importância prática por possibilitar que, a partir dele, sejam feitas diversas aplicações úteis em situações reais, [Reinelt n.d.], o que o torna assim, uma referência para os estudos na área de otimização, e tem motivado vários pesquisadores [Merz & Freisleben 2001], [Krasnogor & Moscato 1995], [Kureichick et al. 1996] a se empenharem no sentido de desenvolver novas técnicas de otimização que se aproximem, cada vez mais do ótimo global e que ao mesmo tempo tenha um desempenho computacional aceitável na resolução deste problema.

Na seção a seguir serão descritos sucintamente, alguns métodos de abordagem aproximativa normalmente utilizados na resolução de problemas NP-Árduos.

2.2.3 Abordagem Aproximativa

Conforme mencionado na seção anterior, a utilização de métodos exatos devem ser aplicados sob determinadas circunstâncias, onde as instâncias consideradas são suficientemente pequenas ou o tempo de processamento que se tem disponível é adequado à aplicação considerada. Além disso, em muitos problemas práticos, os dados de entrada são conhecidos apenas parcialmente ou de forma imprecisa, isto significa que uma solução ótima dispendiosa em termos de tempo de processamento, não compensa quando comparada com uma solução suficientemente próxima desta, e obtida a um baixo custo computacional.

Os algoritmos aproximativos ou heurísticos [Johnson 1974] são algoritmos polinomiais que buscam sacrificar o mínimo possível da qualidade da solução obtida pelos métodos exatos, ganhando, simultaneamente, o máximo possível em eficiência computacional, sem garantir a determinação da solução ótima. Segundo [Hochbaum 1997], a busca do equilíbrio entre estas duas situações conflitantes é o grande paradigma dos algoritmos aproximativos. Para a formalização do conceito de algoritmo aproximativo considere a seguinte notação:

Seja ¶ um problema de otimização combinatória, e I, uma instância qualquer de ¶. Se A é um algoritmo aproximativo para ¶, então $x_A(I)$ é o valor da função objetivo gerado por A, $\forall I \in$ ¶. O valor da solução ótima associado ao problemas será representado por $x^*(I)$.

Os algoritmos aproximativos dividem-se em duas classes:

Algoritmos Aproximativos Determinísticos - Um algoritmo aproximativo deverá, necessariamente, ser polinomial no tamanho de qualquer instância para o problema, desta forma um algoritmo A com solução $x_A(I)$ é $f(n)$-aproximado para um problema de minimização (maximização) ¶ se, e somente se, qualquer que seja a instância I de tamanho n, a solução obtida é no máximo (mínimo) $f(n)$ vezes o valor da solução ótima $x^*(I)$. Em problemas de maximização assume-se que $x^*(I) > 0$.
Na execução de um algoritmo aproximativo determinístico a entrada de uma mesma instância de dados resultará sempre na mesma solução.

Algoritmos Aproximativos Probabilísticos - Nos algoritmos aproximativos probabilísticos, a solução esperada e o tempo esperado de processamento serão parâmetros importantes a serem observados sendo representados por variáveis aleatórias discretas. No caso probabilístico, um algoritmo aleatório A de complexidade polinomial é $f(n)$-aproximado para ¶ se, e somente se, $E(x_A(I)) \leq f(n).x^*(I)$, onde $f(n) \geq 1$. Se ¶ é um problema de maximização então A é $f(n)$-aproximado se e somente se $E(x_A(I)) \geq f(n).x^*(I)$, onde $f(n) \leq 1$ e $E(x_A(I))$ é o valor esperado para a solução $x_A(I)$.
Ao contrário dos algoritmos determinísticos, a partir dos mesmos dados, um algoritmo probabilístico pode obter soluções distintas. Um exemplo de aplicação desta técnica a problemas de otimização pode ser encontrado em [Agarwal & Sen 2001].

Metaheurísticas - Metaheurísticas são algoritmos aproximativos de propósito geral

constituídos de procedimentos iterativos que guiam heurísticas subordinadas combinando de forma inteligente diferentes conceitos que visam explorar e explotar de forma adequada o espaço de busca. Estes métodos fazem uso de heurísticas especializadas e de conhecimento prévio do problema para da melhor maneira possível, escapar de mínimos locais e varrer de forma mais eficiente o conjunto de soluções viáveis.
O desenvolvimento de uma heurística - que é um termo derivado do grego *heuriskein*, e significa descobrir ou achar - se dá, geralmente a partir de problemas específicos, e necessitam de adaptações para serem utilizadas em outros. Dentre as metaheurísticas mais conhecidas destacam-se:
- Algoritmos Genéticos;
- Busca Tabu;
- Colônia de Formigas;
- *Greedy Randomized Adaptive Search Procedures* - GRASP;
- *Simulated Annealing*.

Apesar de conseguir soluções próximas do ótimo as metaheurísticas não garantem a obtenção do ótimo global.

Neste trabalho serão utilizadas as metaheurísticas GRASP e Algoritmos Genéticos, que serão descritas em detalhes no capítulo 3. Informações detalhadas sobre outras metaheurísticas podem ser encontradas em [Viana 1998].

2.3 Aprendizagem por Reforço

Aprendizagem por Reforço - *AR*, em inglês *Reinforcement Learning-RL*, é um paradigma computacional de aprendizagem em que um agente aprendiz procura maximizar uma medida de desempenho baseada nos reforços que recebe ao interagir com um ambiente desconhecido. Sua utilização é recomendada quando não se dispõe de modelos *a priori*, ou quando não se consegue obter exemplos apropriados das situações as quais o agente aprendiz irá enfrentar. O agente tem como objetivo aprender de maneira autônoma uma política ótima de atuação, através da interação com ambiente. Tal aprendizagem se dá por experimentação direta, ou seja, sem a presença de um professor que o ensinaria através de exemplos.

Com o objetivo de facilitar o estudo de *AR* será apresentado na seção a seguir o formalismo matemático conhecido dos Processos de Decisão de Markov (PDM).

2.3.1 Processo de Decisão de Markov

Segundo Puterman, [Puterman 2005] um processo de decisão seqüencial pode ser denotado pela quíntupla $M = \{T, S, A, R, P\}$, cujos os componentes serão descritos a seguir.

T: Conjunto de Instantes de Decisão Em um processo de decisão seqüencial as decisões são tomadas em instantes de tempo denominados de instantes de decisão. Neste contexto o conjunto $T = \{1, 2, ...N\}$ $N \leq \infty$, representa o conjunto discreto

de instantes de decisão, que pode ser finito ou infinito. Em um problema de tempo discreto, o tempo é dividido em períodos, e cada instante de tempo $t \in T$ equivale ao início de um período, e corresponde ao momento específico da tomada de decisão [Puterman 2005].

S: Conjunto de Estados do Ambiente Em cada instante de decisão t o processo seqüencial assume um estado $s \in S$, que representa como o ambiente é percebido pelo agente decisor, naquele instante. O conjunto S pode ser discreto, contínuo, finito ou infinito

A: Conjunto de Ações Possíveis Em cada estado $s \in S$, o agente decisor deve escolher uma ação a de um conjunto de ações disponíveis no estado s, $A(s)$, sendo $A = \bigcup_{A(s)}, \forall s \in S$. É importante observar que S e $A(s)$ não variam com t e que $A(s)$ também pode ser discreto, contínuo, finito ou infinito..

R: Função de Recompensas ou Retornos Como conseqüência da escolha de uma ação $a \in A(s)$ a partir do estado s no instante de decisão t o agente decisor recebe uma recompensa $r_t(s,a)$. A escolha da ação a altera a percepção do agente em relação ao ambiente, levando-o a um novo estado s_{t+1} que o conduzirá ao novo instante de decisão $t+1$. A função $r_t(s,a)$ definida para $s \in S$ e $a \in A_s$ representa o valor da recompensa no instante t. Quando $r_t(s,a)$ é positivo é visto como lucro ou prêmio, quando negativo é visto como custo ou punição.

P: Função de Probabilidades de Transição O estado do ambiente no próximo instante de decisão $t+1$ é determinado pela distribuição de probabilidade $p_t(\cdot|s,a)$ $\forall t, sea$. A função $p_t(j|s,a)$, denominada de função probabilidade de transição, denota a probabilidade do processo está no estado $j \in S$ no instante $t+1$, quando o agente decisor escolheu a ação $a \in A(s)$ estando no estado s, no instante de tempo t.

Uma regra de decisão $d_t(s)$ especifica a ação a ser escolhida em um dado instante particular t. Uma política π permite ao agente decisor definir uma estratégia de escolha das ações em qualquer estado, ou seja, uma política $\pi = \{d_1, d_2, ..., d_N\}$, $N \leq \infty$, é um conjunto de regras de decisão. Resolver um problema de decisão seqüencial é determinar, *a priori*, uma política π^* que otimize a seqüência de retornos a ser obtido pelo agente.

Normalmente, em um processo seqüencial, a resposta de um ambiente em virtude da escolha de uma ação em um tempo $t+1$ qualquer, dependerá de todo o histórico de acontecimentos de tempo passado (anterior a $t+1$). Quando isso acontece diz-se que a dinâmica de comportamento do sistema só poderá ser definida através da distribuição completa de probabilidade:

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, ..., r_1, s_0, a_0\} \tag{2.6}$$

onde Pr representa a probabilidade de s_{t+1} ser s', a depender de todos os estados, ações e reforços passados $s_t, a_t, r_t, ..., r_1, s_0, a_0$.

Um Processo de Decisão Markoviano (*PDM*) é um processo de decisão seqüencial 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 es-

colhidas previamente. Quando isto ocorre, diz-se que o processo possui a propriedade de Markov, que pode ser expressa como:

$$P^{a}_{s,s'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{2.7}$$

Em um processo de decisão de Markov se o espaço de estados e ações for finito, ele será considerado um processo de decisão de Markov finito (*PDM* finito). A propriedade de Markov é importante para a aprendizagem por reforço, pois, ela possibilita que as decisões sejam tomadas em função apenas do estado atual, ou seja, o estado do processo no futuro depende apenas do estado do processo e da decisão escolhida no presente. Esta característica da propriedade de Markov torna possível a utilização de métodos incrementais, onde a partir do estado corrente pode-se obter soluções para cada um dos estados futuros.

2.3.2 Caracterizando um Problema de Aprendizagem por Reforço

Na aprendizagem por reforço pode-se identificar quatro elementos principais: uma política, uma função de reforço, uma função valor e idealmente um modelo do ambiente [Sutton & Barto 1998].

- **Política**: A política é responsável por definir o padrão de comportamento do agente aprendiz, em outras palavras, uma política π determina como o agente deve decidir por certas ações, em detrimento de outras. O agente faz isso através do mapeamento de estados s e ações a, em um valor $\pi(s,a)$ que corresponde a probabilidade do agente escolher a ação $a \in A(S)$ quando ele está no estado $s \in S$.
- **Função de Reforço e Retorno Total Esperado**: O reforço é um sinal do tipo escalar (r_{t+1}), que é percebido pelo agente no ambiente, logo que uma ação a tenha sido executada e uma transição de estado ($s_t \rightarrow s_{t+1}$) tenha ocorrido. Este sinal de reforço é definido com base em uma função que expressa o objetivo que o agente de aprendizagem deseja alcançar.
  O objetivo do agente é maximizar a quantidade total de reforços recebidos, denominado de retorno acumulado, o que nem sempre significa maximizar o reforço imediato a receber, mas maximizar o valor do retorno acumulado durante o processo como um todo. Assim, o que o agente de *AR* busca maximizar retorno total esperado, que é função da sequência de reforços recebidos até um tempo T final. No caso de um problema episódico, o retorno é um somatório simples denotado por:
  $$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T \tag{2.8}$$
  Em alguns casos, a interação entre o agente e o ambiente pode não terminar em um episódio[2], mas continua ilimitadamente. Neste casos $T \rightarrow \infty$ e também o valor do retorno esperado tenderá a infinito. Desta forma, um fator de desconto é introduzido para amortizar os valores futuros, estabelecendo assim, certo controle sobre o grau de influência que estes valores têm sobre o retorno total esperado. A expressão do

[2]Um episódio é uma sequência de estados que termina em um estado final.

retorno aplicando o fator de desconto γ será denotada pela equação 2.9, a seguir:

$$R_t = r_{t+1} + \gamma.r_{t+2} + \gamma^2.r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k.r_{t+k+1} \tag{2.9}$$

onde $0 \leq \gamma \leq 1$. Se $\gamma = 0$, o agente tem uma visão míope dos reforços, maximizando apenas os reforços imediatos. Se $\gamma = 1$, a visão do reforço abrange todos os estados futuros dando a mesma importância para ganhos neste momento e qualquer ganho futuro.
Para uma determinada tarefa, dada uma política π, como não se conhece *a priori* o valor do retorno total esperado R_T^{π}, pode-se então obter uma estimativa do mesmo utilizando aprendizagem por reforço.

- **Função Valor**: A função valor associa um valor a um estado (ou par estado-ação). O valor $V(s)$ é obtido a partir do reforço atual (recebido no estado s) e dos reforços futuros. O valor $V(s)$ representa uma estimativa do valor que o agente espera acumular ao longo do processo de aprendizagem, partindo do estado s. A função valor que considera apenas o estado s é denotada por $V(s)$ e é denominada função valor-estado, enquanto que a função valor que considera o par estado-ação (s,a), denotada por $Q(s,a)$ é denominada função valor estado-ação. Formalmente tem-se:

  - Função Valor Estado: A quantidade de reforços que o agente espera receber no futuro depende de quais ações ele irá escolher, logo a função valor é definida em relação a uma política específica. Considerando uma política π específica, a função valor estado $V(s)$ pode é denotado por:

$$V^{\pi}(s) = E_{\pi}\{R_t|s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s\} \tag{2.10}$$

    onde E_{π} é o valor esperado dado que o agente seguiu a política π, a partir de um estado $s_t = t$, no instante t. Uma política ótima de ações é aquela que maximiza o valor esperado, ou seja:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{2.11}$$

  - Função Valor Estado-Ação: Da mesma forma, considerando o par estado ação (s,a) pode-se ter $Q^{\pi}(s,a)$, denominada função valor estado-ação denotada por:

$$Q^{\pi}(s,a) = E_{\pi}\{R_t|s_t = s, a_t = a\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s, a_t = a\} \tag{2.12}$$

    que é semelhante a equação 2.10, considerando agora o reforço esperado para um estado s_t e uma ação a_t no instante t, assumindo que do momento $t+1$ em diante o comportamento do agente é caracterizado pela política π. De forma

análoga a função valor estado-ação ótima seria determinada por:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{2.13}$$

- **Modelo do Ambiente** Na aprendizagem por reforço o ambiente no qual o agente aprendiz está inserido deve ser, de alguma forma, pelo menos parcialmente observável. Caso toda informação relevante esteja disponível (conhecimento de $p_t(.|s,a)$ e $R_t(.|s,a)$) pode-se criar um modelo que imite o comportamento do ambiente, de forma que, dado um estado s_t e uma ação a_t, tal modelo possibilita predizer o próximo estado s_{t+1} e a próxima recompensa r_{t+1}. Infelizmente nem sempre é possível a construção de um modelo preciso do ambiente.

O processo iterativo de resolução de um problema de Aprendizagem por Reforço pode ser resumido da seguinte forma: O agente e o ambiente interagem em uma seqüência de passos de tempo discretos $t = 0,1,2,3,.....$ Em cada etapa de tempo t, existe uma representação do ambiente (um estado) $s_t \in S$ onde S é o conjunto de possíveis estados do ambiente. Com base neste estado s_t o agente aprendiz seleciona uma ação $a_t \in A(s_t)$, onde $A(s_t)$, é o conjunto de ações disponíveis no estado s_t. Como conseqüência da escolha da ação, o ambiente é, de alguma forma alterado, e esta alteração é comunicada ao agente através um sinal de reforço e da mudança para um novo estado s_{t+1} do ambiente.

Este processo interativo permite que o agente aprendiz possa definir, após um determinado número de experimentações, qual a melhor ação a ser executada em cada estado. Assim, o agente consegue aprender uma política ótima de atuação que maximize a estimativa do retorno total esperado representado pela função valor, independente do estado inicial do sistema. A figura 2.2 apresenta um esquema da interação do agente com o ambiente, conforme descrito anteriormente.

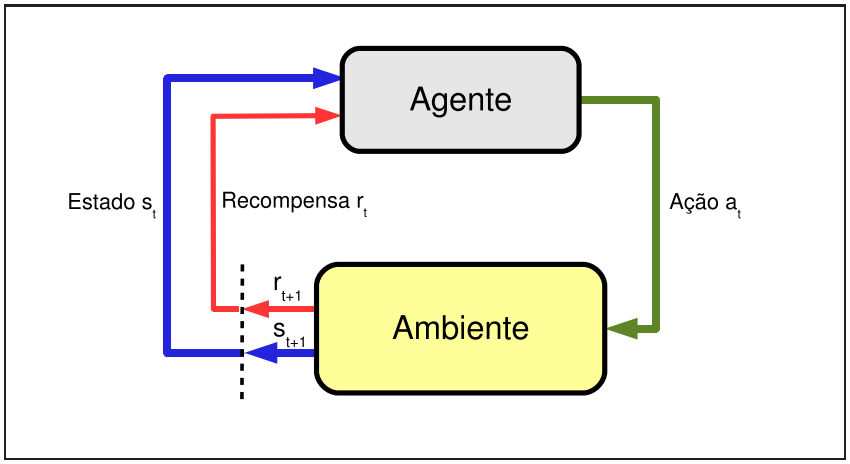


Figura 2.2: Esquema de interação entre um agente de aprendizagem por reforço e o ambiente (Figura traduzida de Sutton e Barto).

É importante lembrar que, na resolução de um problema de aprendizagem por reforço a meta maior é levar o agente a escolher a seqüência de ações que tendem a aumentar a

soma de valores de reforço, ou seja, o objetivo é encontrar uma política ótima, π^*, que maximize os sinais de reforço acumulados ao longo do tempo.

2.3.3 Visão Geral Sobre os Métodos de Resolução de AR

Apesar de alguns métodos apresentados nesta seção não estarem restritos a tarefas de aprendizagem por reforço, pois podem ser aplicados de forma mais ampla, como é o caso da Programação Dinâmica [Silveira & Joas 2002] e dos Métodos de Monte Carlo [Veach 1997], neste trabalho será considerada a abordagem adotada por Sutton e Barto [Sutton & Barto 1998], que os classifica como métodos elementares de solução para problemas de aprendizagem por reforço.

Programação Dinâmica

A Programação Dinâmica (*PD*) corresponde a uma coleção de algoritmos que podem obter políticas ótimas de atuação, com base na modelagem perfeita do ambiente como um *PDM*. Este "modelo perfeito do ambiente" pode ser conseguido a partir do conhecimento dos estados, ações, retornos e probabilidades de transição para todos os estados, ou seja, se faz necessário o conhecimento de:

$$\begin{aligned} P^a_{s,s'} &= Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \qquad \text{e} \\ R^a_{s,s'} &= E\{r_{t+1} = | s_{t+1} = s', s_t = s, a_t = a\} \end{aligned}$$

$\forall$ par $s, s' \in S$, e $a \in A(S)$.

Apesar dos métodos de programação dinâmica apresentarem grande importância teórica, eles são bastante limitados em aplicações práticas, tanto pela necessidade de uma modelagem precisa do ambiente, quanto pelo alto custo computacional por eles exigidos. O funcionamento dos métodos *PD* baseia-se na idéia de utilizar a função valor para gerar boas políticas. Esta idéia parte do princípio de que pode-se obter políticas ótimas a partir da função valor ótima V^*, ou Q^*, [Sutton & Barto 1998] as quais satisfazem as equações de otimalidade de Bellman, denotadas por:

$$V^*(s) = \max_a E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\} \tag{2.14}$$

$$= \max_a \sum_{s'} P^a_{s,s'} [R^a_{s,s'} + \gamma V^*(s')] \tag{2.15}$$

ou considerando a função valor estado-ação:

$$Q^*(s,a) = E\{r_{t+1} + \gamma \max_a Q^*(s_{t+1}, a') | s_t = s, a_t = a\} \tag{2.16}$$

$$= \sum_{s'} P^a_{s,s'} [R^a_{s,s'} + \gamma Q^*(s', a')] \tag{2.17}$$

São exemplos de algoritmos *PD*: Algoritmo de Avaliação de política, Algoritmo de Melhoria de Política, Algoritmo de Iteração de Política, Algoritmo de Iteração de valor

[Sutton & Barto 1998].

Métodos de Monte Carlo

Diferente dos métodos de programação dinâmica os Métodos de Monte Carlo (*MC*) não exigem o completo conhecimento do ambiente, mas apenas experiência, ou seja, necessitam apenas de amostras de seqüencias de estados, ações e recompensas de interações reais ou simuladas com o ambiente. Os métodos de Monte Carlo resolvem problemas de aprendizagem por reforço usando o valor esperado dos retornos totais observados, assim são indicados para tarefas episódicas de forma que possam garantir que tais retornos estejam disponíveis e bem definidos [Sutton & Barto 1998].

Os métodos de Monte Carlo trabalham de forma incremental de episódio em episódio, ou seja, a experiência amostral é dividia em episódios e não passo-a-passo. As mesmas idéias utilizadas nos algoritmos de *PD* são também aplicadas aos métodos de Monte Carlo, entretanto, para tal considera-se disponível apenas a experiência amostral, e não necessariamente o completo conhecimento do ambiente. Os métodos de Monte Carlo calculam a função valor $V^{\pi}(s_t)$ utilizando a o valor espera dos retornos observados R_t, após a visita ao estado s_t, de forma que, e a medida que os retornos são observados, a média deverá se aproximar do retorno real esperado.

Existem duas variantes do Método de Monte Carlo: a primeira, denominada de Método de Monte Carlo "qualquer visita" estima a função valor estado $V^{\pi}(s)$ para uma dada política π, como a média após todos as visitas ao estado s. A segunda variante, denominada de Método de Monte Carlo "primeira visita" estima a função valor estado como a média após a primeira visita ao estado s. Para ambas as variantes do método *MC*, se o número de visitas tende ao infinito a média convergirá para o valor de $V^{\pi}(s)$.

Métodos de Diferença Temporal

Os métodos de Diferença Temporal (*DT*) combinam as características dos métodos de Monte Carlo com as idéias da Programação Dinâmica, ou seja, como os métodos *MC* os métodos de diferença temporal podem aprender diretamente a partir de experiência, não exigindo um modelo completo da dinâmica do ambiente. Semelhante ao métodos *PD*, os métodos de diferença temporal atualizam as estimativas em parte com base em outras estimativas já aprendidas em estados sucessivos, sem precisar aguardar o final do episódio.

Ao contrário dos métodos de Monte Carlo os métodos de diferença temporal não aguardam o término de um episódio, atualizando imediatamente o valor de $V^{\pi}(s_t)$ após cada passo, ou seja:

$$V^{\pi}(s_t) = V^{\pi}(s_t) + \alpha[r_{t+1} + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)] \tag{2.18}$$

Uma das vantagens dos métodos de diferença temporal é que eles podem ser implementados de forma totalmente incremental, não precisando completar um episódio como os métodos *MC*, (além é claro, de não necessitar da descrição precisa do modelo do ambiente, de seus reforços e das distribuições de probabilidades entre os estados, como nos

métodos *PD*). Isso é possível por que os métodos *DT* atualizam as estimativas da função valor a partir de outras estimativas já aprendidas em estado sucessivos.

Os métodos de diferença temporal podem ser agrupados em duas categorias:

- Métodos *On-policy*, que atua fazendo avaliação e melhoria de uma política, que em seguida é utilizada pra tomar decisões.
- Métodos *off-policy*, que utilizam uma política para gerar o comportamento, que por sua vez é utilizado para avaliar ou melhorar outra política.

O algoritmo de aprendizagem por reforço utilizado neste trabalho - algoritmo *Q-learning* - é um dos métodos de diferença temporal *off-policy*, e será descrito em detalhes na seção a seguir.

2.3.4 Algoritmo *Q-learning*

Como já visto no início da seção 2.3, na aprendizagem por reforço existe a presença de um agente aprendiz que interage com o ambiente buscando estimar a ação de melhor resultado para cada estado, com o objetivo de estabelecer uma política ótima de atuação. Quando o agente não conhece o modelo de transição de estados do ambiente, torna-se impossível a utilização de programação dinâmica para a determinação da política ótima. Felizmente, nem todos os algoritmos de aprendizagem por reforço necessitam de uma modelagem completa do ambiente, ou seja, não necessitam conhecer a matriz de probabilidades de transição e valores esperados do sinal de reforço para todos os possíveis estados e ações do ambiente. Este é o caso das técnicas de aprendizagem por reforço baseadas em diferenças temporais.

Uma destas técnicas é o conhecido algoritmo *Q-learning* desenvolvido por Watkins [Watkins. 1989], é considerada uma das mais importantes contribuições em aprendizagem por reforço. O *Q-learning* é classificado como um método de diferença temporal *off-policy* uma vez que a sua convergência para valores ótimos de Q não depende da política que está sendo utilizada, ou seja, a função ação-valor Q se aproxima diretamente da função ação valor ótima $Q*$, através de atualizações dos pares estado-ação, que são feitas a medida que estes pares são visitados. A expressão de atualização da matriz dos *Q-valores* no algoritmo *Q-learning*, fundamenta-se na função ação-valor e é denotada por:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha_q[r_t + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)] \tag{2.19}$$

onde s_t é o estado atual, a_t é a ação realizada no estado s_t, e r_t é o sinal de reforço 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_q $(0 \leq \alpha_q < 1)$ é o coeficiente de aprendizagem [3]. A função $Q(s,a)$ é o valor associado ao par estado-ação (s,a) e representa quão bom é a escolha dessa ação na maximização da função retorno acumulado expresso pela equação 2.9.

Uma característica importante deste algoritmo é que a escolha das ações a serem executadas durante o processo de aproximação iterativa da função Q pode ser feita através

[3] O subscrito "q"é aqui utilizado para distinguir do parâmetro α utilizado na metaheurística GRASP.

de qualquer critério de exploração/explotação, inclusive de forma aleatória. Uma técnica bastante utilizada para tal escolha é a denominada exploração ε-gulosa, que consiste na escolha da ação associada ao maior valor de Q com probabilidade $1-\varepsilon+\frac{\varepsilon}{A(s)}$ e na escolha aleatória de qualquer outra ação com probabilidade $\frac{\varepsilon}{|A(s)|}$, onde $|A(s)|$ corresponde ao número de ações possíveis de serem executadas a partir de s, e ε é o parâmetro de controle entre gula e aleatoriedade.

O *Q-learning* foi o primeiro método de aprendizagem por reforço a possuir fortes provas de convergência. Watkins, [Watkins. 1989] mostrou que se cada par estado-ação (s,a) for visitado um número infinito de vezes, a função valor-ação $Q(s,a)$ irá convergir com probabilidade 1 para Q^* , usando α_q suficientemente pequeno. Sendo o valor ótimo de Q obtido, uma escolha ótima das ações pode ser feita de acordo com a expressão:

$$a^*(s) = arg\max_a Q(s,a) \tag{2.20}$$

De forma que o par estado-ação a ter seu valor atualizado será associado à ação escolhida de acordo com a política ε-gulosa:

$$\pi(s) = \begin{cases} a^*, & \text{com probabilidade } 1-\varepsilon \\ a_{Aleatoria} & \text{, com probabilidade } \varepsilon \end{cases} \tag{2.21}$$

No Algoritmo 2.3.1 é apresentado de forma simplificada o procedimento *Q-learning*.

Os detalhes sobre a implementação do algoritmo *Q-learning* utilizado neste trabalho serão discutidos na seção 4.3. Maiores informações sobre o algoritmo *Q-learning* podem ser encontrados em [Watkins. 1989].

Algoritmo 2.3.1 Algoritmo Q-learning

```
1:  procedure QLEARNING(r, α_q, ε, γ)
2:      Inicialize Q(s,a)
3:      repeat
4:          Inicialize s
5:          repeat
6:              Selecione a de acordo com a política ε-gulosa
7:              Observe os valores de r e s'
8:              Q(s,a) ← (1−α_q)Q(s,a) + α_q(r + γ max_{a∈A}(Q(s',a))
9:              s ← s'
10:         until Encontrar um estado final
11:     until Atingir NEp episódios
12:     return Q(s,a)                          ▷ Matriz dos Q-valores
13: end procedure
```

Apesar do critério de convergência do algoritmo *Q-learning* exigir a execução de um número infinito de episódios, na prática executa-se um número suficientemente grande de episódios (configura-se o número de episódios $(NEps)$, de acordo com o tamanho ou contexto da tarefa a ser aprendida). Nas aplicações deste trabalho o algoritmo *Q-learning*

não têm o propósito de encontrar a solução ótima para o *PCV*, e sim, construir para o mesmo soluções iniciais de boa qualidade, portanto, não existe a necessidade de executar o número muito elevado de episódios.

2.4 Conclusão

Neste capítulo foram apresentados os principais conceitos ligados a otimização combinatória, teoria da NP-completude e técnicas de resolução para problemas de otimização combinatória. Foram também apresentadas definições básicas sobre processo de decisão de Markov e Aprendizagem por reforço, com destaque para o algoritmo *Q-learning*, que é a técnica de *AR* utilizada neste trabalho e que será implementada nos métodos híbridos propostos no capítulo 4. Neste capítulo, foi também descrito formalmente o problema do caixeiro viajante simétrico, o qual terá sua modelagem como um problema de aprendizagem por reforço apresentada na seção 4.2.

Por uma questão de organização do texto, não foram apresentados neste capítulo detalhes sobre as metaheurísticas GRASP e Algoritmo Genético, os quais serão feitos de forma específica no capítulo 3. O texto apresentado não tem a pretensão de esgotar a temática aqui abordada, e sim tem como propósito fundamentar os conceitos necessários à melhor compreensão dos capítulos posteriores. Maiores detalhes sobre cada um dos temas descritos podem ser encontrados na bibliografia proposta.

Capítulo 3

As Metaheurísticas GRASP e Algoritmo Genético

3.1 Introdução

Nas áreas de engenharia, computação e pesquisa operacional, é comum a existência de uma séria de problemas reais de difícil solução, que requerem tarefas como:

- Determinar uma sequência ótima de processos de trabalho em uma cadeia de produção;
- Estabelecer o caminho mais curto entre vários pontos de demanda de um serviço;
- Determinar o melhor roteamento de um pacote de dados na internet;
- Encontrar uma designação ótima de trabalhadores a tarefas a serem realizadas, dentre outras.

Todos estas tarefas têm em comum o fato de que, na prática, a demanda pelos serviços é bem maior que a oferta, o que torna necessaria a administração ótima dos recursos disponíveis, a fim de minimizar os custo, diminuir os riscos e, na maioria das vezes, atender restrições de caráter cronológico.

Outro aspecto importante destes problemas, diz respeito às características estruturais dos mesmos, pois nas suas modelagens utilizam agrupamentos, ordenações ou designações de um conjunto de objetos discretos que satisfaçam determinadas restrições. Este aspecto contribui para que tais problemas apresentem uma alta complexidade computacional que os caracterizam como problemas NP-Árduos.

A aplicação de métodos exatos de resolução (Relaxações, Branch-and-Bound, Programação Dinâmica, dentre outros) são inviáveis em virtude do alto custo de processamento, e/ou pela dificuldade na elaboração de um modelo analítico e preciso das tarefas a serem realizadas. Uma alternativa à resolução de tais problemas é a utilização de metaheurísticas.

O termo metaheurística é obtido da anteposição do afixo “meta” (que significa “em um nível superior”) ao termo “heurística”[1] e foi utilizado originalmente por Fred Glover em 1986, em um texto introdutório sobre Busca Tabu [Glover 1986].

[1] O termo heurística tem origem na palavra grega *eureka* cujo significado está relacionado com o conceito de descobrir ou encontrar, e é vinculado a suposta exclamação de Arquimedes ao descobrir seu famoso princípio.

Segundo [Osman & Kelly 1996] metaheurísticas são métodos iterativos de geração de soluções que guiam uma heurística subordinada combinando de forma inteligente diferentes conceitos para explorar e explotar o espaço de busca, utilizando estratégias de aprendizagem para estruturar informações a fim de localizar eficientemente soluções próximas do ótimo.

Este trabalho propõe a utilizam do algoritmo *Q-learning* como estratégia de melhoria para as metaheurísticas GRASP e Algoritmo Genético, as quais serão descritas em detalhes nas seções a seguir.

3.2 GRASP

A metaheurística GRASP - *Greedy Randomized Adaptive Search Procedure* proposta por Feo e Resende, [Feo & Resende 1995] é um processo iterativo multipartida onde cada iteração é composta por duas fases: uma fase de construção e uma fase de busca local. Na fase de construção uma solução viável para o problema é criada e a fase de busca local tem como objetivo tentar melhorar a solução obtida na fase anterior.

As iterações GRASP são independentes, isto é, na iteração corrente não se leva em conta nenhuma informação das iterações anteriores. O critério de parada normalmente usado, é um número máximo de iterações. Ao final da execução do GRASP a melhor solução até então obtida, é a solução final.

Visando a melhor compreensão da metaheurística GRASP serão descritas em detalhes nas seções subsequentes o funcionamento das duas fases que a compõem.

Fase Construtiva

Na fase de construção do GRASP uma solução é construída um elemento[2] por vez, de forma iterativa, até que uma solução viável[3] para o problema seja concluída. Em qualquer passo intermediário deste processo de construção uma solução incompleta é dita uma solução parcial para o problema.

Cada inserção de um novo elemento na solução parcial é feita através da escolha aleatória em uma lista restrita de candidatos (*LRC*). Existem basicamente duas estratégias de construção de uma *LRC*, uma baseia-se na cardinalidade da *LRC*, outra na qualidade dos elementos que a compõem.

Para melhor compreender as duas estratégias de construção da *LRC*, considere um problema de minimização de uma função $f(s)$. Seja $g(c)$ uma função gulosa que denota o custo incremental associado à cada elemento c a ser incorporado à solução, de forma que em qualquer iteração GRASP, $G_{min} = min\{g(c)|c \in LC\}$ e $G_{max} = max\{g(c)|c \in LC\}$ são respectivamente, o menor e o maior custo incremental dos elementos, onde LC é a lista de todos os possíveis elementos candidatos a compor uma solução. Considere α um parâmetro de entrada que tem seu valor definido no intervalo $[0,1]$.

[2]Um elemento é cada item a compor uma solução em construção, por exemplo, no *PCV* cada uma das cidades a compor a rota que soluciona o problema é um elemento inserido na fase de construção do GRASP.

[3]Uma solução viável para o *PCV* é uma rota contendo todas as cidades sem a ocorrência de repetição

- **Estratégia de construção da *LRC* com base na cardinalidade da lista**

  1. Cria-se uma lista formada pelos indivíduos que ainda não fazem parte da solução parcial chamada Lista de Candidatos (*LC*).
  2. Calcula-se as contribuições de cada elemento $c \in LC$ para a solução através da função gulosa $g(c)$.
  3. Ordena-se a *LC* em ordem decrescente de benefício segundo a função $g(c)$, de modo que o seu primeiro elemento possa oferecer a maior contribuição no processo de minimização ($g(c) = G_{min}$), e o último, a menor ($g(c) = G_{max}$).
  4. incluí-se os p primeiros elementos da *LC* na *LRC*, sendo o valor de p é calculado por:

$$p = 1 + \alpha(N-1) \tag{3.1}$$

  Onde N é número total de candidatos. A figura 3.1 esclarece a atuação do parâmetro α sobre a lista de candidatos, no caso da criação da *LRC* baseada na cardinalidade.

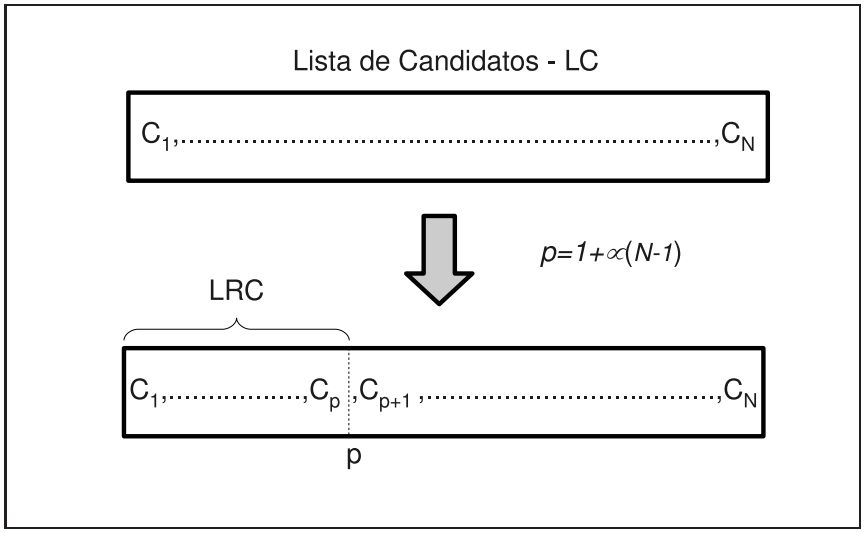


Figura 3.1: Atuação do parâmetro α sobre a lista de candidatos no algoritmo GRASP

- **Estratégia de construção da *LRC* com base na qualidade dos elementos**
  Na estratégia que baseia-se na qualidade dos elementos candidatos a compor a solução, a *LRC* é construída considerando os valores associados a cada elemento $c \in LC$, de forma que o conjunto dos elementos que irão compor a *LRC* é caracterizado por:

$$LRC = \{c \in LC | g(c) \leq G_{min} + \alpha.(G_{max} - G_{min})\} \tag{3.2}$$

  Observe que os elementos de *LC* que irão compor a *LRC* serão apenas aqueles que seu valor na função $g(c)$, respeitarem a condição imposta em 3.2.

É importante perceber que em ambas as estratégias de construção da *LRC*, se $\alpha = 0$ um algoritmo totalmente guloso será executado, já que só será possível a escolha do elemento da *LRC* que tem o menor custo incremental. Por outro lado, se $\alpha = 1$, a lista conterá

todos os possíveis candidatos, o que resultará em um algoritmo com escolha totalmente aleatória.

O aspecto probabilístico do GRASP decorre do fato de se escolher aleatoriamente um dos elementos da *LRC* e não necessariamente o melhor, a não ser no caso em que o critério de seleção se reduz à opção gulosa, como já mencionado. O controle entre solução aleatória e gulosa é feito pelo parâmetro α.

Uma vez que o elemento selecionado é incorporado à solução parcial, a lista de candidatos é atualizada (excluindo-se da *LC* o elemento que acabou de ser inserido na solução) e os custos incrementais são reavaliados recalculando-se $g(c)$. Este é o aspecto adaptativo do algoritmo.

É notável que a escolha do valor de α é um ponto importante na metaheurística GRASP, de tal forma que uma versão melhorada da metaheurística GRASP, denominada de *GRASP Reativo* pode ser concebida através da adoção de uma forma adaptativa da escolha do parâmetro α. Na seção 3.2.1 deste trabalho serão apresentados os detalhes desta versão melhorada da metaheurística GRASP.

Todas as versões da metaheurística GRASP (que utilizam *LRC*) implementadas neste trabalho têm a lista restrita de candidatos construída utilizando o método baseado na qualidade dos elementos.

Fase de Busca Local

Em um problema de otimização a vizinhança de uma solução s_i é um conjunto de soluções que podem ser alcançadas a partir de pequenas modificações realizadas em s_i. Os procedimentos que exploram sistematicamente uma determinada vizinhança no espaço de soluções de um problema são denominados procedimentos de busca local. Formalmente pode-se definir vizinhança como segue:

Seja S o espaço de busca para um problema p, seja s uma solução particular para p. A vizinhança é uma função que mapeia cada solução $s \in S$ para um subconjunto $N(s) \subseteq S$, de forma que um elemento qualquer de $N(s)$ é denominado vizinho de s. Todo vizinho $s' \in N(s)$ é alcançado a partir da solução s através de uma operação denominada movimento.

Associado ao conceito de vizinhança, existem ainda duas definições importantes: ótimo local e ótimo global. Seja $s \in S$ uma solução para um problema de minimização, s é denominada de mínimo local se:

$$f(s) \leq f(s') \quad \forall \quad s' \in N(s) \tag{3.3}$$

a solução s é denominada de mínimo global se:

$$f(s) \leq f(s') \quad \forall \quad s' \in S \tag{3.4}$$

A fase de melhoria da metaheurística GRASP consiste tipicamente de um procedimento de busca local que visa aperfeiçoar a solução obtida na fase de construção que nem sempre é um ótimo local. A busca local atua de forma iterativa através da substituição sucessiva da solução corrente pela melhor encontrada em sua vizinhança. Existem buscas

que trocam de soluções logo que uma solução de melhor qualidade é encontrada e outras que avaliam todos os vizinhos e somente depois substituem a solução corrente pela mais interessante até então encontrada.

Considerando um problema em que o objetivo é minimizar uma função f, sendo s_1 a solução corrente, diz-se então que uma solução s_2 é melhor que s_1 se $f(s_2) < f(s_1)$. Se não existir uma melhor solução na vizinhança, a solução corrente é considerada um ótimo local, e a busca termina.

Existem diversos algoritmos de busca local e sua classificação pode ser feita com base no tamanho da vizinhança explorada, ou seja, considerando o número de movimentos que pode ser utilizados para transformar uma rota em outra. Dentre tais algoritmos, um dos mais famosos é o *2-Opt* que foi proposto inicialmente por Croes [Croes 1958], ele altera uma rota eliminando duas arestas, quebrando assim esta rota em duas subrotas, e então reconecta as subrotas gerando uma rota alternativa. Em outras palavras o método *2-Opt* testa trocas possíveis entre pares de arestas, refazendo as conexões quando houver uma melhoria no roteiro e o processo termina quando não for possível mais realizar nenhuma troca que resulte em melhoria.

A figura 3.2 apresenta um exemplo do movimento que ocorre na busca local do tipo *2-Opt* para o problema do caixeiro viajante.

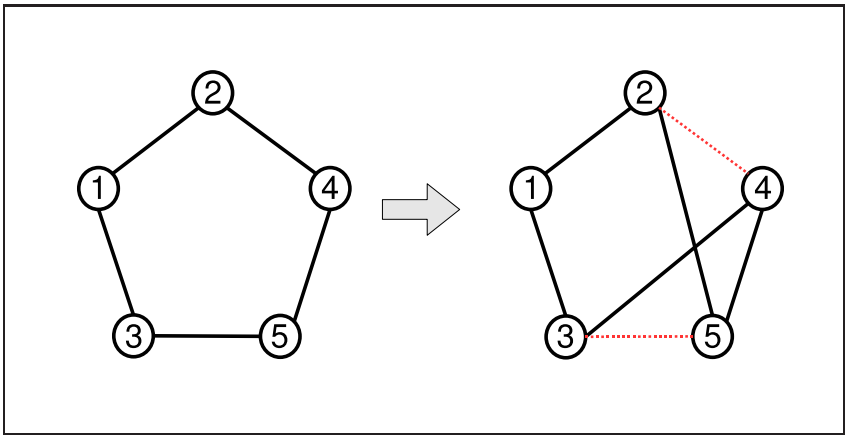


Figura 3.2: Exemplo de movimento típico da busca local *2-Opt* para o *PCV*

Para os casos de minimização os algoritmos de busca local são também chamados de métodos de descida. Neste contexto, um método de descida bastante interessante foi proposto por Mladenovic & Hansen (1997), denominado método de Descida em Vizinhança Variável (em inglês *Variable Neighborhood Descent* - VND). A diferença entre o VND e um algoritmo de busca local tradicional, é que ao invés de utilizar uma única estrutura de vizinhança, várias estruturas de vizinhança são utilizadas.

Os métodos de descida podem variar de acordo com a forma como analisa as soluções vizinhas. Normalmente um método de descida analisa todos os possíveis vizinhos em busca da melhor solução, uma forma variante do método de descida clássico pára ao encontrar a primeira melhoria (*first improvement method*), uma outra variante escolhe aleatoriamente uma solução vizinha que melhore a solução corrente.

Todas as metaheurísticas GRASP implementadas neste trabalho utilizam como algoritmo de busca local um método de descida *2-Opt* que parte de uma solução inicial qualquer, a cada passo analisa todos os possíveis vizinhos, e move-se para aquele vizinho que representa uma melhora no valor da função objetivo, o método pára quando um ótimo local é encontrado.

Segundo Feo & Resende (1995) o segredo do sucesso de um algoritmo de busca local depende da escolha adequada de vários aspectos, dos quais os autores destacam:

- A estrutura da vizinhança da solução;
- A eficiência da avaliação da função de custo;
- A qualidade da solução inicial.

Os autores acrescentam ainda que a fase construtiva do GRASP desempenha um papel muito importante com relação a este último aspecto, pois, o procedimento de busca local pode exigir um tempo exponencial se a busca partir de uma solução inicial qualquer (gerada aleatoriamente, por exemplo), entretanto, o tempo gasto pela busca local pode ser reduzido através do uso de uma fase de construção que gere soluções de boa qualidade. É fundamentado nesta afirmação que é proposto no Capítulo 4 deste trabalho a utilização do algoritmo *Q-learning* como construtor de soluções iniciais de boa qualidade para a metaheurística GRASP.

No algoritmo 3.2.1, é apresentado um procedimento básico para a fase construtiva do GRASP, o algoritmo 3.2.2, mostra uma versão simplificada da busca local, enquanto um procedimento para a metaheurística GRASP padrão é descrito no algoritmo 3.2.3.

Algoritmo 3.2.1 Algoritmo Guloso Aleatório

```
1:  procedure GULOSOALEATORIO(D,α)
2:      LC ← FuncAdapGulosa(D,LC)
3:      while "solução não estiver completa" do
4:          LRC ← ConstruirLRC(LC,α)
5:          e ← SelecElementoAleatorio(LRC)
6:          S_p ← S_p ∪ {e}
7:          LC ← FuncAdapGulosa(D,LC)
8:      end while
9:      return Retornar S_v como solução construída
10: end procedure
```

Na Tabela 3.1 são descritos os parâmetros utilizados nos pseudocódigos apresentados para a metaheurística GRASP.

Algoritmo 3.2.2 Algoritmo de busca local

```
1: procedure BUSCALOCAL(D, S_v)
2:     while "solução não for localmente ótima" do
3:         Encontrar uma solução S' ∈ Viz(S_v)        ▷ Busca na vizinhança de S
4:         if f(D,S') < f(D,S_v) then                 ▷ Caso de Minimização
5:             S ← S'
6:         end if
7:     end while
8:     return S                                       ▷ Solução localmente ótima
9: end procedure
```

Algoritmo 3.2.3 Algoritmo GRASP

```
1: procedure GRASP(D, α, Nmax)                        ▷ GRASP Padrão
2:     f(S*) ← +∞
3:     while i ≤ Nmax do
4:         S ← GulosoAleatorio(D, α)                  ▷ Fase construtiva
5:         S' ← BuscaLocal(D, S)
6:         if f(D,S') < f(D,S*) then
7:             S* ← S'
8:         end if
9:         i ← i + 1
10:    end while
11:    return S*                                      ▷ Melhor solução
12: end procedure
```

Tabela 3.1: Descrição dos parâmetros e variáveis dos algoritmos 3.2.1, 3.2.2 e 3.2.3

PARÂMETRO	ALGORITMO	DESCRIÇÃO
α	Guloso Aleatório	Regulador do nível de gula e aleatoriedade.
D	Guloso Aleatório	Matriz de distâncias do *PCV*
LRC	Guloso Aleatório	Lista restrita de candidatos.
LC	Guloso Aleatório	Lista de candidatos a compor uma solução.
e	Guloso Aleatório	Um elemento a compor uma solução.
S_p	Guloso Aleatório	Uma solução parcial
S_v	Busca Local	Uma solução viável
S'	Busca Local	Uma solução na vizinhança de S_v
$Nmax$	GRASP	Número máximo de iterações
S^*	GRASP	Melhor solução encontrada
$f(D,S)$	GRASP	Valor da função objetivo

A metaheurística GRASP apresentada na seção atual corresponde a uma versão tradicional. Como um dos propósitos deste trabalho é comparar o desempenho da metaheurística GRASP com versão híbrida que utiliza aprendizagem por reforço, é justo que tal comparação seja feita também utilizando uma versão melhorada desta metaheurística, que

será apresentada a seguir.

3.2.1 GRASP Reativo

Como já mencionado na Seção 3.2 a metaheurística GRASP utiliza um parâmetro α, $(0 \leq \alpha \leq 1)$ que controla os níveis de "gula" e aleatoriedade da fase construtiva. O que de fato o parâmetro α faz é determinar o tamanho da lista restrita de candidatos - *LRC*. Feo e Resende [Feo & Resende 1995] estudaram o efeito do valor de α na qualidade e na diversidade das soluções geradas durante a fase de construção do GRASP. Eles concluíram que valores selecionados de α que levam a uma *LRC* de tamanho muito limitado (valor de α muito próximo de zero) implicarão em soluções de qualidade muito próxima daquela puramente gulosa, com um esforço computacional proporcionalmente menor. Entretanto, a escolha de tais valores provocam uma baixa diversidade nas soluções construídas. Já uma escolha de α próxima da seleção aleatória leva a uma grande diversidade de soluções construídas, por outro lado, muitas destas soluções terão qualidade inferior, tornando mais lento o processo de busca local.

Recentemente um estudo elaborado por Prais & Ribeiro (2000) mostra que o uso de um valor fixo para o parâmetro α pode impedir a construção de soluções de melhor qualidade, que poderiam ser obtidas utilizando-se outros valores de α, ou seja, a execução do GRASP com α fixo e próximo da escolha puramente gulosa não gerava soluções suficientemente diversificadas para permitir que soluções ótimas sejam encontradas. Além disso, observaram que a execução do GRASP com outros valores de α (distintos do valor inicialmente arbitrado) permitia frequentemente encontrar a solução ótima. Com base neste resultado os autores propuseram um procedimento GRASP modificado denominado de GRASP Reativo, no qual o parâmetro α é auto-ajustado ao longo das iterações, em função da qualidade das soluções obtidas nas iterações precedentes.

No procedimento GRASP Reativo ao invés de se utilizar um valor fixo para o parâmetro α como no procedimento GRASP padrão, propõe-se que α seja aleatoriamente selecionado de um conjunto discreto $\Psi = \{\alpha_1, ..., \alpha_m\}$ contendo m valores predeterminados. A utilização de valores distintos de α em iterações diferentes permite a construção de *LRC* diferentes, eventualmente permitindo a geração de soluções distintas que não seriam construídas através da utilização de um único valor α fixo. Para efeito de ilustração considere o seguinte exemplo:

Seja $\alpha_1 = 0,1$; $\alpha_2 = 0,2$; ...; $\alpha_m = 1$, e p_i a probabilidade associada à escolha de α_i, $i = 1, ..., m$. Considere também os valores de $p_i = 1/m$, $i = 1, ..., m$. As probabilidades p_i $i = 1, ..., m$, deverão ser atualizadas periodicamente, com base nas informações coletadas durante a busca. Dentre as várias formas existentes para a atualização das probabilidades, os autores utilizam uma regra denominada de qualificação absoluta, denotada pela equação 3.5.

Seja $f(s^*)$ o valor da melhor solução até então encontrada, e M_i o valor médio das soluções encontradas utilizando-se $\alpha = \alpha_i$ na fase de construção. A distribuição de probabilidades será atualizada periodicamente a cada bloco de um certo número pré-determinado

de iterações realizadas, utilizando-se a seguinte expressão:

$$p_i = \frac{q_i}{\sum_{j=1}^{m} q_j}, \tag{3.5}$$

onde,

$$q_i = \left[\frac{f(s^*)}{M_i}\right]^{\delta}, \forall i = 1,2,...,m. \tag{3.6}$$

Observe que desta forma quanto mais adequado for o valor de $\alpha = \alpha_i$ (valor de M_i pequeno), maior será o valor de q_i e conseqüentemente, maior será o valor recalculado para probabilidade p_i. Neste processo valores de α que direcionam à melhores soluções têm maior probabilidade de serem selecionados nos blocos de iterações seguintes, e por conseguinte serão utilizados com maior frequência na fase de construção. Os autores sugerem o uso do expoente δ com o objetivo de penalizar as probabilidades de escolha dos coeficientes α_i cuja média M_i é muito maior que $f(s^*)$. Devido ao fato de não se utilizar um valor fixo para α e sim auto-ajustá-lo em função da qualidade das soluções encontradas durante a busca, o método recebe o nome de GRASP Reativo.

A metaheurística GRASP tem sido aplicada com sucesso aos mais variados tipos de problemas de otimização combinatória, tais como: Problema da Satisfabilidade - SAT [Resende & Feo 1996], Fluxo em Redes [Resende & Ribeiro 2005], Projeto de Redes de Computadores [Cancela et al. 2004], Problema Quadrático de Alocação - *PQA* [Pardalos et al. 1994], Logística de Manutenção de Poços de Petróleo [Lima Junior 2002], dentre outros.

3.3 Algoritmo Genético

Os Algoritmos Genéticos- *AG*, pertencem a uma família de modelos computacionais inspirados na teoria da evolução proposta por Darwin. Estes algoritmos modelam uma solução para um problema específico em uma estrutura de dados como a de um cromossomo e aplicam operadores que recombinam estas estruturas preservando informações críticas. Uma implementação do algoritmo genético começa com uma população (geralmente gerada de forma aleatória) de cromossomos. Estas estruturas são então avaliadas para gerar oportunidades reprodutivas de forma que, cromossomos que representam uma solução "melhor" tenham maiores chances de se reproduzirem do que os que representam uma solução "pior". A definição de uma solução melhor ou pior é tipicamente relacionada à qualidade da população atual.

Os algoritmos genéticos foram inicialmente desenvolvidos pelo professor John Holland, da Universidade de Michigan, nos Estados Unidos da América, em suas explorações dos processos adaptativos de sistemas naturais e suas possíveis aplicabilidades em projetos de *softwares* de sistemas artificiais. Eles foram formalmente introduzidas no seu livro *Adaptation in Natural and Artificial System* [Holland 1975].

Convém ainda salientar que a idéia dos Algoritmos Genéticos é mais antiga, como

reconhece o próprio Holland, referindo-se a trabalhos anteriores e a outras abordagens semelhantes. Em particular, ele menciona em seu livro os trabalhos de Rosenberg [Rosenberg 1967], Cavicchio [Cavicchio 1970], Hollstien [Hollstein 1971] e Frantz [Frantz 1972]. A pesquisa realizada por Holland e seus alunos na Universidade de Michigan tinha as seguintes metas:

- Explicar de forma rigorosa e abstrata o processo evolutivo dos sistemas naturais;
- Desenvolver um programa computacional que reproduzisse o importante mecanismo de solução de problemas empregado pelos sistemas biológicos.

Numa utilização mais abrangente do termo, um algoritmo genético é qualquer modelo baseado em população que utiliza operadores de seleção e recombinação para gerar novos pontos amostrais em um espaço de busca. Muitos algoritmos genéticos foram introduzidos por pesquisadores de uma perspectiva experimental. A maior parte deles tem interesse no algoritmo genético como ferramenta de otimização.

Os algoritmos genéticos trabalham com o conceito de população e reprodução, avaliando as soluções de uma população e selecionando-as de acordo com sua aptidão. A aptidão é calculada por uma função de avaliação específica de cada problema. Indivíduos são selecionados para reprodução que ocorre através de recombinação de partes de cada indivíduo da população, gerando novas soluções, que serão avaliadas e repetirão o processo, durante um número de gerações que é parâmetro do algoritmo.

A construção de um algoritmo genético consiste de quatro pontos:

1. Escolha da forma de representação do cromossomo, ou seja, a forma como as possíveis soluções do problema estudado podem ser codificadas, permitindo assim o processamento pelo algoritmo.
2. Geração da população inicial, geralmente de forma aleatória, p soluções são geradas, onde p é o tamanho da população, que representam um pequeno subconjunto do universo de soluções para o problema. o valor de p é um dos parâmetros de um *AG* que deve ser escolhido com critério, o valor muito pequeno para p faz com que o algoritmo converja rapidamente sem obter bom resultado, por outro lado, um valor muito grande para ele pode comprometer o tempo de execução do algoritmo.
3. Avaliação da função objetivo, neste estágio, cada indivíduo da população tem sua "aptidão" quantificada através da função:

$$f_j = f(S_j), \forall j = 1, \ldots, p \tag{3.7}$$

   onde S_j é uma solução viável (ou um cromossomo) pertencente a população inicialmente gerada.
4. Utilização dos Operadores genéticos na seguinte ordem:

   Seleção - Com base nos valores de f_j os indivíduos considerados mais fortes são selecionados e reproduzidos para a próxima geração em substituição aos indivíduos de menor aptidão. O operador de seleção não precisa necessariamente ser elitista, um método muito comum para este operador é a utilização de uma roleta "viciada" que possibilita que cada indivíduo seja copiado um número de vezes proporcional ao seu valor na função objetivo.

Cruzamento - Este operador permite que as estruturas genéticas da população sejam recombinadas, gerando assim estruturas diferentes das existentes na geração corrente. Para isso, o operador escolhe aleatoriamente dois indivíduos da população (os pais) e trocar partes de seu padrão genético, os cromossomos resultantes (os filhos) substituirão os pais na população. Pode-se utilizar um ou mais pontos de cruzamento. Um parâmetro do algoritmo denominado taxa de cruzamento, controla o percentual da população que sofrerá a ação do operador. A escolha da taxa de cruzamento deve se feita adequadamente, pois, um valor elevado irá introduzir um número muito grande de novos cromossomos na população, que poderá causar a substituição prematura de outros de boa qualidade. Por outro lado uma pequena taxa de cruzamento pode não produzir um número suficiente de novos descendentes.

Mutação - O AG, como todas as metaheurística, corre o risco de convergir e fica preso um ponto de mínimo local. Uma forma de tentar evita que isso aconteça, é utilizar uma operação denominada mutação que corresponde a uma pequena perturbação na configuração de um cromossomo, e que permite a exploração de outras áreas do espaço de busca. De forma análoga ao operado de cruzamento, a mutação exige um parâmetro de controle, a taxa de mutação, que deve ser cuidadosamente escolhido.

Os algoritmos genéticos diferem dos métodos tradicionais de otimização, principalmente em quatro aspectos:

1. Trabalham com uma codificação do conjunto de parâmetros e não com os próprios parâmetros (não opera com os indivíduos, opera com cromossomos, que codificam o indivíduo);
2. Trabalham com uma população e não com um único indivíduo;
3. Utilizam informações de custo ou recompensa e não derivadas ou outro conhecimento auxiliar;
4. Utilizam regras de transição probabilísticas e não determinísticas.

Algoritmos genéticos são bastante eficientes para busca de soluções ótimas, ou aproximadamente ótimas, em uma grande variedade de problemas, tais como: otimização de projeto de redes [White et al. 1999], roteamento de veículos [Thangiah 1995], programação de horários (*Timetable Problem*) [Fang 1994], escalonamento de tarefas [Whitley 2000], dentre outros. O pseudocódigo de um algoritmo genético padrão é exibido no algoritmo 3.3.1.

O algoritmo Genético proposto neste trabalho é uma versão modificada do algoritmo genético tradicional descrito nesta seção, ele terá sua população inicial gerada pelo algoritmo *Q-learning*. Seus operadores genéticos atuarão de forma cooperativa em um processo iterativo com o algoritmo *Q-learning*, de forma que possam tirar proveito da experiência aprendida durante o processo evolutivo, e também contribuam com informações úteis na atualização da matriz dos Q-valores. A Descrição detalhada no método híbrido proposto será apresentada na Seção 4.5 deste texto.

Na implementação de algoritmos genéticos é muito comum a utilização de codificação binária, entretanto, neste trabalho todos os algoritmos genéticos implementados utilizam

Algoritmo 3.3.1 Algoritmo Genético Tradicional

```
1:  procedure GENÉTICO(Tc, Tm, TPop, MaxG)
2:      Pop ← GerarPopulacao(TPop)
3:      Pop ← Avaliar(Pop)
4:      for i = 1...MaxG do
5:          Pop ← Selecao(Pop)
6:          Pop ← Cruzamento(Pop, Tc)
7:          Pop ← Mutacao(Pop, Tm)
8:          Pop ← Avaliar(Pop)
9:          Pop ← OrdenarPop(Pop)
10:     end for
11:     return A melhor solução em Pop
12: end procedure
```

um tipo de codificação em que cada cromossomo é um vetor de números inteiros e cada posição deste vetor corresponde a uma cidade na rota do *PCV*, desta forma, cada cromossomo da população corresponde a uma solução viável para o *PCV*. Um exemplo de cromossomo para a codificação utilizada seria:

$$C_k = [c_1, c_2, c_3, \ldots, c_i, \ldots, c_n], \forall k = 1, \ldots, TPop \tag{3.8}$$

onde cada c_i corresponde a uma cidade na rota, n é o tamanho da instância do *PCV* e $TPop$ é o tamanho da população.

Um outro detalhe importante da codificação utilizada, diz respeito a função de avaliação (ou de aptidão). Em todos os *AG* implementados foi utilizada a função de custo:

$$f(C_k) = \sum_{i=1}^{n} d_{ij}, \forall j = 1, \ldots, n \tag{3.9}$$

onde, d_{ij} corresponde a distância entre as cidades c_i e c_j (genes do cromossomo C_k). Assim, com base no valor da função $f(C_k)$ a população do *AG* é ordenada, de forma que os cromossomos com menor valor desta função correspondem aos indivíduos de melhor aptidão na população.

3.4 Conclusão

Neste Capítulo foram apresentadas formalmente as metaheurísticas GRASP (versão tradicional e reativa) e Algoritmo Genético. É importante salientar que existem na literatura outras versões da metaheurística GRASP (por exemplo, GRASP com Path-Relinking [Resende & Ribeiro 2005]) e de Algoritmos Genéticos, ([Kim et al. 2007] e [Chen et al. 2005]), muitas destas versões são métodos híbridas e possuem algum mecanismo adicional de melhoria, em relação as versões tradicionais.

O texto deste Capítulo não foi abrangente, no sentido de apresentar uma variedade de versões das metaheurísticas GRASP e Algoritmos Genéticos, e sim, teve como propósito

fornecer a fundamentação teórica necessária a compreensão dos métodos híbridos que serão propostos no Capítulo 4.

Nas Seções 4.4 e 4.5 serão apresentadas versões híbridos para a metaheurística GRASP e para o Algoritmo Genético, respectivamente. Os novos métodos proposto utilizarão o algoritmo *Q-learning* como estratégia inteligente para exploração e/ou explotação do espaço de soluções do *PCV*, com o objetivo de tornar mais eficiente a busca pela solução ótima.

Maiores detalhes sobre a metaheurística GRASP podem ser encontrados em [Feo & Resende 1995] e [Prais & Ribeiro 2000] e sobre algoritmos genéticos podem ser encontrados em [Whitley 1994] e [Randy & Haupt 1998].

Capítulo 4

Métodos Híbridos Propostos

4.1 Introdução

Conforme descrito na Capítulo 2 as metaheurísticas são algoritmos aproximativos que dependem de boas estratégias de exploração/explotação e que se baseiam em conhecimento prévio do problema, para guiar o processo de busca de uma solução eventualmente ótima, fugindo de mínimos locais.

Boas estratégias (ou boas heurísticas) são aquelas capazes de alternar adequadamente entre exploração e explotação, ou seja, são aquelas que dispõem de mecanismos que possibilitam manter o equilíbrio entre estes dois processos durante a busca pela solução ótima. Os métodos apresentados neste capítulos são denominados híbridos devido ao fato de utilizarem um algoritmo de aprendizagem por reforço bastante conhecido - o algoritmo *Q-learning* - como mecanismo de exploração/explotação para as metaheurísticas GRASP e Algoritmo Genético.

Todos os métodos apresentados neste capítulo serão aplicados à resolução do Problema do Caixeiro Viajante - *PCV* simétrico, desta forma é conveniente que antes da apresentação dos métodos seja apresentada sua modelagem como um Problema de Aprendizagem por Reforço. A seção a seguir tem esse propósito.

4.2 O Problema do Caixeiro Viajante Modelado como um Problema de Decisão Sequencial

O problema do caixeiro viajante pode ser descrito como um processo de decisão de sequencial, representado pela quíntupla $M = \{T, S, A, R, P\}$, pelo menos de duas formas diferentes. Pode-se por exemplo, considerar um modelo em que o conjunto de estados S é representado pelo conjunto de todas as possíveis soluções para o *PCV* [Ramos et al. 2003]. Para o objetivo deste trabalho, este modelo apresentaria o inconveniente da alta dimensionalidade do conjunto S, pois o *PCV* apresenta um número muito grande de possíveis soluções, no caso simétrico $(n-1)!/2$.

Uma outra alternativa para se modelar o *PCV* como um problema de decisão sequencial pode ser construída considerando que o conjunto S será formado por todas as cidades a compor uma solução para o *PCV*. Neste novo modelo a cardinalidade de S é equivalente

ao tamanho da instância do problema, diminuindo assim, o risco de S recair na "maldição da dimensionalidade". Neste trabalho o *PCV* será modelado como um problema de decisão sequencial com base nesta segunda alternativa.

Para melhor compreenssão do modelo proposto, considere um exemplo do *PCV* com 5 cidades ilustrado pelo grafo completo $G(V,E,W)$ da figura 4.1, onde V é o conjunto de vértices, E é o conjunto de arcos entre os vértices e W é o conjunto dos pesos associados a cada arco. No grafo $G(V,E,W)$, a_{12}, por exemplo, corresponde a ação de escolher visitar a "cidade" s_2 partindo de s_1, e os valores associados a cada arco (número entre parênteses) corresponde a distância entre as "cidades".

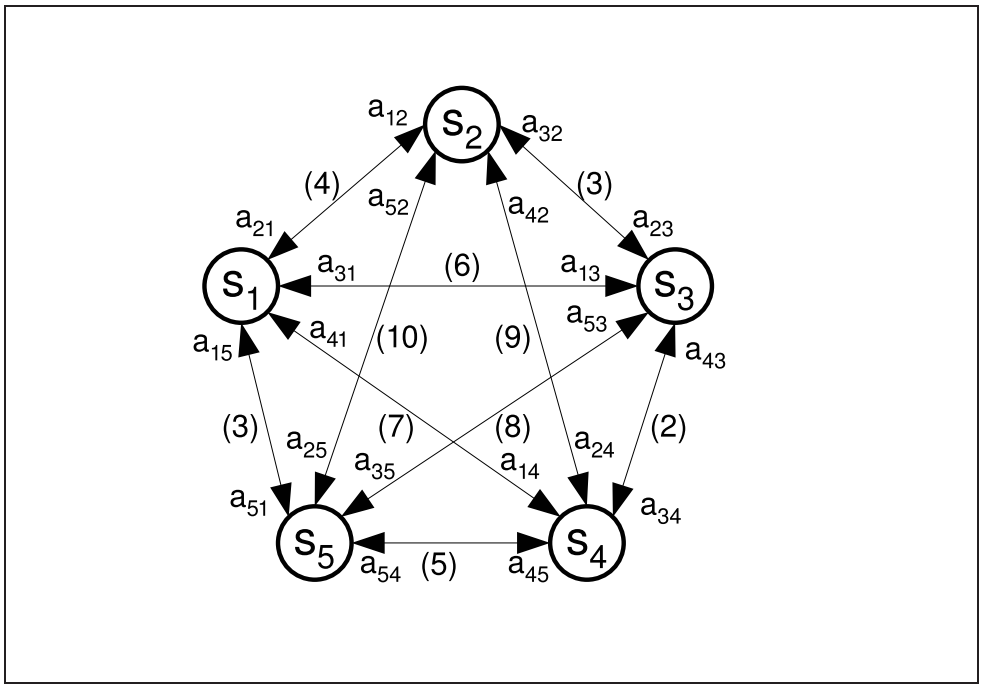


Figura 4.1: Grafo completo, exemplo do *PCV* com 5 cidades

Considerando o grafo $G(V,E,W)$ a quíntupla $M = \{T,S,A,R,P\}$ que representa um processo de decisão sequencial pode ser definida para o *PCV* da seguinte forma:

- T: Conjunto de instantes de decisão, denotado por $T = \{1,2,3,4,5\}$, onde a cardinalidade de T, corresponde à quantidade de cidades a compor uma rota para o *PVC*.
- S: Conjunto de estados, representado por $S = \{s_1,s_2,s_3,s_4,s_5\}$ onde cada um dos estados s_i, $i = 1,...,5$ corresponde a uma cidade[1] em uma rota do *PCV*.

[1]Deste ponto do texto em diante a expressão "cidade" ou "cidades" (entre aspas) será utilizada para indicar a associação de uma cidade do *PCV* com um estado (ou estados) do ambiente.

- A: Conjunto das possíveis ações denotado por:

$$\begin{aligned} A &= A(s_1) \cup A(s_2) \cup A(s_3) \cup A(s_4) \cup A(s_5) \\ A &= \{a_{12}, a_{13}, a_{14}, a_{15}\} \cup \{a_{21}, a_{23}, a_{24}, a_{25}\} \\ &\cup \{a_{31}, a_{32}, a_{34}, a_{35}\} \cup \{a_{41}, a_{42}, a_{43}, a_{45}\} \\ &\cup \{a_{51}, a_{52}, a_{53}, a_{54}\} \\ A &= \{a_{12}, a_{13}, a_{14}, a_{15}, a_{21}, a_{23}, a_{24}, a_{25}, a_{31}, a_{32}, \\ &\quad a_{34}, a_{35}, a_{41}, a_{42}, a_{43}, a_{45}, a_{51}, a_{52}, a_{53}, a_{54}\} \end{aligned}$$

  É importante destacar que em virtude da restrição imposta ao *PCV*, durante a construção de uma solução, algumas ações podem não estar disponíveis. Para melhor entendimento considere por exemplo a seguinte solução parcial para o *PCV*:

$$Sol_p : s_2 \rightarrow s_3 \rightarrow s_5$$

  na qual o processo de decisão encontra-se no instante de decisão 3 e no estado s_5. Neste caso, o conjunto de ações disponíveis seria $A(s_5) = \{a_{51}, a_{54}\}$ já que, pra evitar repetições na rota, as "cidades" s_2 (escolha da ação a_{52}) e s_3 (escolha da ação a_{53}) são proibidas.
- R: $S \times A \mapsto \mathbb{R}$: Função de retorno esperado. Para o *PCV* os elementos r_{ij} são calculados em função da distância entre as "cidades" s_i e s_j. O retorno deve refletir uma premiação que estimule a escolha da "cidade" s_j mais próxima de s_i. Como o *PCV* é um problema de minimização uma forma trivial de calcular a recompensa é considerá-la inversamente proporcional ao custo do deslocamento entre as cidades, ou seja:

$$r_{ij} = \frac{1}{d_{ij}}$$

  onde $d_{ij} > 0$ é um número real que representa a distância da "cidade" s_i à "cidade" s_j. Utilizando os pesos do grafo da figura 4.1, R seria representado por:

$$R = \begin{bmatrix} 0 & 1/4 & 1/6 & 1/8 & 1/3 \\ 1/4 & 0 & 1/3 & 1/9 & 1/10 \\ 1/6 & 1/3 & 0 & 1/2 & 1/8 \\ 1/7 & 1/9 & 1/2 & 0 & 1/5 \\ 1/3 & 1/10 & 1/8 & 1/ & 0 \end{bmatrix}$$

- P: $S \times A \times S \mapsto [0,1]$: Função que define as probabilidades de transição entre os estados $s \in S$, onde os elementos $p_{ij}(s_j|s_i, a_{ij})$ correspondem às probabilidades de se atingir a "cidade" s_j estando na "cidade" s_i e escolhendo a ação a_{ij}. Na modelagem adotada neste trabalho, tem-se sempre $p_{ij}(s_j|s_i, a_{ij}) = 1$, ou seja, o problema é completamente determinístico.

Considerando a quíntupla $M = \{T, S, A, R, P\}$ definida para o exemplo da figura 4.1, pode-se sem perda de generalidade, associá-la a um grafo $G(V, E, W)$ genérico, (com $|V| = N$) no qual seja possível encontrar um ciclo hamiltoniano de cardinalidade N.

Uma política ótima para o *PCV*, admitindo um grafo genérico $G(V,E,W)$ deverá estabelecer qual a seqüência de visitas a cada "cidade" s_i, $i = 1,2,...,N$, de modo que a soma dos retornos seja a melhor possível. Deve-se observar que neste caso o problema possui horizonte de tempo finito, conjuntos de estados e ações discretos e finitos.

4.2.1 Verificação da propriedade de Markov

Em um processo de decisão sequencial, tem-se normalmente que a transição do estado do ambiente em virtude da escolha de uma ação em um instante de decisão t qualquer, dependerá de todo o histórico dos acontecimentos ocorridos nos instantes anteriores a t. Isto significa que a dinâmica de comportamento do ambiente será definida através da distribuição completa de probabilidade [Sutton & Barto 1998]:

$$P^a_{ss'} = Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, ..., r_1, s_0, a_0\} \tag{4.1}$$

onde Pr representa a probabilidade de $s_{t+1} = s'$ e $r_{t+1} = r$, dados todos os estados, ações e reforços passados $s_t, a_t, r_t, ..., r_1, s_0, a_0$.

Entretanto, se o processo de decisão sequencial obedece a propriedade de Markov, a resposta do ambiente no instante $t+1$ depende somente das informações do estado s e ação a disponíveis no instante t, ou seja, a dinâmica do ambiente será especificada por:

$$P^a_{ss'} = Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\} \tag{4.2}$$

e esta característica qualifica tal processo com um processo de decisão de Markov - *PDM*.

No problema do caixeiro viajante modelado neste trabalho, a escolha de uma ação (decisão de qual cidade inserir na rota em construção) é feita a partir de um determinado estado s_t (a atual cidade na rota em construção) com base na distância desta cidade às demais. É importante observar que, para evitar a violação de restrição imposta pelo *PCV* (restrição de não repetição de cidades na rota), as ações que levam a estados já visitados não devem está disponíveis no conjunto de ações relativas ao estado s_t. Tome-se como exemplo duas soluções parciais para o problema representado na figura 4.1:

$$Sol_{p_1} : s_2 \rightarrow s_3 \rightarrow s_5$$
$$Sol_{p_2} : s_4 \rightarrow s_1 \rightarrow s_5$$

Em Sol_{p_1} tem-se que $A(s_5) = \{a_{51}, a_{54}\}$, enquanto em Sol_{p_2} tem-se que $A(s_5) = \{a_{52}, a_{53}\}$. Note-se que, ao se escolher a ação a_{51} em Sol_{p_1} (a_{53} em Sol_{p_2}), tem-se $p_{51}(s_1|s_5, a_{51}) = 1$ ($p_{53}(s_3|s_5, a_{53}) = 1$), independente da "história" (trajetória) percorrida. Neste contexto, o modelo do *PCV* adotado caracteriza-se como um processo de decisão de Markov, pois para a tomada de decisão no instante t toda informação necessária pode ser disponibilizada pelo estado s_t e pelo conjunto de ações disponíveis $A(s_t)$.

A propriedade de Markov é de fundamental importante para caracterização do problema do caixeiro viajante como uma tarefa de aprendizagem por reforço, pois, a conver-

gência dos métodos implementados neste trabalho depende da verificação desta propriedade no modelo proposto.

Entretanto, a restrição imposta ao modelo do *PCV* utilizado, gera uma interessante característica no processo de decisão de Markov associado ao mesmo, pois o conjunto de ações disponíveis $A(s)$, para estado s em cada instante de tempo t, pode variar durante o processo de aprendizagem, o que implica em alterações na definição das estratégias de escolha das ações em qualquer estado s_t. Into significa que durante a aprendizagem a política poderá ser *não-estacionária*. A figura 4.2 ilustra esta característica.

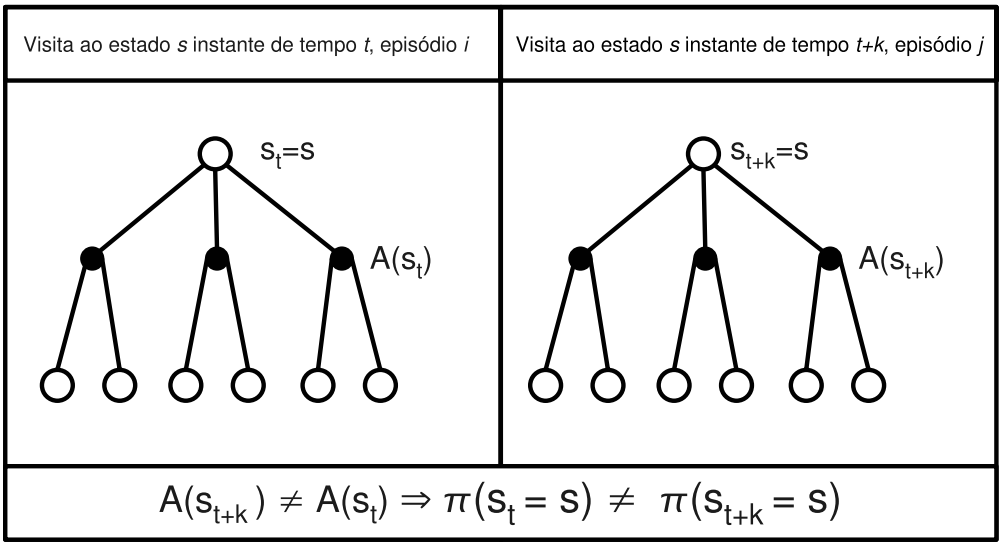


Figura 4.2: Alterações no conjunto de ações disponíveis durante o processo de decisão

O esquema apresentado na figura 4.2 pode ser interpretado da seguinte forma: Em cada episódio do processo de aprendizagem é construída uma rota para o *PCV*. Durante a construção da rota, em cada instante de decisão t visita-se um estado $s \in S$ e a escolha de uma ação $a \in A$ é realizada. Como o conjunto $A(s)$ das ações disponíveis para o estado s pode variar em cada episódio, em um episódio i o conjunto de ações disponíveis para o estado s_t no instante t pode ser diferente do conjunto de ações disponíveis no mesmo estado s no instante de tempo $t+k$ em um episódio j. Uma questão teórica existente em aberto nesta modelagem diz respeito ao fato de que, ao final do processo de aprendizagem, esta política não-estacionária irá convergir para uma política estacionária. Embora não investigado em detalhes neste trabalho, este comportamento pode ser observado nos exemplos tratados, conforme discutido na Seção 4.3.3.

4.3 Detalhes do Algoritmo *Q-learning* Implementado

Dado ao fato de todos os métodos propostos neste capítulo utilizarem o algoritmo *Q-learning*, está seção apresentará detalhes ligados a questões como: política de seleção de

ações e convergência do algoritmo quando aplicado ao problema proposto.

4.3.1 Políticas de Seleção de ações para o Algoritmo *Q-learning*

No processo de resolução de um problema de Aprendizagem por Reforço, uma política de seleção de ações tem como objetivo estabelecer o comportamento do agente aprendiz para que o mesmo alterne adequadamente entre o uso do conhecimento já adquirido e a aquisição de conhecimento novo, de forma a otimizar o processo de exploração/explotação do espaço de busca.

A idéia de experimentar mais de uma política de seleção de ações para o *Q-learning*, tem como meta verificar qual destas políticas é mais adequada para ser utilizada na implementação do métodos híbridos propostos.

Política ε-Gulosa

A política ε-gulosa escolhe a ação que possui o maior valor esperado, com probabilidade definida por $(1-\varepsilon)$, e de ação aleatória, com probabilidade ε. Matematicamente, dada a matriz dos Q-valores Q obtém-se a ação gulosa a^* para um estado s fazendo:

$$\begin{array}{rcl} a^* & = & \max\limits_{a \in A(s)} Q(s,a) \\ \pi(s,a^*) & = & 1-\varepsilon+\frac{\varepsilon}{|A(s)|} \\ \pi(s,a) & = & \frac{\varepsilon}{|A(s)|} \forall a \in A(s) - \{a^*\} \end{array} \tag{4.3}$$

onde $|A(s)|$ corresponde ao número de ações possíveis de serem executadas a partir de s, e ε é o parâmetro de controle entre gula e aleatoriedade. A restrição presente em 4.3 permite que o $Q-learning$ explore o espaço de estados do problema, e é uma das condições necessárias para que o mesmo encontre uma política de controle ótima.

Política ε-Gulosa Adaptativa

A política ε-gulosa adaptativa é semelhante a política $\varepsilon-gulosa$ já descrita anteriormente, ou seja, ela permite escolher a ação que possui o maior valor esperado, com probabilidade definida por (1 - ε), e a ação aleatória, com probabilidade ε. O que a diferencia, e também justifica o termo "Adaptativa" é que o valor de ε sofre um decaimento exponencial calculado por:

$$\varepsilon = \max\{v_i, v_f.b^k\} \tag{4.4}$$

onde, k é o contador de episódios do *Q-learning*, b é um valor próximo de 1 e $v_i < v_f \in [0,1]$. Desta forma inicialmente o algoritmo utilizará valores grandes para ε (próximos a v_f) e a medida que o valor de k cresce a escolha de ε é direcionada para valores menores (mais próximos a v_i). A idéia é permitir que inicialmente sejam feitas escolhas mais aleatórias e a medida que o número de episódios aumentem o aspecto guloso seja mais explorado.

Política Baseada na Contagem de Visitas

Nesta política a escolha das ações é feita baseada em uma técnica denominada Comparação de Reforço (Reinforcement Comparison) [Sutton & Barto 1998]. Nesta técnica a escolha das ações é feita com base no princípio de que ações seguidas de grandes recompensas devem ser preferidas em detrimento de ações seguidas de pequenas recompensas. Para definir o que significa uma "grande recompensa" é feita uma comparação com um nível de recompensa padrão denominado recompensa referencial.

A idéia da política aqui proposta é semelhante a técnica de comparação de reforço, entretanto, a escolha das ações preferidas é feita com base na contagem de visitas aos estados atingidos por tais ações, ou seja, os estados mais visitados indicarão as ações preferidas, em detrimento de ações que levam a estados com menor número de visitas. De posse da medida de preferência das ações, pode-se determinar a probabilidade de seleção das ações de acordo com a seguinte relação *Softmax*:

$$\pi_t(a) = P_r\{a_t = a\} = \frac{e^{p_{t-1}(a)}}{\sum_b e^{p_{t-1}(b)}} \tag{4.5}$$

onde $\pi_t(a)$ denota a probabilidade de se escolher a ação a no passo t e $p_t(a)$ denota a preferência da ação a no tempo t, e que é calculada por:

$$p_{t+1}(a_t) = p_t(a_t) + \beta(N_v(s,a_t)/NEp) \tag{4.6}$$

onde s é o estado atingido em conseqüência da escolha da ação a no passo t, $N_v(s,a_t)$ é o número de visitas ao estado s, NEp é o número total de episódios e $\beta \in [0,1]$ é um parâmetro de controle que pondera o nível de influência das ações preferenciais.

Comparação entre as Políticas Implementadas

Com o objetivo de comparar as políticas testadas foi realizado um experimento no qual foram utilizadas 10 instâncias do *PCV* disponíveis na biblioteca *TSPLIB* [TSPLIB 2008]. Para a realização experimento utilizou-se a seguinte metodologia:

- Os dados apresentados na tabela 4.1 foram gerados pela média de 30 execuções de três versões distintas do algoritmo *Q-learning* (uma versão para cada política), que utilizaram os mesmos valores para os parâmetros em comum.
- Para facilitar a comparação dos resultados foi feita uma normalização dos dados para plotagem, sendo que, para normalizar os dados da função objetivo foi utilizado como referência o valor ótimo conhecido para cada instância, enquanto que os dados do tempo de processamento foram normalizados pela média dos valores obtidos.

Os dados listados na tabela 4.1 mostram os resultados obtidos no experimento para cada versão do algoritmo *Q-learning*. Este experimento tem o propósito de comparar o desempenho do algoritmo *Q-learning* implementado utilizando três políticas distintas, as políticas "contagem de visitas", "ε-gulosa" e "ε-gulosa adaptativa" descritas na seção

Tabela 4.1: Comparação das políticas: Contagem de Visitas, ε-Gulosa e ε-Gulosa Adaptativa

Instância	Número	Contagem de Visita		ε-Gulosa		ε-Gulosa Adaptativa	
TSPLIB	Episódio	Valor	Tempo	Valor	Tempo	Valor	Tempo
gr17	300	2628,37	1,32	2499,56	0,82	2446,40	0,52
bays29	300	2967,83	2,23	2792,87	0,99	2690,70	0,80
swiss42	300	1804,27	3,26	1879,90	1,24	1830,70	1,19
gr48	300	7902,00	3,70	7630,83	1,36	7415,17	1,38
berlin52	300	11789,63	3,90	11376,50	1,48	10910,30	1,43
pr76	1000	178615,33	20,45	161144,33	6,36	156374,00	6,89
gr120	1000	12378,50	39,36	10938,87	9,20	10501,20	11,04
ch150	2000	12443,93	83,17	10122,90	20,67	9342,03	27,85
si175	5000	25721,53	280,45	26000,63	57,54	24642,17	82,10
a280	5000	5206,73	1388,33	4056,17	98,01	3748,80	139,46

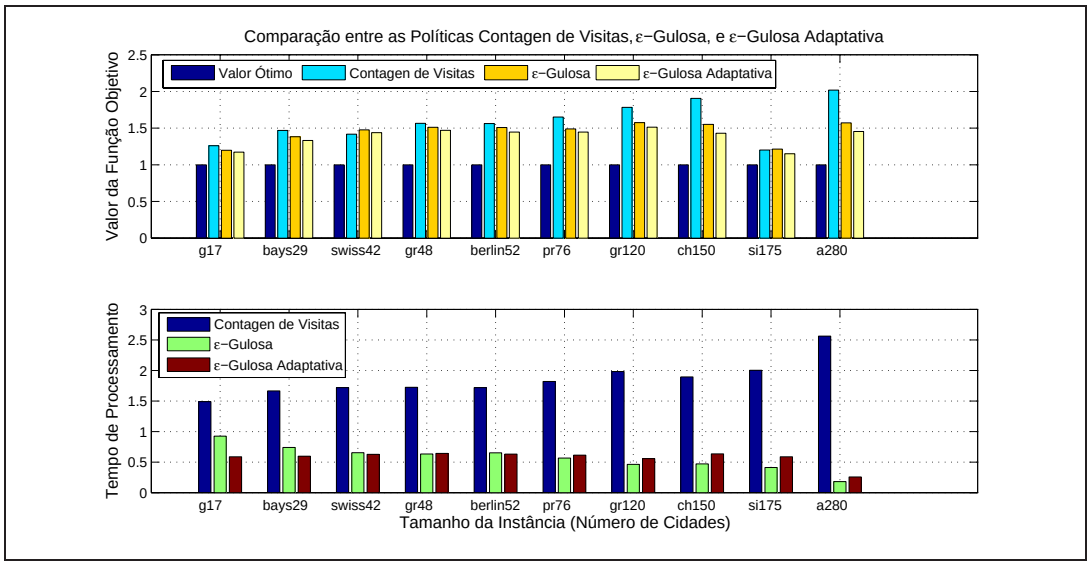


Figura 4.3: Comparação das Políticas testadas para o *Q-learning* implementado

4.3.1. Na mesma tabela pode-se também observar que a quantidade de episódios variou em função do tamanho das instâncias. O experimento realizado não tem o objetivo de comparar os resultados obtidos com as diferentes versões do algoritmo *Q-learning* com os valores ótimos das 10 instâncias do *PCV* utilizadas, e sim verificar, dentre as três políticas implementadas qual a de melhor desempenho. A figura 4.3 apresenta uma comparação do desempenho das políticas descritas anteriormente.

Ao analisar os resultados obtidos para as três políticas testadas, considerando simultaneamente os fatores: valor da função e tempo de processamento, percebe-se que a política

ε-gulosa adaptativa foi a que obteve melhor desempenho e portanto, será a política utilizada na versão do algoritmo *Q-learning* implementado nos métodos híbridos propostos neste Capítulo.

4.3.2 Determinação da Função de Recompensa

Na aprendizagem por reforço um agente aprendiz tem como objetivo maximizar o total de recompensa recebida ao longo do processo. Para atingir este objetivo o agente necessita mensurar quão bom é escolher uma determinada ação a partir do estado corrente por meio de um valor numérico, isso é feito através da função de recompensa. Assim, para cada problema específico se faz necessaria a definição da função de recompensa. Nesta seção serão discutidos os detalhes da função de recompensa utilizado no algoritmo *Q-learning* para o métodos propostos.

A função de recompensa dada pela Equação 4.2 estabelece de forma trivial a recompensa imediata como sendo inversamente proporcional a distância de uma cidade de partida c_i à cidade de destino c_j. Esta forma de estabelece a recompensa imediata apresenta a inconveniência de, em casos de valores de distâncias muito próximos, ocorrer casos de empate na escolha dos Q-valores $Q(s,a)$. Uma forma de resolver este inconveniente é ponderar o inverso da distância por algum valor que permita distinguir os casos de empate.

O cálculo da função de recompensa aqui proposto faz a ponderação do inverso da distancia entre duas cidades (dois estados distintos) utilizando a contagem de visitas aos estados do ambiente. Desta forma as ações que levam a estados mais frequentemente visitados serão premiadas com recompensa imediata maior. Isso pode ser feito utilizando a seguinte equação:

$$R(s,a) = \frac{1}{d_{ij}} * N_v(s,a) \tag{4.7}$$

onde $\frac{1}{d_{ij}}$ é o inverso da distância entre as cidades c_i e c_j (sendo a cidade c_i representada pelo estado s, e a cidade c_j representada pelo estado a ser atingido em conseqüência da escolha da ação a), e $N_v(s,a)$ o número de visitas ao estado corrente.

4.3.3 Análise de Convergência do *Q-learning*

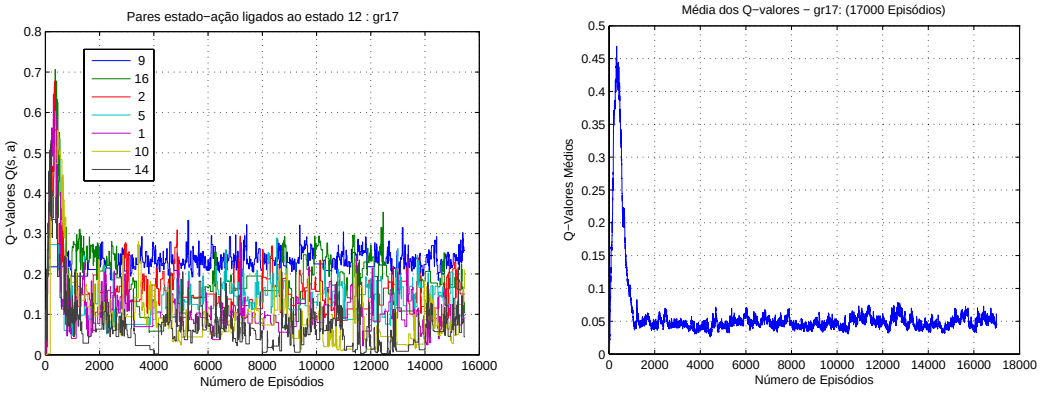
O algoritmo *Q-learning* é um dos métodos de aprendizagem por reforço que possui forte prova de convergência [Watkins. 1989]. Entretanto, quando trata-se de modelos baseados em processos de decisão de Markov não estacionário, esta prova não se aplica.

O teste de convergência aqui proposto consiste de investigar numericamente o comportamento do algoritmo *Q-learning*, quando aplicado ao problema do caixeiro viajante modelado como um processo de decisão de de Markov. O teste foi elaborado aplicando a seguinte metodologia:

- Executa-se o algoritmo *Q-learning* aplicado a instâncias do caixeiro viajante, partindo de um estado s_j escolhido aleatoriamente;
- Armazena-se os pares estado-ação $Q(s,a)$ e o número de visitas a cada estado s_i, $N(s_j,s_i)$, $\forall i = 1,2,...n, i \neq j$, onde n é o tamanho da instância;

- Calcula-se em cada episódio a média dos Q-valores e armazena-se em uma matriz *QMedia*, onde as linhas representam as media dos Q-valores de cada episódio e cada coluna representa um estado do problema.
- Gera-se os gráficos dos valores de $Q(s,a)$ e *QMedia* armazenados durante todo o processo.

O teste foi realizado com as instâncias *gr*17, *gr*42, *berlin*52 e *pr*76, executando-se $1000 * n$ episódios, sendo n o tamanho das instâncias, $\alpha_q = 0.8$ e $\gamma = 1$, e o valor de ε adaptativo de acordo com a política ε-gulosa adaptativa definida na seção 4.3.1. As Figuras 4.4, 4.6, 4.5 e 4.7, apresentam os resultados com os Q-valores (7 melhores resultados) e a média deste, obtidos para as instâncias citadas.



(a) Q-Valores

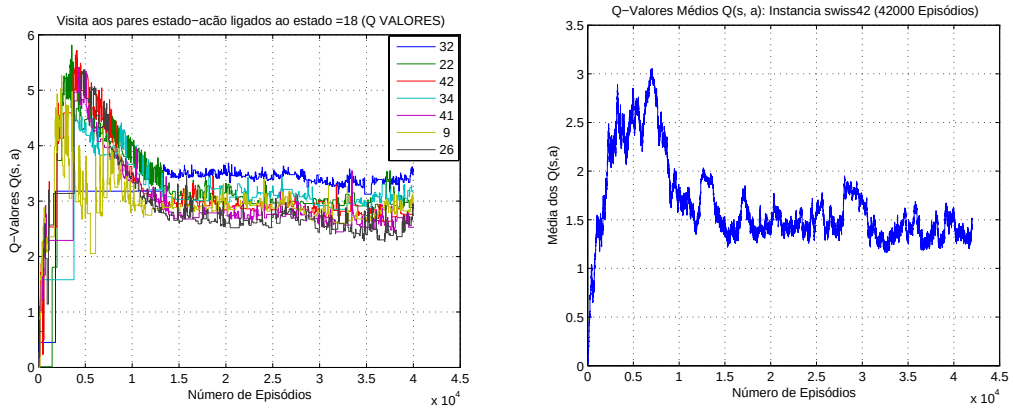
(b) Média dos Q-Valores

Figura 4.4: Convergência do Q-learning - Instância TSPLIB: gr17

Observando os gráficos das médias para cada um das figuras anteriores pode-se perceber que ao longo do tempo as curvas tendem a se estabilizarem em torno de um determinado valor. Os gráficos demonstram também uma diferença de comportamento entre as instâncias, tendo a instância *pr*76 um gráfico com uma curva de melhor comportamento. Essa diferença é justificada pela natureza do problema, pois no *PCV* as diversas instâncias apresentam diferentes níveis de dificuldade em sua otimização, o que poderá implicar em tarefas de aprendizagem com diferentes graus de dificuldade na convergência.

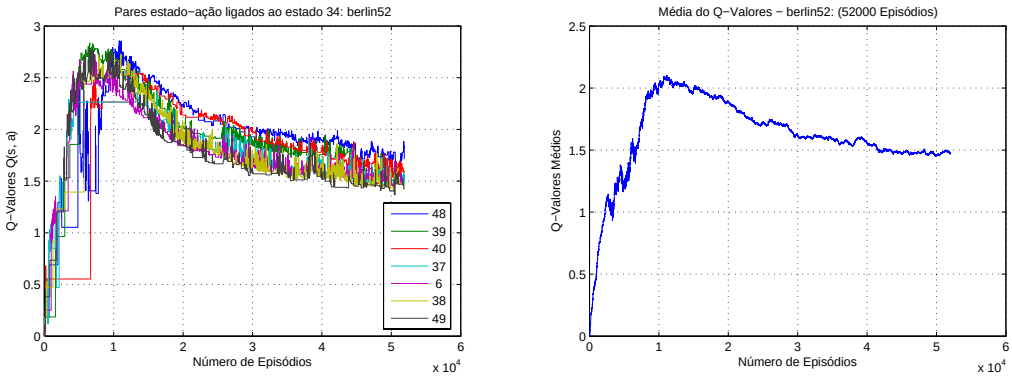
A diferença entre os graus de dificuldade de convergência das instâncias testadas pode ser notada pela observação do valor da variância de cada uma delas, sendo que as instâncias menores apresentaram maior valor para variância. Apesar de para as diferentes instâncias, as curvas não apresentarem comportamento idêntico, de forma geral o experimento demonstrou que existe uma tendência de que os valores de utilidade $Q(s,a)$ converjam, em média, para uma política ótima de ações.

Na próxima seção serão apresentadas as metaheurísticas GRASP e Algoritmo Genético Híbridos, que utilizam como estratégia de exploração/explotação o algoritmo *Q-learning* descrito nesta seção. Os novos métodos propostos serão aplicados ao problema



(a) Q-Valores (b) Média dos Q-Valores

Figura 4.5: Convergência do Q-learning - Instância TSPLIB: swiss42



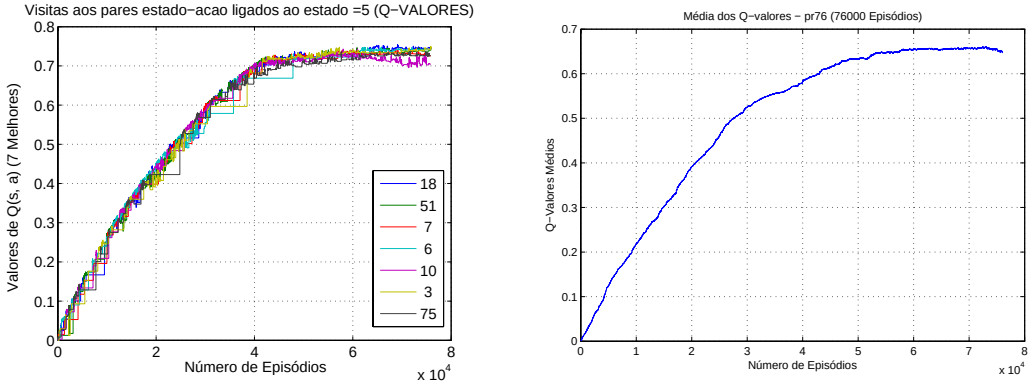
(a) Q-Valores (b) Média dos Q-Valores

Figura 4.6: Convergência do Q-learning - Instância TSPLIB: berling52

do caixeiro viajante simétrico modelado como um *PDM* não estacionário conforme já descrito na seção 4.2.

4.4 O Método GRASP-Learning

No caso específico da metaheurística GRASP os processos de exploração e explotação acontecem em momentos bem distintos, a fase construtiva explora o espaço de soluções viáveis, enquanto a busca local melhora a solução construída na fase inicial explotando sua vizinhança. Apesar desta clara delimitação de papéis, as duas fases da metaheurística GRASP trabalham de forma conjunta, e o bom desempenho do algoritmo de busca local está vinculado à qualidade da vizinhança escolhida e depende fortemente da qualidade da



(a) Q-Valores

(b) Média dos Q-Valores

Figura 4.7: Convergência do Q-learning - Instância TSPLIB: pr76

solução inicial [Feo & Resende 1995].

Nesta seção será apresentado um método híbrido que utiliza o algoritmo *Q-learning* na fase construtiva da metaheurística GRASP. Na metaheurística GRASP tradicional as iterações são independentes, ou seja, na iteração atual não se faz uso da informação obtida nas iterações passadas [Fleurent & Glover 1999]. A idéia básica do método aqui proposto é fazer uso das informações contidas na matriz dos Q-valores, como uma espécie de memória que possibilite repetir as boas decisões tomadas em iterações anteriores, e evitar aquelas que não foram interessantes, facilitando assim o processo de exploração/explotação.

Com base em resultados obtidos utilizando o algoritmo *Q-learning* isoladamente na resolução de pequenas instâncias do *PCV*, percebe-se que a abordagem proposta pode melhorar significativamente o desempenho da metaheurística, no que diz respeito a localização do ótimo global. O método híbrido proposto denominado *GRASP-Learning* será descrito a seguir.

No método *GRASP-Learning* o algoritmo *Q-learning* será utilizado como construtor de soluções iniciais para a metaheurística GRASP, de forma que, a cada iteração do algoritmo seja construída uma solução viável de boa qualidade utilizando as informações contidas na matriz dos Q-valores, desta forma, a cada iteração GRASP a matriz dos Q-valores poderá se utilizada como uma espécie de memória adaptativa que permite que a experiência passada se seja utilizada no futuro, o termo adaptativa refere-se ao fato de, a cada iteração novas informações serem inseridas através da atualização da matriz Q, influenciando assim na fase construtiva da próxima iteração.

O algoritmo *Q-learning* será então utilizado como um algoritmo guloso-aleatório onde o controle entre "gula" (Exploração) e "aleatoriedade" (Explotação) será feito pelo parâmetro ε da regra de transição definida na equação 4.4 e utilizará a equação 4.7 para determinar a matriz de recompensas

De forma resumida o algoritmo *Q-learning* proposto para a fase construtiva do GRASP, aplicada ao *PCV* simétrico, realiza o seguinte procedimento:

Inicialmente a tabela de valores estado-ação Q (matriz dos Q-valores) recebe valor zero para todos os intens; em seguida um índice da tabela Q é sorteado aleatoriamente para iniciar o processo de atualização da mesma, este índice passa a ser o estado s_0 para o *Q-learning*. A partir do estado s_0, utilizando a regra de transição ε-gulosa adaptativa, pode-se obter o estado s_1 através de uma das seguintes opções:

- Aleatoriamente, sorteando-se uma nova cidade para a rota;
- Utilizando o argumento máximo de Q em relação ao estado anterior s_0.

Dispondo dos estados s_0 e s_1, inicia-se o processo iterativo que atualiza a tabela Q utilizando a equação 2.19. A possibilidade de um estado de menor argumento ser escolhido, através da aleatoriedade, garante a capacidade do método explorar outras regiões de busca. Após um número máximo de episódios, obtém-se a tabela Q da qual será retirada a rota para o *PCV*. A construção de uma solução para o *PCV*, após a obtenção da tabela Q é feita da seguinte forma:

- Copia-se os dados da matriz Q para uma matriz auxiliar Q_1;
- Sorteia-se um índice l que indique a linha inicial da tabela Q_1 (correspondente à cidade que vai iniciar a rota);
- Partido da linha l escolhe-se o maior valor na mesma, o índice c da coluna deste valor corresponderá à próxima cidade a compor a rota;
- Atribui-se valor nulo a todos os valores da coluna c em Q_1, a fim de garantir a não repetição da mesma cidade na rota, respeitando assim a restrição básica do *PCV*;
- Continua-se o processo enquanto a rota não estiver completa.

A cada iteração do GRASP a matriz Q é atualizada pela execução do algoritmo *Q-learning*, ou seja, no final de *Nmax* iterações o algoritmo *Q-learning* terá executado $Nmax * NEp$ episódios, onde NEp denota o número de episódios executados em cada iteração. O fato da matriz dos Q-valores ser atualizada a cada iteração do GRASP permite que ao longo do processo de busca a qualidade das informações coletadas seja melhorada. A figura 4.8 apresenta uma visão geral do método *GRASP-Learning* implementado.

Com relação à fase de busca local, o método *GRASP-Learning* não sofre nenhuma alteração, ou seja, utiliza o mesmo algoritmo do método GRASP tradicional, que neste trabalho foi implementado um método de descida *2-Opt*, que é uma técnica de busca local que move-se para uma solução vizinha somente se esta representar uma melhoria na função objetivo. O algoritmo 4.4.1 apresenta, de forma simplificada, o pseudo código do método *GRASP-learning* conforme proposto:

Ao se examinar o pseudo-código do método *GRASP-Learning* percebe-se que as modificações significativas estão presentes nas linhas 3 e 5. Na linha 3 a função *GeraRecompensa* utiliza a matriz de distâncias D para gerar a matriz de recompensas R, tendo como base a Equação 4.7. Na linha 5 a função *PegarRota* executa a atualização do algoritmo *Q-learning* retornando uma solução viável S para o *PCV*, e a matriz dos Q-valores atualizada Q, que será utilizada na próxima iteração. Os demais aspectos da metaheurística $GRASP - learning$ são idênticos à metaheurística GRASP tradicional.

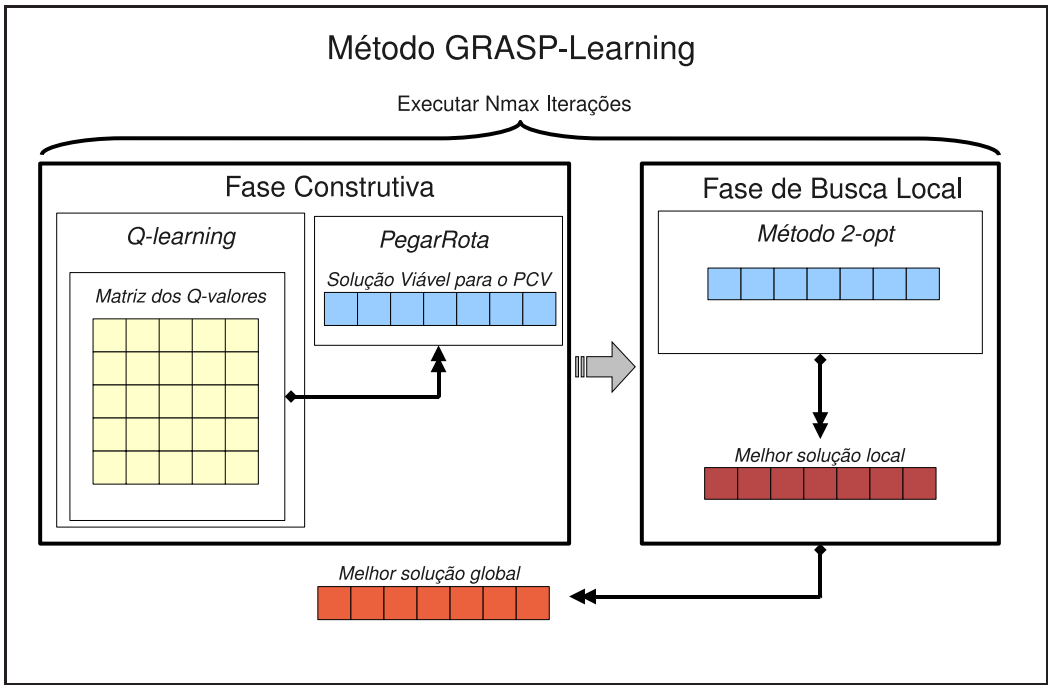


Figura 4.8: Visão geral do método *GRASP-Learning* implementado

4.5 O Algoritmo Genético-Learning Cooperativo

O algoritmo genético híbrido proposto nesta seção recorre à aprendizagem por reforço com o objetivo de construir melhores soluções para o problema do caixeiro viajante simétrico. O foco principal do método é utilizar o algoritmo *Q-learning* para auxiliar os operadores genéticos na difícil tarefa de se obter o equilíbrio entre explorar e explotar o espaço de solução do problema. O novo método propõe a utilização do algoritmo *Q-learning* como gerador de uma população inicial de alta aptidão com boa taxa de diversidade, e que também atue de forma cooperativa com os operadores genéticos.

No processo de aprendizagem o algoritmo *Q-learning* pondera entre usar o conhecimento já obtido e escolher novos espaços de busca ainda não explorados, ou seja, ele apresenta características de um algoritmo guloso-aleatório, guloso ao fazer uso do máximo valor de $Q(s,a)$ para selecionar a ação a que contribua com maior retorno no estado s e aleatório ao usar uma política de escolha de ações que possibilite eventuais visitas a todos os demais estados do ambiente, este comportamento por parte do *Q-learning* lhe confere atributos capazes de produzir consideráveis melhorias ao algoritmo genético tradicional.

Como já mencionado o algoritmo genético aqui proposto tem sua população inicial gerada pelo algoritmo *Q-learning*. A população é gerada executado-se o *Q-learning* um determinado número de episódios (NEp). Em seguida, utilizando a matriz Q (matriz dos q-valores) gera-se Tp cromossomos, onde Tp é tamanho da população. O critério de escolha na criação de cada cromossomo é o valor de $Q(s,a)$ associado a cada gene, de forma que, genes que têm maior valor de $Q(s,a)$ serão escolhidos prioritariamente na

Algoritmo 4.4.1 Algoritmo GRASP-Learning

```
1: procedure GRASP-LEARNING(D, α_q, ε, γ, Nmax)
2:     f(S*) ← +∞
3:     R ← GeraRecompensa(D)
4:     while i ≤ Nmax do
5:         [S, Q] ← PegarRota(Qlearning(R, α, ε, γ, Q))
6:         S' ← BuscaLocal(S, Viz(S))
7:         if f(S') < f(S*) then
8:             S* = S'
9:         end if
10:        i ← i + 1
11:    end while
12:    return S*                                   ▷ Melhor Solução
13: end procedure
```

composição do cromossomo[2].

Além de gerar a população inicial o algoritmo *Q-learning* atua em um processo de cooperação com os operadores genéticos da seguinte forma:

- Em cada nova geração a melhor solução S_M obtida pelos operadores genéticos é utilizada para atualizar a matriz dos Q-valores Q produzida pelo algoritmo *Q-learning* nas gerações anteriores.
- Após atualizar a matriz Q, ela é utilizada pelo algoritmo genético nas gerações subsequentes. Este processo iterativo permite que os pares estado-ação que compõem as melhores soluções obtidas nas últimas gerações sejam premiados recebendo um incremento adicional que os identificarão como boas decisões. O cálculo deste valor incremental tem a seguinte fundamentação teórica:
  De acordo com o modelo de *PCV* descrito na seção 4.2 uma solução para o mesmo pode ser denotada pela Figura 4.9.

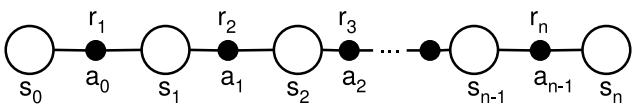


Figura 4.9: Solução para o *PCV*, modelado como um problema de aprendizagem por reforço

Na figura 4.9 a sequência de estados, ações e recompensas correspondem a uma solução S construída durante um episódio do algoritmo *Q-learning*, onde, s_i, $i =$

[2]Um cromossomo corresponde a um indivíduo na população (por exemplo, uma rota para o *PCV*), enquanto que um gene corresponde a cada parte que compõe um cromossomo (no *PCV* uma cidade a compor uma rota)

$0,\ldots,n$ é o estado corrente e r_j, $j = 1,\ldots,n$ é a recompensa imediata recebida por executar a mudança de s_i, para s_{i+1}. Neste contexto, dada a solução S_M o valor da recompensa acumulada R_i, é facilmente calculada da seguinte forma:

$$\begin{aligned} Q(s_0,s_1) &= r_1 + r_2 + r_3 + \cdots + r_{n-1} + r_n = R_0 \\ Q(s_1,s_2) &= r_2 + r_3 + \cdots + r_{n-1} + r_n = R_1 \\ &\vdots \\ Q(s_{n-1},s_n) &= r_n = R_n \end{aligned} \tag{4.8}$$

- Como já mencionado na seção 2.3.4 o algoritmo *Q-learning* usa a equação (2.19) na sua atualização, entretanto, como o valor de R_i para a solução S_M é previamente conhecido, o valor incremental, aqui proposto, pode ser calculado usando:

$$Q(s_i,a_i) = Q(s_i,a_i) + \theta[R_i - Q(s_i,a_i)] \tag{4.9}$$

onde a_i é a ação de partindo do estado s_i escolher ir para o estado s_{i+1}, e θ é um parâmetro que pondera a importância do valor incremental com base no número de visitas de cada estado, denotado por:

$$\theta = NVis(s_i,a_i)/NEp \tag{4.10}$$

onde, *NVis* corresponde ao número de visitas ao par estado-ação (s_i,a_i), e NEp representa o número de episódios executados.

A figura 4.10 apresenta uma visão geral do método *Genético-Learning Cooperativo* implementado.

Os operadores genéticos foram implementados de forma idêntica a versão do algoritmo genético tradicional, utilizando para a seleção uma roleta "viciada" que possibilita que cada indivíduo seja selecionado proporcionalmente ao seu valor na função de aptidão. Para o operador de cruzamento foi utilizado cruzamento de dois pontos e o operador de mutação consistiu de permutar a posição entre duas cidades numa rota. O pseudo-código apresentado no Algoritmo 4.5.1, resume o método descrito.

As alterações propostas no algoritmo *Genético-Learning Cooperativo* que modificam o algoritmo genético tradicional, podem ser visualizadas no pseudo-código nas linhas 2, 6, 7 e 14. Na linha 2 a função *GerarPop* executa o algoritmo *Q-learning* gerando a população inicial do algoritmo. Na linha 6 a taxa de diversificação da população é comparada com um limite L, e caso seja menor que esse limite o operador de cruzamento utilizará um cromossomo obtido com uma execução extra do algoritmo *Q-learning*. Na linha 14 a matriz dos Q-valores Q é atualizada utilizando as informações da melhor solução da população corrente.

4.6 Conclusão

Os métodos híbridos descritos neste Capítulo, são versões modificadas das metaheurísticas GRASP e Algoritmo Genético. As modificações efetuadas consistiram utilização

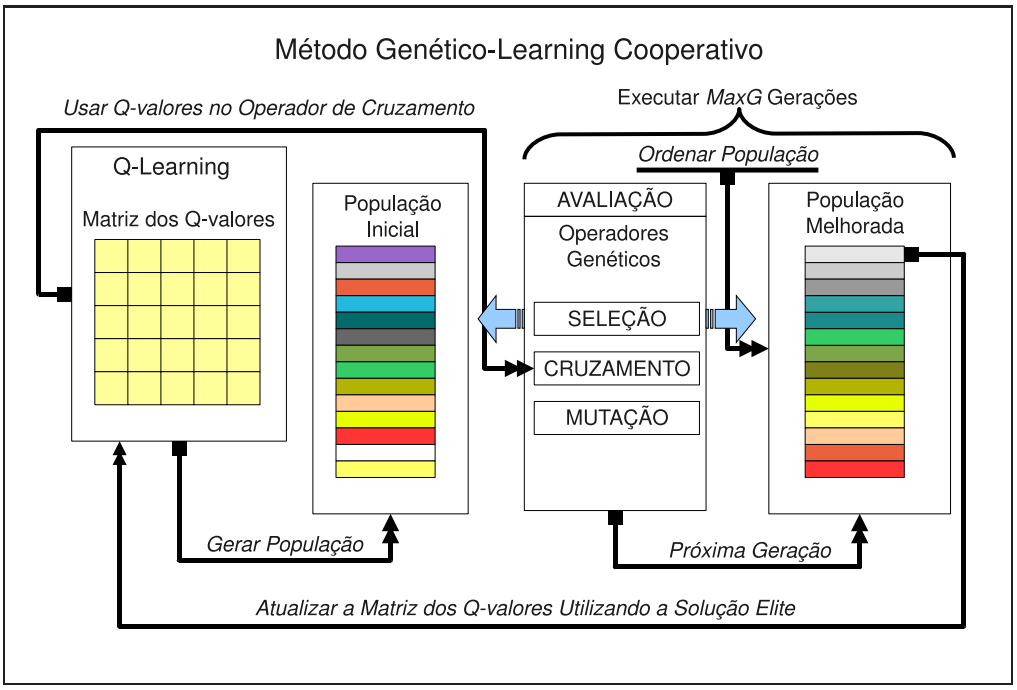


Figura 4.10: Visão geral do método *AG-Learning* Cooperativo implementado

do algoritmo *Q-learning* como estratégia inteligente para a exploração e/ou explotação dos espaço de soluções do problema do caixeiro viajante simétrico.

No caso da metaheurística GRASP, as modificações realizadas consistiram na utilização do algoritmo *Q-learning* como construtor de soluções inicias para mesma.

Segundo Feo & Resende (1995) uma das desvantagens da metaheurística GRASP tradicional é a independência entre suas iterações, isto é, o fato de tal algoritmo não guardar informações do histórico das soluções encontradas nas iterações passadas. Os mesmos autores mencionam ainda que um dos fatores que contribui para o sucesso de um algoritmo de busca local é a boa qualidade da solução inicial. A proposta de uso do algoritmo *Q-learning* como partida exploratória para a metaheurística GRASP, conforme descrita neste Capítulo, vem suprir ambas as deficiências do método tradicional mencionadas por Feo & Resende (1995), pois, o algoritmo *Q-learning* consegue produzir soluções iniciais de boa qualidade e utiliza sua matriz dos q-valores como uma espécie de memória adaptativa que permite que boas decisões tomadas no passado sejam repetidas no futuro.

Na seção 5.1.3 serão apresentados os resultados experimentais que comparam o desempenho das metaheurísticas GRASP e GRASP Reativo descritos no Capítulo 3, com os obtidos com o *GRASP-Learning* proposto neste Capítulo.

No que diz respeito ao Algoritmo Genético, o novo método proposto (método *Genético-Learning-Cooperativo*) utilizou o algoritmo *Q-learning* na geração da população inicial, conseguindo uma população inicial de boa qualidade, tanto nos aspecto do valor da função objetivo, quanto no aspecto relativo ao nível de diversidade da mesma. Uma outra importante inovação proposta neste método foi a atuação cooperativa entre o algoritmo *Q-learning* e os operadores genéticos.

Algoritmo 4.5.1 Algoritmo Genético-Learning-Cooperativo

```
1:  procedure GENÉTICO-LEARNING-COOPERATIVO(Tc, Tm, Tp, MaxG, L)
2:      Q ← QLearning(R, α_q, ε, γ, NEp)
3:      Pop ← GerarCromossomo(Q, Tp)
4:      Pop ← Avaliar(Pop)
5:      for i = 1...MaxG do
6:          Pop ← Selecionar(Pop)
7:          if Div(Pop) < L then
8:              CromoQ ← GerarCromossomo(Q, 1)
9:              Pop ← Cruzamento(Pop, CromoQ, Tc)
10:         else
11:             Pop ← Cruzamento(Pop, Tc)
12:         end if
13:         Pop ← Mutacao(Pop, Tm)
14:         Pop ← OrdenarPop(Pop)
15:         Q ← AtualizarQ(Q, Pop[1])
16:     end for
17:     return A melhor solução em Pop
18: end procedure
```

Com o objetivo de verificar a importância da contribuição da atuação cooperativa proposta no algoritmo Genético híbrido foram efetuados testes com duas versões distintas de tal algoritmo, uma com e outra sem o uso do processo Cooperativo. O resultado destes testes serão apresentados na seção 5.1.4.

Capítulo 5

Resultados Experimentais

5.1 Introdução

Antes de apresentar os resultados computacionais deve-se esclarecer que os experimentos aqui realizados não têm como objetivo encontrar a solução ótima do *PCV*, mas são realizados com o intuito de comparar o desempenho dos métodos tradicionais com os novos métodos propostos. Durante a realização dos experimentos não houve preocupação com a qualidade dos parâmetros de entrada, de forma que os algoritmos foram executados sobre as mesmas condições, ou seja, foi usado o mesmo número de iterações e os algoritmos utilizaram valores idênticos para os parâmetros comuns ajustáveis.

Todos os algoritmos foram implementados utilizando o software *Matlab* e os testes foram executados utilizando 10 instâncias da biblioteca *TSPLIB* [TSPLIB 2008], em computador com processador Pentium IV 2.8 *Ghz*, com 2 Gb de memória RAM, em sistema operacional Linux e o tempo de processamento foi medido em segundos. Como se trata de algoritmos não determinísticos os resultados apresentados são a média de 30 execuções para cada instância.

Os experimentos realizados foram divididos em duas etapas:

- Na primeira etapa é feita a comparação dos resultados obtidos utilizando apenas os algoritmos da fase construtiva das metaheurística GRASP, GRASP Reativo e *GRASP-Learning*. Nesta etapa, no que diz respeito aos algoritmos genéticos é feita a comparação da qualidade da população inicial considerado o valor da função objetivo e também o nível de diversidade da população dos *AG* tradicional e da versão híbrida.
- A segunda parte do experimento compara o desempenho final das metaheurísticas GRASP, GRASP Reativo e *GRASP-Learning*, e também dos Algoritmos Genéticos Tradicional e das versões utilizando aprendizagem por reforço.

5.1.1 Comparação da Fase Construtiva das Metaheurísticas GRASP

Como já mencionado na Seção 4.4 a diferença básica entre as metaheurísticas GRASP, GRASP Reativo e *GRASP-Learning* é a fase construtiva. Os algoritmos GRASP tradicional e GRASP Reativo implementado neste trabalho tem na fase construtiva um algoritmo

guloso-aleatório que utiliza a heurística do vizinho mais próximo, enquanto que a versão com aprendizagem utiliza na fase construtiva o algoritmo *Q-learning*.

Uma das condições para que a metaheurística GRASP tenha um bom desempenho é a necessidade de utilizar busca local partindo de boas soluções iniciais. O objetivo desta seção é apresentar uma análise comparativa que permita indicar qual dos algoritmos utilizados na fase construtiva tem melhor desempenho. A Tabela 5.1 apresenta o resultados computacionais obtidos (média de 30 execuções) executando apenas a fase construtiva de cada um das metaheurísticas GRASP citadas anteriormente, enquanto que a Figura 5.1 apresenta os mesmos resultados de forma gráfica.

Tabela 5.1: Resultado da Fase Construtiva do GRASP, GRASP Reativo e *GRASP-Learning*

Instância	GRASP Tradicional		GRASP Reativo		*GRASP-Learning*	
TSPLIB	Valor	Tempo	Valor	Tempo	Valor	Tempo
gr17	4679,10	0,03	3517,07	0,04	2469,00	0,70
bays29	5957,90	0,10	4561,37	0,16	2938,35	1,19
swiss42	4774,85	0,21	3513,90	0,45	1769,50	1,73
gr48	21197,95	0,27	14440,25	0,65	7359,25	1,98
berlin52	29782,40	0,32	22607,95	0,82	11445,70	2,16
pr76	573749,50	0,70	384152,00	2,44	174823,00	3,20
gr120	51733,50	1,77	32081,20	9,16	11471,70	8,73
ch150	54572,25	2,82	36813,85	17,63	11705,20	11,16
si175	48396,40	3,90	41672,10	27,97	30729,35	13,19
a280	34454,45	10,41	22130,55	113,01	5203,85	31,49

Pela análise dos valores listados na tabela 5.1 pode-se perceber que a fase construtiva do método *GRASP-Learning* foi a que obteve melhor desempenho, tendo inclusive, menor tempo de processamento que a versão GRASP Reativo nas instâncias maiores. O fato da qualidade e da taxa de diversidade da população se tornarem competitivas a medida que o tamanho das instâncias crescem são decorrentes da forma como o algoritmo *Q-learning* atua, ou seja, para as instâncias maiores existe um número maior de estados a serem escolhidos (mais ações disponíveis),e em consequência disso, um número maior de soluções distintas podem ser geradas (maior taxa de diversidade). Em relação a qualidade da população os melhores resultados para as instâncias maiores são justificados dado ao fato de que nas instâncias maiores o algoritmo *Q-learning* executa um número maior de episódios antes de começar a construção das soluções.

5.1.2 Comparação da População Inicial dos Algoritmos Genéticos

Em um algoritmo genético uma boa população inicial é aquela que contenha diversidade suficiente para permitir ao algoritmo combinar características e produzir novas e

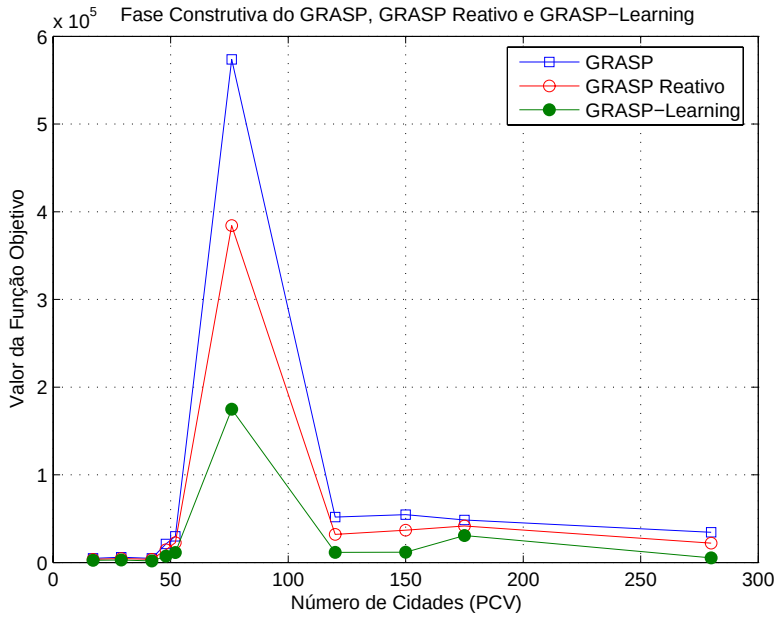


Figura 5.1: Comparação das Fases Construtivas do GRASP, GRASP Reativo e *GRASP-Learning* (Valor da Função Objetivo)

melhores soluções, ou seja, permitir que ao logo das gerações possam surgir indivíduos mais aptos. A qualidade da população inicial em um algoritmo genético tem influência direta no nível de diversificação e na velocidade de convergência. Portanto, produzir uma população inicial de boa qualidade não é uma tarefa trivial, pois soluções geradas aleatoriamente diversificam o espaço de soluções, mas geralmente apresentam baixa qualidade, exigindo um tempo de processamento muito elevado para serem melhoradas. Por outro lado, soluções gulosas, apesar de terem boa qualidade, tendem a possuírem baixa diversidade, o que poder levar o processo evolutivo a ficar preso em ótimos locais.

O algoritmo genético proposto na Seção 4.5 utiliza uma heurística alternativa para gerar sua população inicial, tal heurística utiliza a matriz dos Q-valores produzida pelo algoritmo *Q-learning*, como fonte de informação para ponderar entre gula e aleatoriedade durante a geração dos indivíduos da população.

A Tabela 5.2 apresenta alguns resultados que permitem realizar uma análise comparativa da qualidade da população inicial dos algoritmos genéticos implementados. Os valores listados para cada instância corresponde a média dos melhores indivíduos obtidos em cada população, sendo que foram geradas 30 populações para cada algoritmo. Na Figura 5.2 os resultados do experimento são apresentados de forma gráfica.

Os dados apresentados anteriormente na Tabela 5.2 são relativos ao valor da função objetivo dos algoritmos genéticos, calculada a média dos melhores indivíduos de 30 populações. No que diz respeito à análise de diversidade da população, esta pode ser feita

Tabela 5.2: Comparação das Populações Iniciais dos Algoritmos Genéticos (Valor da Função Objetivo)

Nome da Instância	Genético Tradicional	*Genético-Learning*
gr17	3600,30	2408,70
bays29	4870,40	3292,10
swiss42	4070,60	2380,70
gr48	17797,00	8438,30
berlin52	25615,00	11653,00
pr76	509770,00	222350,00
gr120	46316,00	16893,00
ch150	49319,00	18786,00
si175	46194,00	35513,00
a280	31869,00	10369,00

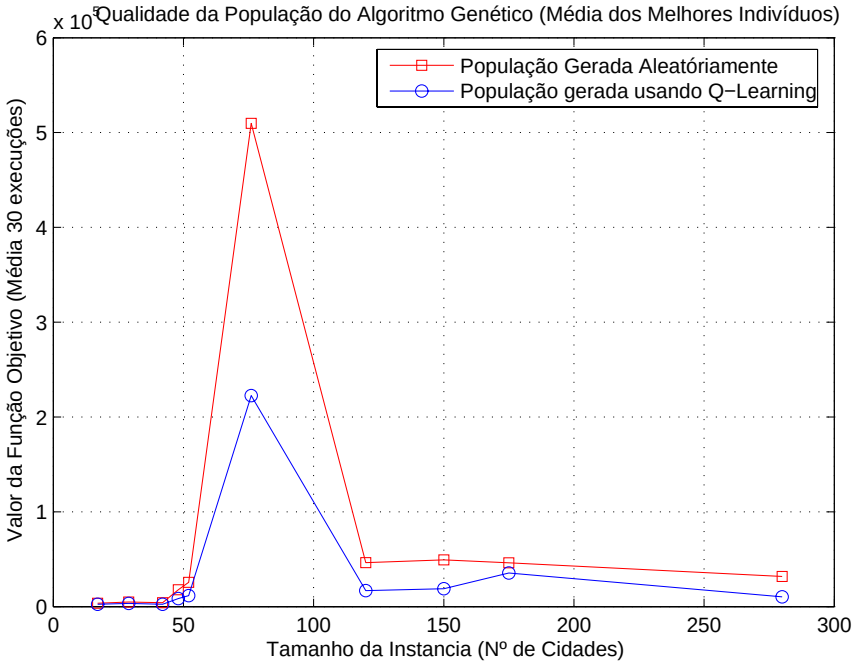


Figura 5.2: Comparação das Populações Iniciais dos Algoritmos Genéticos (Média dos Melhores Indivíduos)

comparando o valor da taxa de diversidade obtida através da Equação 5.1

$$T_{Div} = (1 - \frac{N_{Rep}}{N_{Total}})100 \tag{5.1}$$

onde N_{Rep} corresponde ao número de elementos repetidos na população e N_{Total} é o número total de elementos, isto é, o tamanho da população. A quantidade de elementos repetidos N_{Rep} foi determinado por um procedimento que verifica em cada população de indivíduos (cromossomos) idênticos.

A Tabela 5.3 apresenta os resultados relativos a taxa de diversificação para cada uma das instâncias, com a população gerada de forma aleatória (algoritmo Genético tradicional) e utilizando o algoritmo *Q-learning* (*Genético-Learning Cooperativo*). A Figura 5.3 mostra a comparação entre as duas populações de forma gráfica.

Tabela 5.3: Comparação das Populações Iniciais dos Algoritmos Genéticos (Taxa de Diversidade)

Nome da Instância	Genético Tradicional (%)	*Genético-Learning Cooperativo* (%)
gr17	98,93	92,00
bays29	98,96	95,26
swiss42	98,83	96,60
gr48	99,00	96,56
berlin52	98,90	97,80
pr76	99,00	98,00
gr120	98,93	98,90
ch150	99,00	99,00
si175	98,96	98,73
a280	98,96	98,86

A comparação da qualidade das populações dos algoritmos genéticos demonstra que a população gerada utilizando o algoritmo *Q-learning* tem melhor qualidade no aspecto relativo ao valor da função objetivo, e taxa de diversidade competitiva com a taxa de diversidade da população gerada aleatoriamente.

5.1.3 Comparação de desempenho das metaheurísticas GRASP Implementadas

Nesta seção será apresentado um comparativo entre os resultados obtidos com a implementação computacional das metaheurísticas GRASP, GRASP Reativo e o novo método proposto *GRASP-Learning*. Todos os algoritmos foram executados sob as mesma condições paramétricas, sendo os valores dos parâmetros ajustáveis listados na tabela 5.4.

Ao se observar a tabela 5.4 é importante entender que os parâmetros α do GRASP Reativo e ε do *GRASP-Learning*, ambos auto-ajustáveis, são valores que variam no intervalo [0,1]. Uma outra observação pertinente é que os parâmetros α (versão tradicional e reativo) não tem equivalência com o parâmetro α_q do *GRASP-Learning*, ou seja, o parâmetro α é utilizado para controlar os índices de "gula" e aleatoriedade no GRASP

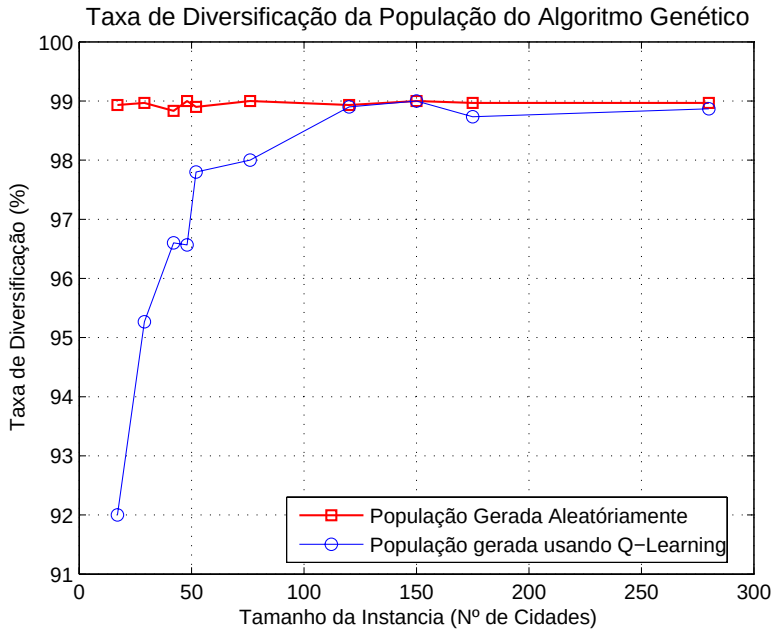


Figura 5.3: Comparação das Populações Iniciais dos Algoritmos Genéticos (Taxa de Diversidade)

Tabela 5.4: Parâmetros Ajustáveis para as Metaheurísticas GRASP Implementadas

Instância	GRASP Tradicional		GRASP Reativo		*GRASP-Learning*				
TSPLIB	Nº Iter.	α	Nº Iter.	α	Nº Iter.	Nº Epis.	α_q	γ	ε
gr17	300	0,8	300	Adaptativo	300	10	0,9	1	Adaptativo
bays29	300	0,8	300	Adaptativo	300	10	0,9	1	Adaptativo
swiss42	300	0,8	300	Adaptativo	300	20	0,9	1	Adaptativo
gr48	300	0,8	300	Adaptativo	300	20	0,9	1	Adaptativo
berlin52	300	0,8	300	Adaptativo	300	50	0,9	1	Adaptativo
pr76	300	0,8	300	Adaptativo	300	100	0,9	1	Adaptativo
gr120	300	0,8	300	Adaptativo	300	100	0,9	1	Adaptativo
ch150	300	0,8	300	Adaptativo	300	150	0,9	1	Adaptativo
si175	300	0,8	300	Adaptativo	300	200	0,9	1	Adaptativo
a280	300	0,8	300	Adaptativo	300	200	0,9	1	Adaptativo

tradicional e no GRASP Reativo, enquanto que o parâmetro α_q representa o coeficiente de aprendizagem do algoritmo *Q-learning* utilizado na versão GRASP híbrida.

Os valores listados na tabela 5.5 correspondem a média de 30 execuções obtidos com 10 instâncias do caixeiro viajante simétrico (valor da função objetivo e tempo de processamento), cujos valores na íntegra estão disponíveis nas tabelas da seção A do Apêndice 6.0.3.

As figuras 5.4 e 5.5 apresentam uma comparação entre as três versões das metaheurísticas implementadas, considerando o valor da função objetivo e o tempo de processamento, respectivamente. Com o objetivo de melhorar a visualização e facilitar o entendi-

Tabela 5.5: Resultados das Metaheurísticas GRASP, GRASP Reativo e *GRASP-Learning*

Instância	TSPLIB	GRASP Tradicional		GRASP Reativo		*GRASP-Learning*	
TSPLIB	Valor	Valor	Tempo	Valor	Tempo	Valor	Tempo
gr17	2085,00	2085,00	25,30	2085,00	21,07	2085,00	17,03
bays29	2020,00	2085,63	112,41	2060,50	103,96	2030,30	46,34
swiss42	1273,00	1441,43	373,62	1385,07	354,05	1281,40	84,64
gr48	5046,00	5801,77	559,81	5454,17	532,62	5442,77	127,41
berlin52	7542,00	8780,73	752,37	8420,93	599,10	8053,60	156,36
pr76	108159,00	140496,00	1331,51	131380,33	1724,84	129707,33	719,65
gr120	6942,00	10553,61	5000,10	8773,33	5945,75	8540,47	2443,82
ch150	6528,00	10785,59	11167,91	9304,40	12348,96	7012,93	1753,29
si175	21407,00	25733,07	21509,92	23646,14	12803,00	22700,35	8830,08
a280	2579,00	5799,77	40484,48	4244,19	37075,25	2991,30	8442,40

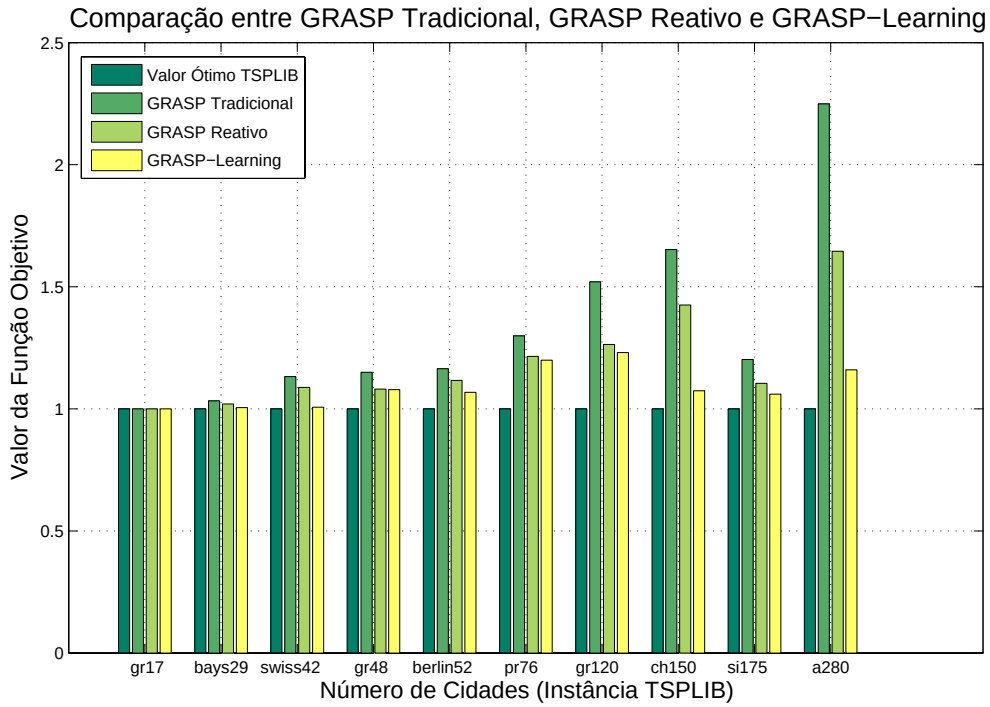


Figura 5.4: Resultados GRASP Tradicional, GRASP Reativo e *GRASP-Learning* (Valor da Função Objetivo)

mento os dados representados nos gráficos foram submetidos a uma normalização, sendo os valores da função objetivo normalizados pelo valor ótimo conhecido de cada instância da *TSPLIB*, e o tempo de processamento normalizado pela média do tempo de execução

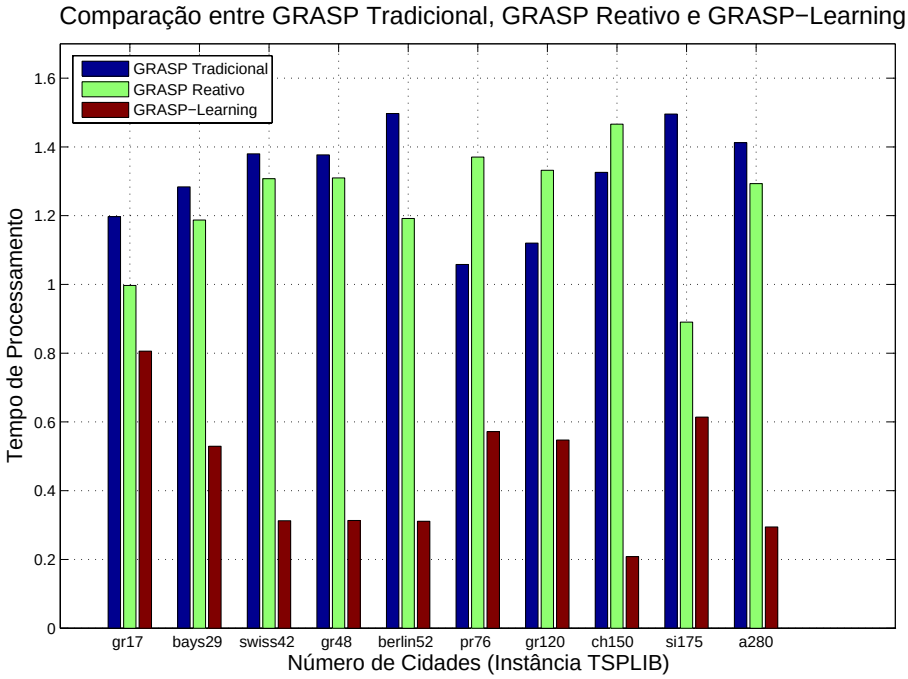


Figura 5.5: Resultados GRASP Tradicional, GRASP Reativo e *GRASP-Learning*(Tempo de Processamento)

de cada metaheurística, para cada uma das instância, ou seja:

$$\begin{aligned} M_i &= (\sum_{j=1}^{m} T_{ij})/m \quad \forall \quad i = 1,2,...,n \\ TN_{ij} &= T_{ij}/M_i \end{aligned} \tag{5.2}$$

onde, n é a quantidade de instâncias utilizadas no experimento, m é a quantidade de algoritmos testados, T_{ij} é o tempo de processamento obtido para a instância i, executando o algoritmo j, e TN_{ij} é o valor do tempo normalizado para a instância i executando o algoritmo j.

A análise dos resultados apresentados na tabela 5.5 demonstra que a metaheurística *GRASP-Learning* obteve, em média, melhores resultados que a metaheurística GRASP tradicional, e superou até mesmo o desempenho de sua versão melhorada, a metaheurística GRASP Reativo. Os resultados da função objetivo obtidos com o método híbrido foram melhores do que os resultados relativos à versão tradicional, tendo no pior caso ocorrido empate, pois, todas as versões encontraram o ótimo da instância $gr17$. No melhor caso o novo método apresentou redução no custo de $48,42\%$ (instância $a280$).

Quando comparado ao GRASP Reativo, o método híbrido obteve resultados bastante competitivos, obtendo desempenho superior na maioria das instâncias para os valores da função objetivo, tendo reduzido $29,52\%$ o custo da instância $a280$.

Ao se comparar os resultados relativos ao tempo de processamento, a vantagem da me-

taheurística *GRASP-Learning* é ainda mais expressiva, obtendo quando comparado com o GRASP tradicional $32,69\%$ de redução do tempo de processamento para a instância *gr*17 (pior caso) e $84,30\%$ para a instância *ch*150 (melhor caso). Na comparação com o GRASP reativo o método híbrido conseguiu, no pior caso, $19,17\%$ para a instância *gr*17 e $85,80\%$ para o melhor caso (instância *ch*150).

Os melhores resultados relativos ao tempo de processamento conseguidos pela metaheurística *GRASP-Learning* são justificados devido a boa qualidade das soluções iniciais geradas pelo algoritmo *Q-learning* na fase construtiva, pois, partindo de soluções de boa qualidade a metaheurística conseguiu acelerar a busca local. Além da boa qualidade das soluções iniciais, a metaheurística *GRASP-Learning* tem a vantagem de utilizar memória de uma iteração para outra, e isso permite que a cada iteração a qualidade da solução inicial seja melhorada, com base nas informações contidas na matriz dos Q-valores, que é atualizada a cada iteração pelo algoritmo *Q-learning*. É importante observar que os bons resultados são mais expressivos a medida que as instâncias crescem, e isso é justificado pois, quanto maior a instância do *PCV* maior o grau de dificuldade de resolvê-la, e portanto, mais perceptível é a vantagem do algoritmo *Q-learning* na construção de boas soluções iniciais utilizando o mecanismo de memória presente na matriz dos q-valores.

5.1.4 Comparação de desempenho dos Algoritmos Genéticos Implementados

Os resultados experimentais para todas as versões dos algoritmo genéticos implementadas foram obtidas utilizando os mesmos parâmetros de entrada, os quais têm os valores listados na tabela 5.6.

Tabela 5.6: Parâmetros Ajustáveis para os Algoritmos Genéticos Implementados

Instância	Todos os AGs				AGs com Aprendizagem			
TSPLIB	Nº Gerações	Tc	Tm	Tp	Nº Episódios	α_q	γ	ε
gr17	1000	0,7	0,2	100	500	0,8	1	Adaptativo
bays29	1000	0,7	0,2	100	500	0,8	1	Adaptativo
swiss42	1000	0,7	0,2	100	500	0,8	1	Adaptativo
gr48	1000	0,7	0,2	100	500	0,8	1	Adaptativo
berlin52	1000	0,7	0,2	100	500	0,8	1	Adaptativo
pr76	1500	0,7	0,2	100	1000	0,8	1	Adaptativo
gr120	2000	0,7	0,2	100	1000	0,8	1	Adaptativo
ch150	2000	0,7	0,2	100	2000	0,8	1	Adaptativo
si175	2000	0,7	0,2	100	2000	0,8	1	Adaptativo
a280	2000	0,7	0,2	100	2000	0,8	1	Adaptativo

A Tabela 5.7 apresenta a média obtida com 30 execuções para cada versão dos algoritmos genéticos implementados. Os resultados obtidos nos experimentos são disponibilizados na íntegra nas tabelas da seção A do Apêndice 6.0.3.

Os gráficos 5.6 e 5.7 apresentam uma comparação entre as três versões dos algoritmos testados, (Genético tradicional, *Genético-Learning*, *Genético-Learning Coopera-*

Tabela 5.7: Resultados do Genético, Genético-Learning e Genético-Learning Cooperativo

Instâncias	TSPLIB	Genético Tradicional		Genético-Learning		AG-Learning Cooperativo	
TSPLIB	Valor	Valor	Tempo	Valor	Tempo	Valor	Tempo
gr17	2085,00	2104,97	48,73	2087,37	48,03	2085,00	51,77
bays29	2020,00	2286,23	52,27	2252,13	52,24	2166,47	57,45
swiss42	1273,00	1614,20	54,52	1546,53	55,01	1457,73	63,66
gr48	5046,00	6839,77	55,99	5967,90	56,19	5744,90	66,59
berlin52	7542,00	10095,63	55,55	8821,93	55,68	8679,43	69,17
pr76	108159,00	189659,00	89,49	165281,67	95,36	132100,33	123,60
gr120	6942,00	16480,73	132,05	12205,87	140,41	8787,00	211,51
ch150	6528,00	19705,47	142,44	9612,33	153,26	8803,37	247,47
si175	21407,00	30860,97	151,86	29264,13	193,25	24818,03	285,86
a280	2579,00	13642,07	205,92	3852,67	274,55	3768,03	543,17

tivo) considerando o valor da função objetivo e o tempo de processamento, respectivamente. De forma análoga ao procedimento executado com as metaheurísticas GRASP, os dados representados nos gráficos foram submetidos a uma normalização, sendo como antes, os valores da função objetivo normalizados pelo valor ótimo de cada instância da *TSPLIB*, e o tempo de processamento normalizado pelo tempo médio de execução de cada algoritmo, para cada instância, conforme descrito na equação 5.2.

Ao se analisar os resultados experimentais obtidos pode-se perceber que os algoritmos genéticos com aprendizagem obtiveram melhores resultados para o valor da função objetivo, o mesmo não acontece com o tempo de processamento.

Em relação ao valor da função objetivo, o resultado obtido é conseqüência da boa qualidade da população inicial gerada pelo algoritmo *Q-learning* e também pelo processo de iteração cooperativa dos operadores genéticos com a matriz dos Q-valores. O bom desempenho relativo ao valor da função objetivo é notório principalmente a medida que as instâncias crescem. Ao se considerar por exemplo, a comparação entre o Algoritmo Genético tradicional e o *Genético-Learning Cooperativo* tem-se no pior caso $0,95\%$ (instância $gr17$) e no melhor caso $72,38\%$ (instância $a280$) de melhoria.

No que diz respeito ao tempo de processamento o algoritmo *Genético-Learning Cooperativo* leva notória desvantagem em relação as outras duas versões, isso é justificável, uma vez que tal versão utiliza o algoritmo *Q-learning* na construção da população inicial e também no processo de cooperação com os operadores genéticos, e portanto, o cálculo final do tempo de execução tem acrescentado o tempo gasto no processamento dos NEp episódios do algoritmo *Q-learning*. Apesar da desvantagem do algoritmo *Genético-Learning Cooperativo* no que diz respeito ao tempo de processamento, a análise de sobrevivência realizada com os algoritmos genéticos (ver seção 5.2.2) demonstrou que esta versão conseguiu melhores resultados (maior valor de fitness) em menor número de gerações.

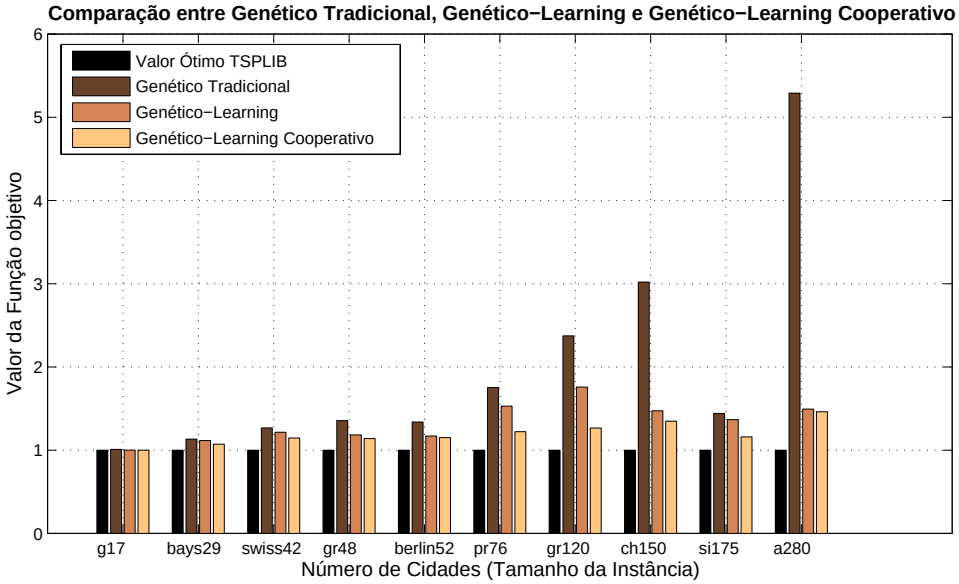


Figura 5.6: Resultados Genético Tradicional, Genético-Learning e *Genético-Learning Cooperativo* (Função Objetivo)

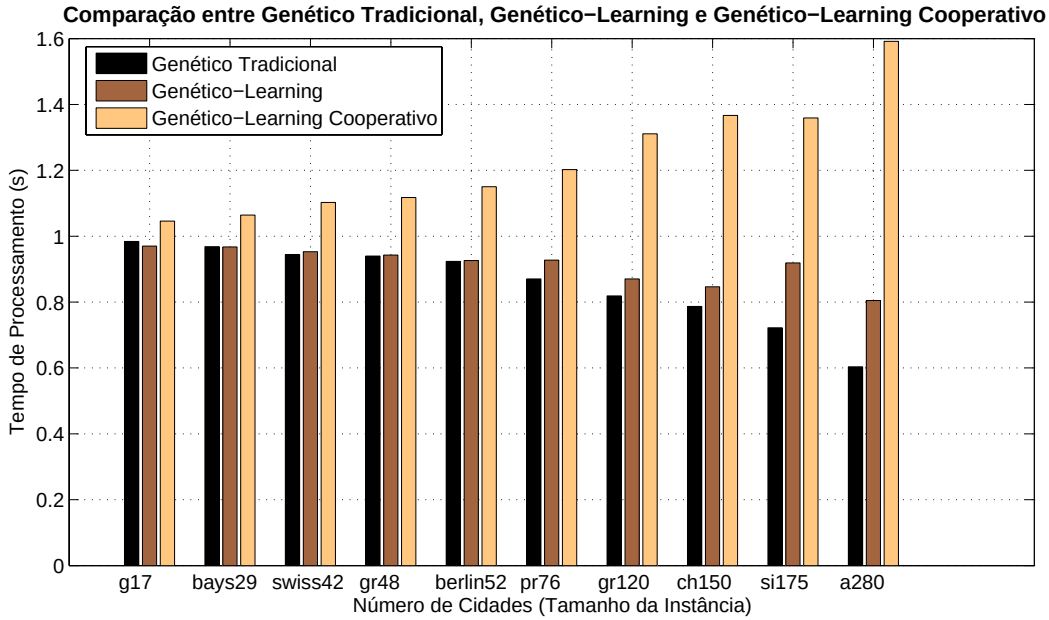


Figura 5.7: Resultados Genético Tradicional, Genético-Learning e *Genético-Learning Cooperativo* (Tempo de Processamento)

5.2 Análise Estatística

Nesta seção será apresentada uma análise estatística dos resultados computacionais obtidos com a implementação dos métodos proposto no Capítulo 4. A análise foi feita

por meio de teste de hipótese e análise de sobrevivência que serão detalhados nas seções a seguir.

5.2.1 Teste de Hipótese

O objetivo de um teste de hipótese é decidir, com base na informação fornecida pelos dados de uma amostra, sobre a aceitação ou não de uma dada hipótese. Neste trabalho este teste estatístico é utilizado para confirmar a hipótese que os métodos híbridos propostos têm melhor desempenho computacional do que as metaheurísticas GRASP e Algoritmo Genético tradicionais.

O teste de hipótese foi então elaborado utilizando a média independente das variáveis aleatórias X_1, X_2, X_3 e X_4 as quais correspondem os resultados experimentais obtidos através da implementação da metaheurísticas GRASP Reativo, *GRASP-Learning*, Algoritmo Genético e Algoritmo *Genético-Learning Cooperativo*, respectivamente.

Os dados amostrais foram obtidos utilizando a Instância *bays*29 do Problema do Caixeiro Viajante [TSPLIB 2008]. A escolha desta instância foi motivada pelo fato de que os valores dos resultados experimentais obtidos com a mesma para cada um dos algoritmos, são bem próximos, o que permite portanto, que o resultado do teste de hipótese seja generalizado para as demais instâncias.

Para o teste foram utilizados os resultados de 30 execuções da instância *bays*29 com todos os algoritmos previamente mencionados. Os valores das variáveis aleatórias X_1, X_2, estão disponíveis nas tabelas A.3 e A.5, enquanto que os valores das variáveis X_3 e X_4 são listados nas tabelas A.7 e A.11, ambas no apêndice 6.0.3.

Teste para o GRASP Reativo e *GRASP-Learning*

Para elaboração dos teste de hipótese foram considerados os dois algoritmos de melhor desempenho, segundo os resultados apresentados na seção 5.1.3, ou seja, as metaheurísticas GRASP Reativo e *GRASP-Learning*.

1. **Parâmetros de Interesse para o Teste**
   Os parâmetros de interesse para o teste são μ_1 e μ_2 que representam custo médio do percurso para o *PCV* da instância *bays*29, obtidos utilizando as metaheurísticas GRASP Reativo e *GRASP-Learning*, respectivamente. Uma vez que, o melhor algoritmo para o *PCV* será aquele que obtiver menor custo médio para tal percurso, deseja-se testar se:
   $$\mu_1 = \mu_2 \Longleftrightarrow \mu_1 - \mu_2 = 0 \tag{5.3}$$
   Partido da hipótese implícita em 5.3, deseja-se responder a seguinte pergunta: O método GRASP Learning tem melhor desempenho que o método GRASP Reativo? É importante notar que a pergunta refere-se a qual método apresenta melhor média. Que para o caso do *PCV* refere-se à menor média.
2. **Definição das Hipóteses**
   - Hipótese Nula:

   $$H_0 : \mu_1 = \mu_2 \tag{5.4}$$

- Hipótese Alternativa:

$$H_1 : \mu_1 > \mu_2 \tag{5.5}$$

3. **Escolha do Nível de Significância** No teste foi utilizado nível de significância $\alpha = 0,01$, ou seja, 1% de probabilidade de ocorrência do erro do tipo I[1].
4. **Estatística Apropriada para o Teste** A estatística de teste escolhida utilizará a comparação das médias amostrais, com variâncias desconhecidas e não necessariamente iguais, e será utilizada a distribuição *t* de *Student*, com valor de t_0^* aproximado para a estatística calculado por:

$$t_0^* = \frac{\bar{X_1} - \bar{X_2}}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}} \tag{5.6}$$

   Onde, $\bar{X_1}$ e $\bar{X_2}$ é a média e S_1^2 e S_2^2 são as variâncias relativas as variáveis aleatórias X_1, X_2, referentes aos resultados obtidos para as metaheurísticas GRASP Reativo e GRASP-Learning, respectivamente. As variáveis n_1, n_2 representam a quantidade de execuções para cada um dos algoritmos, que neste caso tem-se $n_1 = n_2 = 30$
5. **Grau de Liberdade** O valor tabelado t_{tab} foi consultado na tabela da distribuição de *Student* adotando-se o grau de liberdade ν dado pela equação:

$$\nu = \frac{(\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2})^2}{\frac{(S_1^2/n_1)^2}{n_1+1} + \frac{(S_2^2/n_2)^2}{n_2+1}} - 2 \tag{5.7}$$

6. **Cálculo da Estatística do Teste** Obtendo-se as médias das variáveis X_1 e X_2 e suas respectivas variâncias:

   - $\bar{X_1} = 2060,50$;
   - $\bar{X_2} = 2030,30$;
   - $S_1^2 = 560,40$;
   - $S_2^2 = 43,75$;

   Substituindo os valores das variáveis na equação 5.6 tem-se:

$$t_0^* = \frac{2060,50 - 2030,30}{\sqrt{\frac{560,40}{30} + \frac{43,75}{30}}}$$
$$t_0^* = 6,71$$

[1] A probabilidade de ocorrência do erro do tipo *I* é a probabilidade de se rejeitar a hipótese H_0, sendo ela verdadeira.

Da mesma forma substituindo os valores das variáveis na equação 5.7 tem-se:

$$\nu = \frac{(\frac{560,40}{30} + \frac{43,75}{30})^2}{\frac{(560,40/30)^2}{30+1} + \frac{(43,75/30)^2}{30+1}} - 2$$
$$\nu = 33,81$$
$$\nu \cong 34$$

Dados o nível de significância $\alpha = 0,01$ e o grau de liberdade $\nu = 34$ pode-se consultar o valor tabelado t_{tab} na distribuição de *Student*, encontrando o valor de $t_{\alpha,\nu}$, ou seja:

$$t_{0,01;34} = 2,728 \tag{5.8}$$

7. **Decisão Sobre a Hipótese** H_0 Para se tomar uma decisão sobre a hipótese H_0 é necessário comparar o valor calculado t_0^*, com o valor tabelado $t_{\alpha,\nu}$, de maneira que:

$$\begin{cases} \text{Se } t_0^* > t_{\alpha,\nu}: & H_0 \text{ será rejeitada} \\ \text{Caso contrário:} & H_0 \text{ será aceita} \end{cases} \tag{5.9}$$

Comparando os valores do t_0^* calculado com o t tabelado tem-se:

$$t_0^* > t_{0,01,34}$$
$$6,71 > 2,728$$

Logo a hipótese H_0 deve ser rejeitada.

8. **Conclusão do Teste** Como a hipótese nula $H_0 : \mu_1 = \mu_2$ foi rejeitada a hipótese alternativa $H_1 : \mu_1 > \mu_2$ é aceita, e isso significa que pode-se afirmar com 99% de certeza que os resultados conseguidos com a nova metaheurística *GRASP-Learning* são em média melhores que os resultados obtidos com a GRASP Reativo.

A decisão do teste de hipótese pode ser interpretada com maior clareza através da análise do gráfico apresentado na figura 5.8. É importante observar que o valor de $t_0^* = 6,71$ (denotado pelo ponto vermelho no gráfico) encontra-se na região de rejeição de H_0, o que justifica a decisão tomada sobre tal hipótese.

Teste para o Algoritmo Genético e Algoritmo *Genético-Learning Cooperativo*

De forma análoga ao teste efetuado com as metaheurísticas GRASP-Reativo e *GRASP-Learning*, considerou-se os dois algoritmos genéticos de melhor desempenho, segundo os resultados apresentados na seção 5.1.4.

1. **Parâmetros de Interesse para o Teste** Os parâmetros de interesse para o teste são μ_3 e μ_4 o custo médio do percurso para o *PCV* da instância *bays*29, utilizando os algoritmos Genético e *Genético-Learning Cooperativo*, respectivamente. Deseja-se testar se:

$$\mu_3 = \mu_4 \Longleftrightarrow \mu_3 - \mu_4 = 0 \tag{5.10}$$

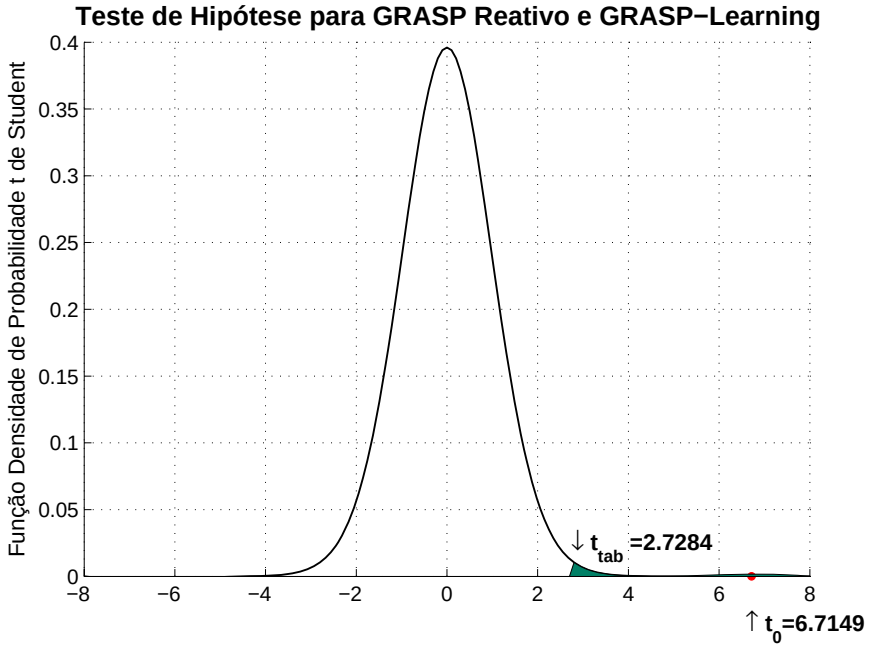


Figura 5.8: Resultado do Teste de Hipótese para as Metaheurísticas GRASP

2. **Definição das Hipóteses**

   - Hipótese Nula:

   $$H_0 : \mu_3 = \mu_4 \tag{5.11}$$

   - Hipótese Alternativa:

   $$H_1 : \mu_3 > \mu_4 \tag{5.12}$$

3. **Escolha do Nível de Significância** A exemplo do teste efetuado com as metaheurísticas GRASP foi utilizado nível de significância $\alpha = 0,01$, ou seja, 1% de probabilidade de ocorrência do erro do tipo I.
4. **Estatística Apropriada para o Teste**

   $$t_0^* = \frac{\bar{X_3} - \bar{X_4}}{\sqrt{\frac{S_3^2}{n_3} + \frac{S_4^2}{n_4}}} \tag{5.13}$$

   Onde, $\bar{X_3}$ e $\bar{X_4}$ são as médias e S_3^2, S_4^2 são as variâncias relativas as variáveis aleatórias X_3, X_4, referentes aos resultados obtidos para algoritmos Genético tradicional, e *Genético-Learning Cooperativo*, respectivamente. De forma análoga ao teste apresentado na seção 5.2.1, as variáveis n_3, n_4 representam a quantidade de execuções para cada um dos algoritmos Genéticos, que neste caso tem-se $n_3 = n_4 = 30$.
5. **Grau de Liberdade** O valor tabelado t_{tab} foi consultado na tabela da distribuição

de *Student* adotando-se o grau de liberdade ν dado pela equação:

$$\nu = \frac{(\frac{S_3^2}{n_3} + \frac{S_4^2}{n_4})^2}{\frac{(S_3^2/n_3)^2}{n_3+1} + \frac{(S_4^2/n_4)^2}{n_4+1}} - 2 \tag{5.14}$$

6. **Cálculo da Estatística do Teste** Obtendo-se as médias das variáveis X_3 e X_4 e suas respectivas variâncias:

   - $\bar{X_3} = 2286,20$;
   - $\bar{X_4} = 2166,50$;
   - $S_3^2 = 15630,74$;
   - $S_4^2 = 2802,53$;

   Substituindo os valores das variáveis na equação 5.13 tem-se:

$$t_0^* = \frac{2286,20 - 2166,50}{\sqrt{\frac{15630,74}{30} + \frac{2802,53}{30}}}$$
$$t_0^* = 4,83$$

   Da mesma forma substituindo os valores das variáveis na equação 5.14 tem-se:

$$\nu = \frac{(\frac{15630,74}{30} + \frac{2802,53}{30})^2}{\frac{(15630,74/30)^2}{30+1} + \frac{(2802,53/30)^2}{30+1}} - 2$$
$$\nu = 39,77$$
$$\nu \cong 40$$

   Dados o nível de significância $\alpha = 0,01$ e o grau de liberdade $\nu = 40$ pode-se consultar o valor tabelado t_{tab} na distribuição de *Student*, encontrando o valor de $t_{\alpha,\nu}$, ou seja:

$$t_{0,01;40} = 2,7045 \tag{5.15}$$

7. **Decisão Sobre a Hipótese** H_0 Para se tomar uma decisão sobre a hipótese H_0 é necessário comparar o valor calculado t_0^*, com o valor tabelado $t_{\alpha,\nu}$, de maneira que:

$$\begin{cases} \text{Se } t_0^* > t_{\alpha,\nu}: & H_0 \text{ será rejeitada} \\ \text{Caso contrário:} & H_0 \text{ será aceita} \end{cases} \tag{5.16}$$

   Comparando os valores do t_0^* calculado com o t tabelado tem-se:

$$t_0^* > t_{0,01,40}$$
$$4,83 > 2,7045$$

   Logo a hipótese H_0 deve ser rejeitada.
   Como na seção anterior, a decisão do teste de hipótese pode ser interpretada com maior clareza através da análise do gráfico apresentado na figura 5.9. Novamente,

é importante atentar que o valor de $t_0^* = 4,83$ (denotado pelo ponto vermelho no gráfico) encontra-se na região de rejeição de H_0, o que justifica a decisão tomada sobre a mesma.

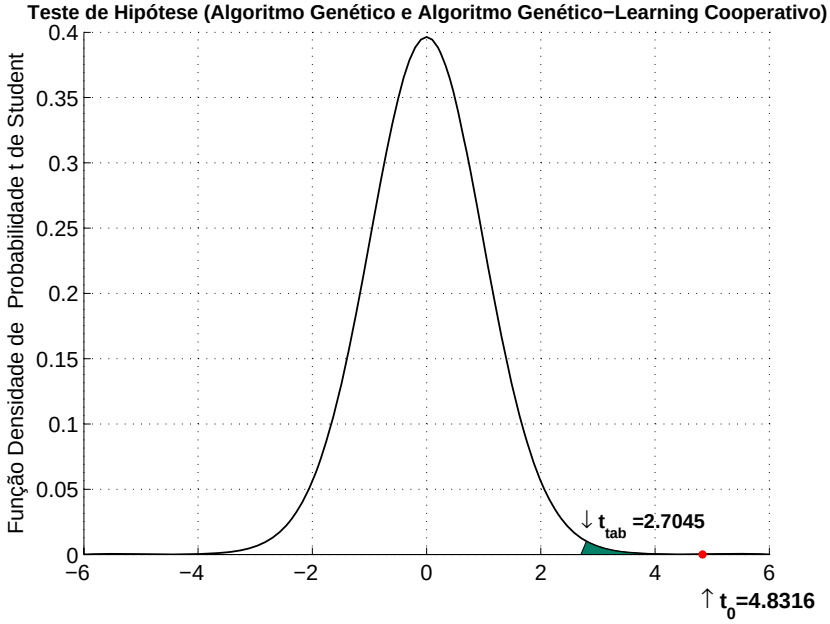


Figura 5.9: Resultado do Teste de Hipótese para os Algoritmos Genéticos

8. **Conclusão do Teste** Como a hipótese nula $H_0 : \mu_3 = \mu_4$ foi rejeitada a hipótese alternativa $H_1 : \mu_3 > \mu_4$ é aceita, e isso significa que pode-se afirmar com 99% de certeza que os resultados conseguidos com a novo método Algoritmo *Genético-Learning Cooperativo* são em média melhores que os resultados obtidos com a Algoritmo Genético Tradicional.

É importante observar que os testes de hipóteses realizados comparam apenas o desempenho relativo ao valor da função objetivo, não considerando assim o aspecto relativo ao tempo de processamento. Para comparar também esse aspecto será apresentado na seção a seguir um método estatístico denominado análise de sobrevivência.

5.2.2 Análise de Sobrevivência

A Análise de Sobrevivência é um método estatístico aplicado originalmente a controle de mortalidade em sistemas biológicos ou falhas em sistemas mecânicos. A análise de sobrevivência utiliza um conjunto de procedimentos estatísticos apropriados ao estudo de variáveis observadas com base na ocorrência de um determinado evento. O estudo é elaborado supondo a existência de uma variável aleatória T que representa, por exemplo, o tempo de vida de um organismo vivo ou o tempo de utilidade de um dispositivo mecânico.

Na análise de sobrevivência existem duas características importantes [Ramos 2005]:

1. A distribuição da variável respostas é geralmente assimétrica, o que dificulta a análise através de procedimentos clássicos que necessitam da suposição de que a distribuição é gaussiana.
2. A presença da denominada censura, que acontece quando, sob determinadas circunstâncias inerentes ao problema em estudo, o registro da variável resposta não pode ser observado para certos indivíduos.

Na área das engenharias existem duas nomenclaturas para análise de sobrevivência: Análise de Confiabilidade ou Análise de Tempo de Falha, entretanto, a literatura estatística em geral utiliza termos associados à denominação Análise de Sobrevivência, como por exemplo, função de sobrevivência, tábua de vida, risco de morte, etc, o que indica que essa nomenclatura é mais amplamente utilizada, e portanto será a nomenclatura adotada neste texto.

Na análise de sobrevivência apresentada nesta seção a variável aleatória T representará o tempo decorrido entre o inicio da execução de um algoritmo e o momento t necessário para determinação da solução ótima de uma dada instância do Problema do Caixeiro Viajante- *PCV* em execução, esse momento t será denominado de "tempo de morte". A função de sobrevivência é expressa por:

$$S(t) = P(T > t) \tag{5.17}$$

que neste contexto denota a probabilidade de que o tempo de processamento necessário para se obter a solução ótima de uma determinada instância do *PCV* seja maior do que t. Um outro enfoque, e certamente o mais adequado ao se tratar de execução de algoritmos, é o uso da função de distribuição acumulada:

$$F(t) = P(T \leq t) = 1 - S(t). \tag{5.18}$$

onde $F(t)$ representa a probabilidade de que o tempo de processamento necessário para se obter a solução ótima seja inferior a um determinado tempo t, ou seja, considerando que quanto menor for o tempo de processamento envolvido nesta solução melhor será o desempenho do algoritmo, deseja-se obter um valor de $F(t)$ elevado para um tempo t curto, e um valor de $F(t)$ baixo para um tempo t longo.

A análise desenvolvida utiliza censura à direita, que ocorre dada a impossibilidade de se registrar o tempo necessário à obtenção da solução ótima, por ter sido estabelecido um tempo limite para a execução do algoritmo. Para o cálculo estimado da função de sobrevivência foi utilizado o estimador $\hat{S}(T)$ de Kaplan-Meier [Meier et al. 2004] que é dado pela proporção amostral do número de observações com um tempo maior do que t. O tempo de censura é determinado utilizando a equação:

$$T_c = \max_k(T_k) \tag{5.19}$$

$$T_k = (\sum_{j=1}^{n} t_{kj})/n, \quad \forall k = 1, ..., m \tag{5.20}$$

onde t_{kj} é o tempo de processamento do algoritmo k na execução j, n é o tamanho da

amostra e m é a quantidade de algoritmos envolvidos na análise.

A forma de calcular o tempo de censura expressa na equação 5.20 estabelece uma justa competição entre os algoritmos envolvidos na análise, pois, ao se escolher como tempo de censura o valor máximo dentre as médias do tempo de execução dos algoritmos, permite-se que o algoritmo de melhor desempenho tenha menor número de observações censuradas.

A análise de sobrevivência aqui desenvolvida testa para cada algoritmo o aspecto relativo ao tempo de processamento, considerando também a qualidade da solução no que diz respeito ao valor da função objetivo. Enquanto o teste de hipótese apresentado na seção anterior compara a qualidade dos algoritmos com base no valor da função objetivo, a analise de sobrevivência verifica qual algoritmo consegue encontra a solução ótima das instâncias em menor tempo de execução.

Análise de Sobrevivência para as Metaheurísticas GRASP

A análise de sobrevivência para os resultados experimentais das metaheurísticas GRASP, GRASP Reativo e *GRASP-Learning* foi elaborada utilizando a seguinte metodologia:

Para cada um dos algoritmos:

- Selecione inicialmente v_t como sendo o menor tempo de execução dentre todos os existentes (30 execuções);
- Calcule a função de sobrevivência estimada utilizando o estimador de Kaplan-Meier dado por:

$$S(v_t) = \frac{N_m}{n} \tag{5.21}$$

  onde N_m é o número de resultados cujos valores são maiores que v_t obtidos no experimento, v_t é o valor corrente no tempo t, e n é o número de execuções realizadas no experimento.
- De posse dos valores obtidos com a função de sobrevivência estimada, pode-se verificar o comportamento de cada algoritmo em relação ao tempo de censura estabelecido pela equação 5.20.

Na prática, a análise de sobrevivência pode ser descrita da seguinte forma: Para cada uma das 30 observações, cada algoritmo é executado até que o valor ótimo da instância em questão seja encontrado. Os tempos de execução das 30 observações são armazenados e dentre estas observações, aquelas que tiverem tempo de execução inferior ao tempo de censura ($v_t < T_c$) serão consideradas observações completas, caso contrário, tais observações serão consideradas censuradas. A ocorrência de censura pode ser interpretada como a incapacidade do algoritmo encontrar a solução ótima dentro de um limite de tempo previamente estabelecido.

A análise de sobrevivência foi realizada utilizando as instâncias $gr17$ e $pr76$. Os valores obtidos para as funções de sobrevivência, funções de distribuição acumulada e demais dados estatísticos relativos a análise de sobrevivência para as metaheurísticas GRASP, estão disponíveis nas Tabelas A.13 e A.14 do Apêndice 6.0.3.

O resultado da análise de sobrevivência é melhor visualizado através de representação gráfica, assim as análises obtidas com as instâncias *gr*17, e *pr*76 têm seus resultados apresentados nas Figura 5.10 e 5.11, respectivamente.

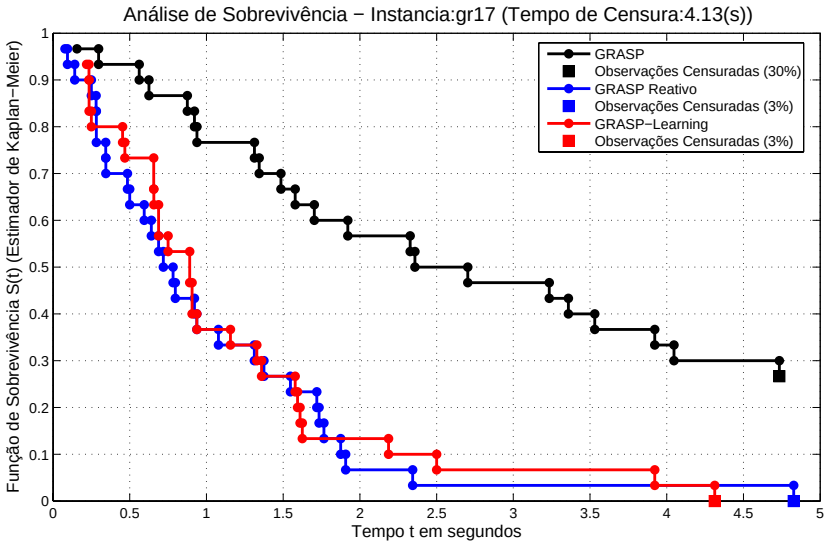


Figura 5.10: Análise de Sobrevivência para as Metaheurísticas GRASP, GRASP Reativo e *GRASP-Learning* (Instância gr17)

Para se interpretar os gráficos das figuras 5.10 e 5.11 é necessário entender que quanto menor for o tempo de execução de um algoritmo para atingir a solução ótima, melhor é a eficiência deste algoritmo. Desta forma, quanto mais uma curva está localizada à direita do gráfico, piores são os resultados do algoritmo que gerou esta curva. Pode-se ainda comparar o desempenho através do percentual de observações censuradas de cada algoritmo (as observações censuradas são indicadas nos gráficos pelo sinal ■)

A análise de sobrevivência realizada com as instâncias *gr*17, *pr*76 ratifica os resultados apresentados na seção 5.1.3, ou seja, através desta análise fica comprovado o melhor desempenho do método híbrido *GRASP-Learning* quando comparado com as metaheurísticas GRASP tradicional e GRASP Reativo.

Análise de Sobrevivência para os Algoritmos Genéticos

Para a análise de sobrevivência realizada nos resultados experimentais obtidos com o Algoritmos Genético Tradicional, *Genético-Learning* e *Genético-Learning Cooperativo* seguiu-se a mesma metodologia aplicada com as metaheurísticas GRASP. Como no caso da seção anterior foram utilizadas as instâncias *gr*17, *pr*76 e os valores obtidos para as

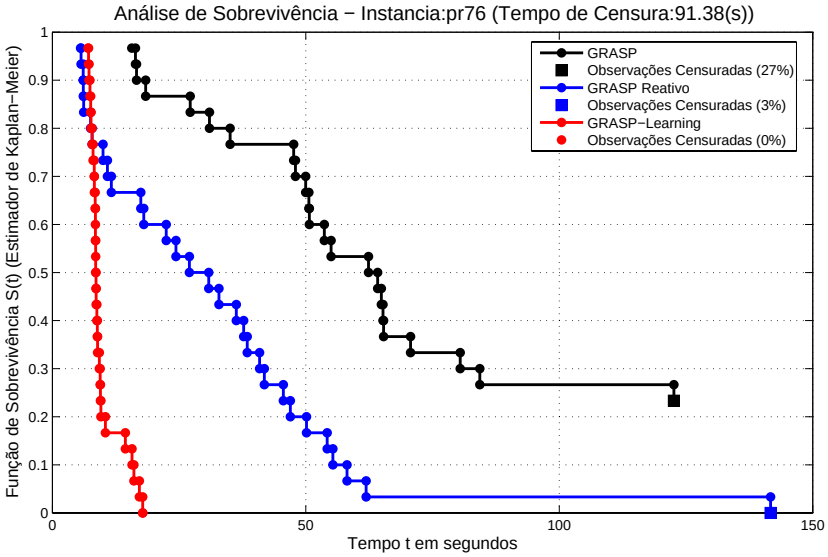


Figura 5.11: Análise de Sobrevivência para as Metaheurísticas GRASP, GRASP Reativo e *GRASP-Learning* (Instância pr76)

funções de sobrevivência, funções de distribuição acumulada e demais dados estatísticos relativos a análise de sobrevivência, estão disponíveis na Tabela A.15 e A.16 do Apêndice 6.0.3. As Figuras 5.12 e 5.13 apresentam o resultado da análise de sobrevivência para os algoritmos Genéticos.

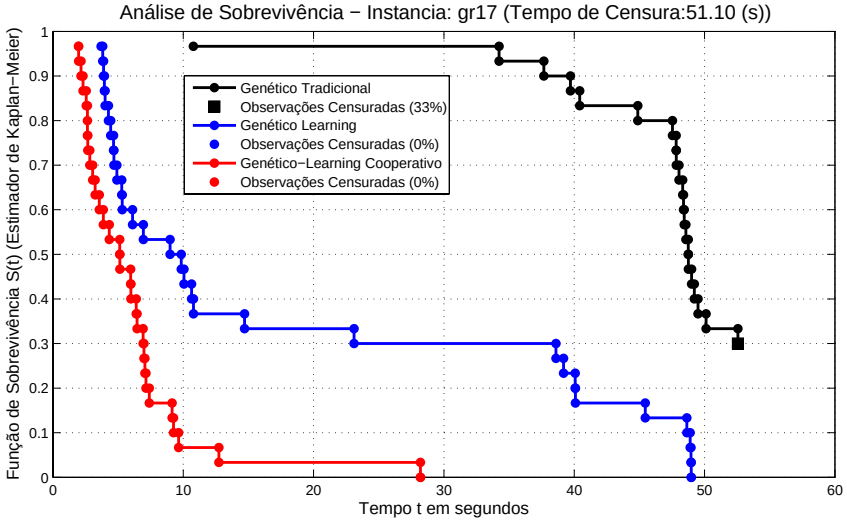


Figura 5.12: Análise de Sobrevivência para os Algoritmos: Genético, *Genético-Learning* e *Genético-Learning Cooperativo* (Instância gr17)

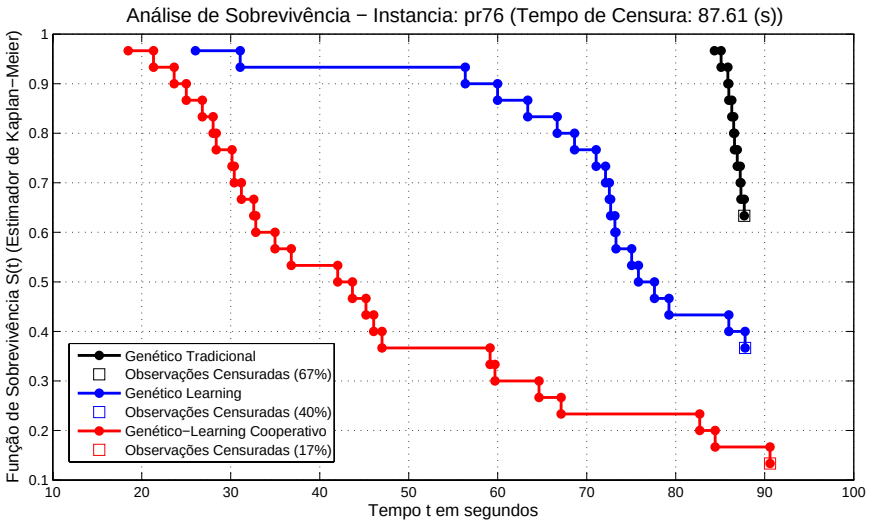


Figura 5.13: Análise de Sobrevivência para os Algoritmos: Genético, *Genético-Learning* e *Genético-Learning Cooperativo* (Instância pr76)

O resultado obtido com a análise de sobrevivência para os algoritmos genéticos demonstram um desempenho superior do algoritmo *Genético-Learning Cooperativo*. Este resultado da análise de sobrevivência aparentemente diverge dos resultados de desempenho apresentados na Seção 5.1.4, no que diz respeito ao tempo de processamento. Entretanto, é importante lembrar que os resultados apresentados na Figura 5.7, consideram o tempo integral de execução (critério de parada número máximo de gerações) de cada instância para cada um dos algoritmos, enquanto que os resultados da análise de sobrevivência foram obtidos com um experimento que executa cada algoritmo até que a solução ótima da instância em teste seja encontrada.

É importante observa ainda no teste de desempenho, os resultados relativos ao valor da função objetivo (Figura 5.6), neste resultados o algoritmo *Genético-Learning Cooperativo* obteve considerável vantagem sobre os demais, portanto, era esperado que nos experimentos feitos para a análise de sobrevivência tal versão do algoritmo tivesse melhor desempenho.

Considerando então os dois diferentes aspectos dos experimentos realizados, pode-se concluir que, em um teste comparando apenas o tempo de execução de cada algoritmo o algoritmo *Genético-Learning Cooperativo* será o mais lento, mas se o teste de desempenho considerar também a qualidade das soluções encontradas, este algoritmo obterá melhores resultados em menor tempo, utilizando como critério de parada localização da solução ótima.

5.3 Conclusão

Os resultados computacionais apresentados neste Capítulo demonstraram que os métodos híbridos propostos neste trabalho obtiveram melhor desempenho, quando comparados as versões tradicionais das metaheurísticas GRASP e Algoritmo Genético. Para validação dos resultados foram realizados dois testes Estatísticos: Teste de hipótese e Análise de Sobrevivência.

O teste de hipótese realizado teve como parâmetro de interesse o custo médio do percurso do caixeiro viajante determinado por cada um dos algoritmos, utilizando a instância *bays29*. Para realização dos teste foi escolhida esta instância por que a mesma apresentou resultados muito parecidos na comparação de desempenho dos algoritmos.

A análise de sobrevivência foi realizada com o propósito de verificar o desempenho dos algoritmos, considerando também o critério tempo de processamento, já que no teste de hipótese, a qualidade dos algoritmos foram comparadas considerando apenas o valor da médio da função objetivo. Este segundo teste foi realizado com duas instâncias: *gr17* e *pr76*.

A instância *gr17* foi escolhida levando em conta a praticidade. Os dados para a análise de sobrevivência realizada foram obtidos executando os algoritmos tendo com critério de parada a obtenção do ótimo. Como a instância *gr17* é pequena, permitiu que tal critério fosse alcançado com tranquilidade, para todos os algoritmos.

A instância *pr76* foi escolhida por que, dentre as instâncias utilizadas nos experimentos ela representa uma instância de médio porte, ou seja, é grande o suficiente pra ser interessante, e não tão grande a ponto de comprometer a praticidade do experimento. Nos resultados relativos a instância *pr76*, a análise de sobrevivência apresentou menor percentual de observações censuradas, tanto para metaheurística *GRASP-Learning* como para o *Genético-Learning Cooperativo*.

A partir dos resultados conseguidos com as duas instâncias utilizadas na análise de sobrevivência, e pela verificação de que o desempenho dos novos métodos propostos tende a melhorar para as instâncias maiores (ver seção 5.1.3 e 5.1.4) supõe-se então, que este comportamento represente uma tendência para as instâncias maiores utilizadas neste trabalho. Para as instâncias maiores a metodologia aplicada na análise de sobrevivência realizada poderá ser aplicada “relaxando-se” o critério de parada para um valor de aproximação do ótimo conhecido.

A análise Estatística realizada ratifica os resultados apresentados, os quais, têm as listagens na íntegra disponibilizados no apêndice 6.0.3.

Capítulo 6

Conclusão

Neste Capítulo serão apresentadas algumas observações, conclusões e perspectivas do trabalho. O texto a seguir será subdividido em seções de acordo com cada método proposto.

6.0.1 Sobre o método GRASP-Learning

O método *GRASP-Learning* proposto vem suprir duas necessidades da metaheurística GRASP tradicional: o fato de seu bom desempenho final está condicionado a soluções iniciais de boa qualidade, e a falta de um mecanismo de memória de uma iteração para outra.

No que diz respeito a qualidade das soluções iniciais, a utilização do algoritmo *Q-learning* como construtor destas soluções apresentou resultados promissores, tanto em relação ao valor da função objetivo, quanto no que diz respeito ao tempo de processamento, isto foi demonstrado pela comparação do desempenho dos algoritmos da fase construtiva de cada método (seção 4.4).

A proposta do algoritmo *Q-learning* como um algoritmo guloso-aleatório para a fase construtiva da metaheurística GRASP, além de suprí-la de boas soluções iniciais, prover também uma forma de memória adaptativa que possibilita que boas decisões tomadas em iterações passadas possam ser repetidas no futuro. Este mecanismo de memória adaptativa é implementado através do uso das informações contidas na matriz dos Q-valores gerada pelo algoritmo *Q-learning*. O termo memória adaptativa é aqui utilizado dado ao fato de que a matriz dos Q-valores é atualizada a cada episódio do *Q-learning*, incorporando assim em cada atualização a experiência obtida pelo agente de aprendizagem.

Na comparação do desempenho geral a metaheurística *GRASP-learning* obteve melhores resultados que as versões do GRASP tradicional e GRASP Reativo. No que diz respeito ao tempo de processamento o bom desempenho do método *GRASP-Learning* é notório principalmente a medida que o tamanho das instâncias crescem, isto é justificado dado ao fato de que a diferença entre o novo método e as versões do GRASP tradicional e reativo está apenas na fase construtiva e que o algoritmo parcialmente guloso utilizados nas versões GRASP e GRASP reativo tem complexidade $O(n^2)$, enquanto o algoritmo *Q-learning* tem complexidade $O(Nep * n)$, onde Nep é o número de episódios e n o tamanho da instância. Como a atualização dos Q-valores durante a execução do *GRASP-learning* é cumulativa (em k iterações do algoritmo *GRASP-learning* são executados $k * Nep$ epi-

sódios do *Q-learning*), o algoritmo *Q-learning* pode ser parametrizado com um valor de Nep relativamente pequeno, assim, com base nas ordens de complexidade dos algoritmos, a medida que as instâncias crescem o algoritmo *Q-learning* supera o algoritmo parcialmente guloso.

Outro aspecto importante que justifica o menor tempo de processamento obtido pelo método híbrido é a boa qualidade das soluções iniciais construídas pelo algoritmo *Q-learning*, pois, a metaheurística GRASP partindo de boas soluções iniciais teve o processo de busca local acelerado.

A análise estatística realizada ratificou os bons resultados obtidos para o método *GRASP-learning*, tendo o teste de hipótese confirmado seu desempenho superior para os valores da função objetivo. Na análise de sobrevivência, que é um teste estatístico que considera simultaneamente o valor da função objetivo e tempo de processamento, o algoritmo *GRASP-learning* foi o que apresentou o menor número de observações censuradas.

6.0.2 Sobre o Algoritmo Genético-Learning

O algoritmo Genético Híbrido proposto neste trabalho apresentou resultados bastante significativos principalmente a sua versão cooperativa. A idéia de fazer a atualização da matriz dos Q-valores a partir das soluções elites de cada população produziu uma expressiva melhoria no desempenho do método, principalmente no que diz respeito ao valores da função objetivo. Em relação ao tempo de processamento a versão tradicional obteve melhor desempenho, o que já era esperado, uma vez que tal versão constrói a população inicial de forma aleatório, enquanto que as versões com aprendizagem utilizam o algoritmo *Q-learning*, e portanto, tem adicionado ao seu tempo de processamento o tempo de execução dos episódios.

Apesar dos resultados inexpressivos em relação ao tempo de processamento, as versões dos algoritmos genéticos com aprendizagem conseguiram melhorar significativamente o valor da função objetivo, e isso já era notável quando feita a comparação da qualidades das populações iniciais (Seção 5.1.2) pois a população gerada pelo algoritmo *Q-learning* obteve melhor qualidade (melhor *Fitness*) e taxa de diversificação equivalente para as instâncias maiores.

Uma observação importante, diz respeito a análise de sobrevivência elaborada para estes algoritmos, pois apesar das versões com aprendizagem obterem pior tempo de processamento, elas conseguem encontrar melhores resultados em relação a localização da solução ótima, pois as mesmas obtiveram menor número de observações censuradas do que a versão do algoritmo genético tradicional.

Uma outra contribuição deste método, decorrente da versão cooperativa é a forma como o algoritmo *Q-learning* e os operadores genéticos cooperam mutuamente trocando informações no decorrer do processo evolutivo. Este processo de cooperação oferece um leque de possibilidades para a implementação paralela destes algoritmos, utilizando por exemplo, uma estratégia cooperativa/competitiva na resolução do problema do Caixeiro Viajante.

6.0.3 Trabalhos Futuros

Os métodos propostos neste trabalho foram testados apenas com o problema do caixeiro viajante simétrico. Apesar do *PCV* ser um problema clássico de otimização combinatória, a partir do qual muitos problemas práticos podem ser derivados, a aplicação dos métodos propostos neste trabalho a outros problemas desta classe requer que os mesmos sejam cuidadosamente modelados. Dentre as preocupações durante o processo de modelagem de problemas para a aplicação dos métodos aqui propostos destacam-se:

- Atentar para o tamanho da cardinalidade do conjunto de estados, a fim de não cair na "maldição da dimensionalidade". Neste trabalho o *PCV* foi modelado tendo um conjunto de estados que tem a cardinalidade do tamanho da instância, ou seja, o número de estado equivale ao número de cidades na rota.
- Verificar a possibilidade de ocorrência de estados com características parcialmente observáveis. Dependendo do contexto do problema, existem situações em que a tarefa de aprendizagem por reforço enfrentará dificuldades em escolher ações ótimas em domínios estocásticos parcialmente observáveis. Problemas neste contexto deverão ser modelados como Processos de Decisão de Markov Parcialmente Observáveis (*PDMPO*), para os quais existem abordagens específicas que tratam o problema. O *PCV* modelado neste trabalho apresenta estados com características totalmente observáveis, ou seja, as informações contidas en cada estado (a identificação da cidade e as distâncias entre ela e todas as outras) é suficiente para capacitar o agente aprendiz na tomada de decisão.
- Verificar possíveis características de não estacionalidade do processo de aprendizagem. Existem situações em que a tarefa de aprendizagem modelada atuará em um domínio em que a localização do objetivo, ou mesmo a estrutura do ambiente, pode mudar ao longo do tempo. Nestes casos se faz necessário o uso de técnicas específicas para a situação problema, assim também como a preocupação com a verificação da convergência do método.

Outro fator importante que deverá ser melhorado, diz respeito ao tamanho das instâncias de teste, pois por uma questão de celeridade na validação dos métodos, foram utilizadas instâncias do *PCV* de pequeno e médio porte.

Com base no trabalho desenvolvido e ciente das melhorias que podem ser conseguidas, existe a perspectiva dos seguintes trabalhos futuros:

- Executar testes computacionais com instâncias do *PCV* com maior número de cidades, com o objetivo de verificar o comportamento dos métodos propostos diante de instâncias de grande porte.
- Aplicar a metaheurística *GRASP-Learning* e Algoritmo *Genético-Learning* a outros problemas de Otimização Combinatória.
- Efetuar a implementação paralela híbrida para o problema do caixeiro viajante tendo como base os métodos híbridos propostos neste trabalho. Na realidade este trabalho já está em desenvolvimento como dissertação de mestrado do *PPgEEC* [Queiroz 2009] e tem apresentado resultados bastante interessantes.

- Investigar o uso de Aprendizagem por Reforço - algoritmo *Q-learning*- na melhoria de outras metaheurísticas.

Referências Bibliográficas

Abramson, Myriam & Wechsler Harry (2003), A distributed reinforcement learning approach to pattern inference in Go, *em* 'ICMLA', pp. 60–65.

Acosta, Espejo. Luis Gonzalo & Galvão. Roberto D. (2002), O uso das relaxações lagrangeana e surrogate em problemas de programação inteira, Vol. 22, Pesquisa Operacional, Print ISSN 0101 7438, Rio de Janeiro, pp. 387–402.

Agarwal, Pankaj K. & Sandeep Sen (2001), Randomized algorithms for geometric optimization problems. handbook of randomization, *em* 'Handbook of Randomized Computation', Kluwer Academic Publishers, pp. 151–201.

Akker, Marjan Van Den, Han Hoogeveen & Steef L. Van de Velde (2002), 'Combining column generation and lagrangean relaxation to solve a single-machine common due date problem', *INFORMS Journal on Computing* **14**(1), 37–51.

Alves, Daniela Pereira, Li Weigang & Bueno Borges de Souza (2006), Using meta-level control with reinforcement learning to improve the performance of the agents, *em* 'FSKD', pp. 1109–1112.

Atiya., A. F., A. Parlos & L. Ingber (2003), A reinforcement learning method based on adaptive simulated annealing, Vol. 1, Proceedings IEEE Midwest Symp Circuits Systems, Cairo, Egypt, pp. 121–124.

Backer, Bruno De, Philip Kilby, Patrick Prosser & Paul Shaw (2000), 'Solving vehicle routing problems using constraint programming and metaheuristics', *Journal of Heuristics* **6**, 501–523.

Barbosa, Marco A. C., Toscani. Laira V. & Ribeiro. Leila (2001), 'Anac uma ferramenta para análise automática da complexidade de algoritmos', **5**(8), 57–65.

Barros, C. A. (2001), Uma aplicação de grasp na otimização do emprego da unidade móvel de pistoneio, Dissertação de mestrado, Natal-RN, Brasil.

Bellman, Richard (1962), 'Dynamic programming treatment of the travelling salesman problem', *Journal ACM* **9**(1), 61–63.

Belluzzo, Luciano & Reinaldo Morabito (2005), Otimização nos padrões de corte de chapas de fibra de madeira reconstituída: um estudo de caso, Vol. 25, Pesquisa Operacional, pp. 391–415.

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

Blazewicz, Jacek, Piotr Formanowicz, Marta Kasprzak, Wojciech T. Markiewicz & Aleksandra Swiercz (2004), 'Tabu search method for dna sequencing by hybridization with isothermic libraries', *Computational biology and chemistry* **28 (1)**, 11–19.

Cancela, Héctor, Franco Robledo & Gerardo Rubino (2004), 'A grasp algorithm for designing a wide area network backbone', *Journal of Computer Science and Technology* **4(1)**, 52–58.

Cavicchio, D. J. (1970), Adaptive Search Using Simulated Evolution, Tese de doutorado, Michigan, USA.

Chang, H. S. (2004), An ant system based exploration-exploitation for reinforcement learning, Vol. 4, Systems, Man and Cybernetics, IEEE International Conference on Publication, pp. 3805–3810.

Chen, Dingjun, Chung-Yeol Lee & Cheol Hoon Park (2005), Hybrid genetic algorithm and simulated annealing (hgasa) in global function optimization, *em* 'ICTAI '05: Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence', IEEE Computer Society, Washington, DC, USA, pp. 126–133.

Crites, Robert H. & Andrew G. Barto (1996), Improving elevator performance using reinforcement learning, *em* D. S.Touretzky, M. C.Mozer & M. E.Hasselmo, eds., 'Advances in Neural Information Processing Systems', Vol. 8, The MIT Press, pp. 1017–1023.

Croes, G. A. (1958), A method for solving traveling salesman problems, *em* 'Operations Research', Vol. 6, pp. 791–812.

Dantzig, G. B., D. R. Fulkerson & S. M. Jonson (1954), 'Solutions of a large scale traveling salesman problem', *Operations Research* (12), 393–410.

Du, D. Z., Y. Zhang & Q. Feng (1991), 'On better heuristic for euclidean steiner minimum trees', *Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on* pp. 431–439.

Ernst, D., G.B. Stan, J. Goncalves & L. Wehenkel. (2006), Clinical data based optimal sti strategies for hiv: a reinforcement learning approach, *em* 'In Proceedings of the Machine Learning Conference of Belgium and The Netherlands (Benelearn)', pp. 65–72.

Fang, H. (1994), Genetic Algorithms in Timetabling and Scheduling, Tese de doutorado, Scotland, UK.

Faroe, O., D. Pisinger & M. Zachariasen (2003), 'Guided local search for the three-dimensional bin packing problem', **15**, 267–283.

Feo, T. & M. Resende (1995), Greedy randomized adaptive search procedures, Vol. 6, Journal of Global Optimization, pp. 109–133.

Fleurent, Charles & Fred Glover (1999), 'Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory', *INFORMS Journal on Computing* **11**(2), 198–204.

Fonteneau, R., L. Wehenkel & D. Ernst (2008), Variable selection for dynamic treatment regimes: a reinforcement learning approach, *em* 'In Proceedings of the European Workshop on Reinforcement Learning (EWRL'08), Lille, France', pp. 312–319.

Frantz, D.R. (1972), Non-linearities in genetic adaptive search, Tese de doutorado, Michigan, USA.

Gambardella, Luca Maria & Marco Dorigo (1995), Ant-q: A reinforcement learning approach to the traveling salesman problem, *em* 'International Conference on Machine Learning', pp. 252–260.

Garey, M. R. & D. S. Johnson (1979), *Computers and Intractability: A Guide to the Theory of NPCompleteness*, W. E. Freeman and company, New York, USA.

Gedson, Faria. & Roseli Francelin Romero (1999), Explorando o potencial de algoritmos de aprendizado com reforço em robôs móveis, Proceedings of the IV Brazilian Conference on Neural Networks - IV Congresso Brasileiro de Redes Neurais, ITA, São José dos Campos - SP - Brazil, pp. 237–242.

Gilbert, E. N. & H. O. Pollak (1968), 'Steiner minimal trees', *SIAM Journal on Applied Mathematics* **16**(1), 1–29.

Glover, F. (1986), 'Future paths for integer programming and links to artificial intelligence', *Computers and Operations Research* **5**, 533–549.

Guelpeli, Marcus V. C., Carlos H. C. Ribeiro & Nizam Omar (2004), 'Utilização de aprendizagem por reforço para modelagem autônoma do aprendiz em um tutor inteligente.', *Revista Brasileira de Informática na Educação, SBC* **12**(II), 69–77.

Guo, M., Y. Liu & Malec J. (2004), A new q-learning algorithm based on the metropolis criterion, Vol. 34, IEEE Trans Syst Man Cybern B Cybern, pp. 2140–2143.

Hochbaum, Dorit S., ed. (1997), *Approximation algorithms for NP-hard problems*, PWS Publishing Co., Boston, MA, USA.

Holland, John (1975), *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan, USA.

Hollstein, R.B. (1971), Artificial Genetic Adaptation in Computer Control Systems, Tese de doutorado, Michigan, USA.

Johnson, David S. (1974), 'Approximation algorithms for combinatorial problems', *Journal of Computer and System Sciences (JCSS)* **9**(3), 256–278.

Karp, R.M. (1975), On the computational complexity of combinatorial problems, Vol. 5, Networks, pp. 45–68.

Kim, Dong Hwa, Ajith Abraham & Jae Hoon Cho (2007), 'A hybrid genetic algorithm and bacterial foraging approach for global optimization', *Information Sciences* **177**(18), 3918–3937.

Krasnogor, P. & M. Moscato (1995), A new hybrid heuristic for large geometric traveling salesman problem based on delaunay triangulation, Anales del XXVII Simposio Brasileiro de Pesquisa Operacional, pp. 6–8.

Kureichick, V. M., A. N. Melikhov, V. V. Miagkikh, O. V. Savelev & A. P. Topchy (1996), Some new features in genetic solution of the travelling salesman problem, *em* 'Adaptive Computing in Engineering Design and Control '96 (ACEDC'96), 2nd International Conference of the Integration of Genetic Algorithms and Neural Network Computing and Related Adaptive Techniques with Current Engineering Practice', pp. 321–337.

Lima Junior, F. C. (2002), Otimização das intervenções em poços de petróleo por sondas de produção terrestre: Uma abordagem metaheurística, Dissertação de mestrado, Natal-RN, Brasil.

Mateus, Geraldo R., Henrique P. L. Luna & Adriana B. Sirihal (2000), 'Heuristics for distribution network design in telecommunication', *Journal of Heuristics* **6**(1), 131–148.

Meier, Paul, Theodore Karrison, Rick Chappell & Hui Xie (2004), 'The price of kaplan meier', *Journal of the American Statistical Association* **99**(467), 890–896.

Merz, Peter & Bernd Freisleben (2001), 'Memetic algorithms for the traveling salesman problem', *Complex Systems* **13**, 297–345.

Miagkikh, Victor V. & William F. Punch III (1999), An approach to solving combinatorial optimization problems using a population of reinforcement learning agents, Vol. 2, Morgan Kaufmann, Orlando, Florida, USA, pp. 1358–1365.

Miller, David M., Hui-Chuan Chen, Jessica Matson & Qiang Liu (1999), 'A hybrid genetic algorithm for the single machine scheduling problem', *Journal of Heuristics* **5**(4), 437–454.

Mladenovic, N. & P. Hansen (1997), 'Variable neighborhood search', *Computers and Operations Research* **24**(11), 1097–1100.

Olson, Daniel Kenneth (1993), Learning to play games from experience: An application of artificial neural networks and temporal difference learning, Dissertação de mestrado, Washington - USA.

Osman, I. H. & J.P. Kelly (1996), *Metaheuristics. An overview*, Kluwer Academic Publishers, Boston - USA.

Pardalos, Panos M., Tianbing Qian, Mauricio & G. C. Resende (1994), A greedy randomized adaptive search procedure for the quadratic assignment problem, *em* 'Quadratic assignment and related problems, DIMACS Series on Discrete Mathematics and Theoretical Computer Science', Vol. 16, American Mathematical Society, pp. 237–261.

Pettinger, James E. & Richard M. Everson (2002), Controlling genetic algorithms with reinforcement learning, *em* 'GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference', Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, p. 692.

Prais, Marcelo & Celso C. Ribeiro (2000), 'Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment', *Journal on Computing* **12**(3), 164–176.

Puterman, M. L. (2005), *Markov Decision Processes Discrete Stochastic Dynamic Programming*, John Wiley e Sons, Inc, New York, USA.

Queiroz, J. P. Santos (2009), Uma implementação paralela híbrida para o problema do caixeiro viajante usando algoritmos genéticos, grasp e aprendizagem por reforço, Dissertação de mestrado, Natal- RN - Brasil.

Ramos, Iloneide C. O., Adrião D. Dória Neto & Fernando C. de Miranda (2003), Uma abordagem didática do simulated annealing usando o modelo markoviano aplicada ao problema do caixeiro viajante, XXXV SBPO: A Pesquisa Operacional e os Recusrsos Renováveis, Natal - RN - Brasil, pp. 01–06.

Ramos, Iloneide Carlos de Oliveira (2005), Metodologia estatística na solução do problema do caixeiro viajante e na avaliação de algoritmos: um estudo aplicado à transgenética computacional, Tese de doutorado, Nata-RN, Brasil.

Randy, Haupt. & Sue Ellen Haupt (1998), *Patrical Genetic Algoritms, Second Edition*, Wiley Interscience, Hoboken, New Jersey - USA.

Reinelt, G. (n.d.), *The Traveling Salesman: Computational Solutions for TSP Applications*, Vol. 840, Springer Berlin, Heidelberg, Berlin - Germany.

Resende, M. & C. Ribeiro (2005), *GRASP with Path-relinking: Recent Advances and Applications*, Springer.

Resende, M. G. C. & T. A. Feo (1996), A grasp for satisfiability, *em* D. S.Johnson & M. A.Trick, eds., 'Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Series In Discrete Mathematics and Theoretical Computer Science', Vol. 26, AMS, pp. 499–520.

Resende, Mauricio G. C., Jorge Pinho de Sousa & Ana Viana, eds. (2004), *Metaheuristics: computer decision-making*, Kluwer Academic Publishers, Norwell, MA, USA.

Rosenberg, R. S. (1967), Simulation of genetic populations with biochemical properties, Tese de doutorado, Michigan, USA.

Rothfarb, B., H. Frank, D. M. Rosenbaum, K. Steiglitz & D. J. Kleitman (1970), 'Optimal Design of Offshore Natural-Gas Pipeline Systems', *OPERATIONS RESEARCH* **18**(6), 992–1020.

S. Girgin, Ph. Preux (n.d.), *Feature discovery in reinforcement learning using genetic programming*, Vol. 4971, Springer Berlin, Heidelberg, Berlin - Germany.

Scardua., Leonardo Azevedo, José Jaime da Cruz. & Anna Helena Costa Reali. (2002), 'Optimal control of ship unloaders using reinforcement learning', *Advanced Engineering Informatics* **16**(3), 217–227.

Senne, E. L. F. & L. A. N. Lorena (2000), Lagrangean/surrogate heuristics for p-median problems., Kluwer Academic Publishers, pp. 115–130.

Silveira, Rejane & Morabito. Joas (2002), 'A heuristic method based on dynamic programming for the constrained two-dimensional guillotine cutting problem.', *Gestão e Produção* **9**(1), 78–92.

Singh, Satinder & Dimitri Bertsekas (1997), Reinforcement learning for dynamic channel allocation in cellular telephone systems, *em* 'In Advances in Neural Information Processing Systems: Proceedings of the 1996 Conference', MIT Press, pp. 974–980.

Sutton, R.S. & A.G. Barto (1998), *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA.

Thangiah, S. (1995), *Vehicle Routing with Time Windows using Genetic Algorithms*, Vol. 2, CRC Press.

TSPLIB (2008), 'Tsplib - a traveling salesman problem library'.
URL: *http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/ acessado em dezembro de 2008.*

Veach, Eric. (1997), Robust Monte Carlo Methods for Light Transport Simulation, Tese de doutorado, Stanford, CA, USA.

Viana, G. Valdisio R. (1998), *Metaheurísticas e Programação Paralela em Otimização Combinatória*, UFC Edições, Fortaleza - CE, Brasil.

Wang, Xiao-Dong & Tom Chen (1995), 'On performance and area optimization of vlsi systems using genetic algorithms', *VLSI Design* **3(1)**, 43–51.

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

Wei-Chang Yeh, Ali Allahverdi (2004), *A branch-and-bound algorithm for the three-machine flowshop scheduling problem with bicriteria of makespan andtotal flow-time*, Vol. 11, Department of Industrial Engineering, Feng Chia University, Taiwan; Department of Industrial and Management Systems Engineering, College of Engineering and Petroleum, Kuwait University.

White, A., J. Mann & G. Smith (1999), Genetic algorithms and network ring design, *em* 'Annals of Operations Research', Vol. 86, pp. 347–371.

Whitley, Darrell (1994), 'A genetic algorithm tutorial', *Statistics and Computing* **4**, 65–85.

Whitley, L. Darrell (2000), A comparison of genetic algorithms for the static job shop scheduling problem, *em* 'Parallel Problem Solving from Nature', Springer, pp. 303–312.

Wiering, Marco (2000), Multi-agent reinforcement learning for traffic light control, *em* 'Proc. 17th International Conf. on Machine Learning', Morgan Kaufmann, San Francisco, CA, pp. 1151–1158.

Zhang, Wei & Thomas G. Dietterich (1996), High-performance job-shop scheduling with A time-delay TD(λ) network, *em* D. S.Touretzky, M. C.Mozer & M. E.Hasselmo, eds., 'Advances in Neural Information Processing Systems', Vol. 8, The MIT Press, pp. 1024–1030.

Zhang, Xin Li and Xiaojun Shen and Yuanwei Jing and Siying (2007), Simulated annealing-reinforcement learning algorithm for abr traffic control of atm networks, *em* '46th IEEE Conference on Decision and Control, New Orleans, LA, USA', pp. 5716 –5721.

Apêndice A

Listagem dos Resultados Experimentais

Resultados Experimentais das Metaheurísticas:
GRASP, GRASP Reativo, GRASP-Learning
Algoritmo Genético Tradicional, Algoritmo Genético-Learning
e Algoritmo Genético-Learning Cooperativo

Resultados Experimentais: GRASP Tradicional

Tabela A.1: Resultados Obtidos para o Metaheurística GRASP Tradicional (Valor da Função Objetivo)

Nº	INSTÂNCIAS TSPLIB									
Iter.	gr17	bays29	swiss42	gr48	berlin52	pr76	gr120	ch150	si175	a280
1	2085,00	2122,00	1345,00	5794,00	8781,00	144660,00	10745,00	10860,00	26006,00	5942,00
2	2085,00	2080,00	1468,00	5903,00	8441,00	140250,00	10510,00	10771,00	25734,00	5916,00
3	2085,00	2102,00	1374,00	5593,00	8925,00	135240,00	10761,00	11081,00	25993,00	5882,00
4	2085,00	2101,00	1426,00	5670,00	8665,00	148030,00	10011,00	10692,00	25595,00	6007,00
5	2085,00	2135,00	1409,00	5776,00	8996,00	148900,00	10823,00	10752,00	25704,00	5670,00
6	2085,00	2020,00	1471,00	5890,00	9073,00	143260,00	10819,00	11242,00	25863,00	5832,00
7	2085,00	2125,00	1457,00	5963,00	8790,00	133790,00	10579,00	10696,00	25464,00	5562,00
8	2085,00	2061,00	1392,00	5799,00	8453,00	138430,00	10717,00	11295,00	25769,00	6030,00
9	2085,00	2116,00	1447,00	5633,00	8769,00	140950,00	10888,00	10231,00	25865,00	6024,00
10	2085,00	2055,00	1438,00	5944,00	8397,00	139010,00	10617,00	11151,00	25809,00	5847,00
11	2085,00	2050,00	1473,00	5937,00	8785,00	141020,00	10846,03	10499,42	25559,65	6005,20
12	2085,00	2075,00	1451,00	5854,00	8610,00	129170,00	10819,56	10540,09	25855,68	5944,90
13	2085,00	2114,00	1449,00	5967,00	8712,00	135700,00	10057,88	10888,60	25721,64	5998,10
14	2085,00	2077,00	1485,00	5596,00	8814,00	131530,00	10658,67	10513,47	25546,95	5707,60
15	2085,00	2094,00	1461,00	5839,00	8696,00	145020,00	10247,19	11108,70	25649,21	5688,30
16	2085,00	2067,00	1426,00	5498,00	8713,00	140810,00	10382,37	11277,70	25793,71	5813,20
17	2085,00	2123,00	1415,00	5850,00	8869,00	141610,00	10492,46	11008,64	25568,24	5638,70
18	2085,00	2062,00	1465,00	5846,00	8972,00	139810,00	10838,98	10597,03	25864,12	5660,50
19	2085,00	2068,00	1449,00	5664,00	8812,00	142410,00	10377,16	10853,07	25595,61	5663,90
20	2085,00	2053,00	1444,00	5892,00	8962,00	148210,00	10874,86	10345,32	25962,45	5867,30
21	2085,00	2173,00	1438,00	6002,00	8836,00	139610,00	10275,64	11196,53	25610,46	5586,70
22	2085,00	2079,00	1395,00	5817,00	9063,00	142880,00	10626,38	11167,65	25879,66	5669,90
23	2085,00	2082,00	1521,00	5835,00	8761,00	140330,00	10596,19	11101,41	25566,77	5875,60
24	2085,00	2115,00	1385,00	5810,00	9169,00	141580,00	10484,43	10508,82	25620,35	5707,90
25	2085,00	2100,00	1471,00	5806,00	8475,00	139020,00	10623,48	10863,72	25513,66	5705,70
26	2085,00	2046,00	1463,00	5867,00	8633,00	137290,00	10596,12	10254,97	25776,42	5899,40
27	2085,00	2058,00	1476,00	5820,00	8856,00	146020,00	10167,59	10683,53	25835,84	6009,70
28	2085,00	2066,00	1457,00	5887,00	8716,00	138500,00	10123,23	10564,33	25760,83	5623,20
29	2085,00	2100,00	1425,00	5693,00	8872,00	141230,00	10888,38	10402,11	25695,26	5595,00
30	2085,00	2050,00	1467,00	5608,00	8806,00	140610,00	10161,58	10421,61	25813,61	5620,20
$\bar{x}$	2085,00	2085,63	1441,43	5801,77	8780,73	140496,00	10553,61	10785,59	25733,07	5799,77
S	0,00	33,00	37,05	127,70	188,39	4564,76	269,98	321,35	147,19	155,37

Tabela A.2: Resultados Obtidos para a Metaheurística GRASP Tradicional (Tempo de Processamento)

Nº	INSTÂNCIAS TSPLIB									
Iter.	gr17	bays29	swiss42	gr48	berlin52	pr76	gr120	ch150	si175	a280
1	25,39	116,30	395,94	576,28	766,83	1337,80	5010,20	11143,00	20954,00	40017,00
2	25,28	111,72	373,58	561,92	748,38	1345,40	5041,10	11149,00	21537,00	40875,00
3	25,14	109,14	374,63	553,11	761,66	1352,60	4973,80	11195,00	21142,00	40535,00
4	25,53	110,25	372,17	557,98	752,92	1331,60	5999,00	11215,00	21897,00	40312,00
5	25,36	111,08	374,97	560,77	756,03	1332,60	4958,00	11264,00	21143,00	40326,00
6	25,19	110,25	373,94	560,47	761,19	1318,30	5956,10	11222,00	21807,00	40273,50
7	25,30	109,74	373,47	554,30	743,63	1325,20	5026,90	11017,00	22148,00	40663,00
8	24,92	110,64	374,36	554,39	757,91	1333,50	5040,10	11294,00	21186,00	40712,50
9	25,23	110,78	371,55	558,52	740,58	1328,80	5005,40	11260,00	21582,00	40780,50
10	25,28	111,42	371,95	556,78	758,11	1318,60	5033,10	11160,00	21719,00	40697,00
11	25,49	110,97	368,33	556,88	758,53	1311,00	4963,17	11233,25	21081,85	39672,50
12	25,44	111,59	373,34	562,92	748,58	1333,40	5015,39	11134,22	22103,62	40228,00
13	25,14	109,86	373,78	556,98	753,95	1329,40	4961,83	11042,67	20959,35	40818,40
14	25,34	111,80	374,24	552,58	750,76	1336,60	4964,80	11091,84	21880,51	40863,01
15	25,44	113,52	371,81	560,64	752,49	1325,40	5001,06	11059,34	21930,40	40537,96
16	25,70	111,09	370,39	553,84	753,91	1329,80	4966,40	11095,78	21992,08	40340,36
17	25,02	111,11	373,14	552,75	745,05	1352,00	5026,53	11139,68	21054,24	40428,75
18	25,23	111,34	375,97	553,34	753,97	1333,40	5026,42	11163,01	21431,12	40519,79
19	25,22	111,80	371,89	565,13	753,91	1334,90	5018,66	11144,60	21264,18	40631,62
20	24,89	110,53	370,58	570,92	755,48	1334,60	4970,63	11260,39	21910,24	40431,23
21	25,42	111,47	373,06	569,88	752,09	1326,10	5013,29	11161,92	21469,42	40636,48
22	25,11	116,64	372,69	563,47	747,16	1332,70	5001,43	11279,00	22042,05	40701,78
23	25,27	114,84	371,44	563,47	763,34	1325,90	5039,02	11194,46	21171,90	40406,76
24	25,69	114,06	376,38	566,19	742,97	1331,80	5012,98	11283,42	21269,94	40344,11
25	25,22	114,73	368,61	560,33	752,09	1336,20	5025,17	11083,46	21127,49	40452,25
26	25,31	115,31	370,58	555,88	745,63	1326,10	4996,11	11204,77	21116,49	40465,31
27	25,09	114,80	377,28	555,34	741,14	1342,70	4994,37	11097,32	21992,34	40528,84
28	25,22	114,50	362,78	555,92	751,89	1334,60	5027,20	11203,78	21646,90	40579,20
29	25,50	115,20	380,55	558,73	752,88	1314,50	5965,49	11210,47	21611,37	40324,68
30	25,64	115,92	375,20	564,64	748,00	1329,80	4969,34	11035,04	21127,11	40432,00
$\bar{x}$	25,30	112,41	373,62	559,81	752,37	1331,51	5100,10	11167,91	21509,92	40484,48
S	0,20	2,21	5,25	5,87	6,53	9,38	297,33	78,29	387,96	253,16

Resultados Experimentais: GRASP Reativo

Tabela A.3: Resultados Obtidos para a Metaheurística GRASP Reativo (Valor da Função Objetivo)

Nº	INSTÂNCIAS TSPLIB									
Iter.	gr17	bays29	swiss42	gr48	berlin52	pr76	gr120	ch150	si175	a280
1	2085,00	2020,00	1428,00	5470,00	8459,00	138050,00	8875,00	9286,00	23394,00	4179,00
2	2085,00	2050,00	1323,00	5468,00	8280,00	134110,00	8959,00	9213,00	23907,00	4270,00
3	2085,00	2075,00	1361,00	5370,00	8573,00	132340,00	8833,00	9377,00	23688,00	4278,00
4	2085,00	2031,00	1389,00	5508,00	8408,00	131880,00	8737,00	8999,00	24018,00	4342,00
5	2085,00	2033,00	1396,00	5262,00	8468,00	131340,00	8685,00	8898,00	23556,00	4294,00
6	2085,00	2062,00	1417,00	5473,00	8442,00	130030,00	8506,00	8964,00	23422,00	4150,00
7	2085,00	2080,00	1368,00	5446,00	8435,00	135160,00	9076,00	9179,00	23734,00	4293,00
8	2085,00	2050,00	1363,00	5733,00	8781,00	131620,00	8736,00	9488,00	23463,00	4254,00
9	2085,00	2068,00	1388,00	5494,00	8184,00	129620,00	9120,00	9624,00	23543,00	4173,00
10	2085,00	2059,00	1363,00	5681,00	8448,00	133190,00	8581,00	8887,00	23580,00	4253,00
11	2085,00	2086,00	1361,00	5269,00	8609,00	132610,00	8719,82	9488,66	23619,75	4292,50
12	2085,00	2061,00	1367,00	5503,00	8803,00	131370,00	8598,43	9555,04	23590,26	4317,70
13	2085,00	2069,00	1385,00	5280,00	8343,00	132350,00	8866,89	8980,85	23787,51	4341,30
14	2085,00	2020,00	1428,00	5450,00	8207,00	134010,00	8667,39	9561,93	23802,70	4247,20
15	2085,00	2067,00	1403,00	5683,00	8225,00	129120,00	8533,77	9353,68	23490,89	4271,40
16	2085,00	2026,00	1394,00	5445,00	8244,00	137180,00	8970,40	8958,76	23645,96	4302,30
17	2085,00	2053,00	1417,00	5356,00	8551,00	130380,00	8655,81	9092,74	23623,55	4236,70
18	2085,00	2020,00	1387,00	5344,00	8565,00	126950,00	8778,76	9290,39	23726,14	4251,90
19	2085,00	2082,00	1338,00	5374,00	8013,00	133520,00	8928,38	9593,66	23758,15	4183,30
20	2085,00	2104,00	1334,00	5481,00	8419,00	127880,00	8726,22	9599,17	23781,26	4175,60
21	2085,00	2047,00	1351,00	5587,00	8681,00	125610,00	8958,79	9003,71	23535,84	4192,10
22	2085,00	2068,00	1407,00	5366,00	8534,00	129560,00	8748,95	9603,03	23743,25	4170,20
23	2085,00	2056,00	1383,00	5431,00	8510,00	128870,00	8926,33	9593,28	23730,81	4177,30
24	2085,00	2119,00	1436,00	5448,00	8467,00	126080,00	8938,67	9245,05	23477,24	4238,60
25	2085,00	2065,00	1306,00	5407,00	8346,00	131810,00	8778,44	9477,10	23455,93	4302,50
26	2085,00	2069,00	1437,00	5557,00	8086,00	136480,00	8518,83	8991,82	23650,35	4205,00
27	2085,00	2060,00	1492,00	5589,00	8468,00	128810,00	8709,77	9198,69	23887,20	4193,20
28	2085,00	2067,00	1374,00	5226,00	8223,00	133340,00	8766,17	9562,32	23568,25	4325,50
29	2085,00	2086,00	1425,00	5548,00	8418,00	130000,00	8672,86	9471,95	23694,62	4151,50
30	2085,00	2061,00	1331,00	5376,00	8438,00	128140,00	8627,99	9595,03	23509,47	4263,80
$\bar{x}$	2085,00	2061,50	1385,07	5454,17	8420,93	131380,33	8773,36	9304,40	23646,14	4244,19
S	0,00	23,67	40,24	124,69	184,85	3114,31	161,32	256,21	152,58	58,79

Tabela A.4: Resultados Obtidos para a Metaheurística GRASP Reativo (Tempo de Processamento)

Nº	INSTÂNCIAS TSPLIB									
Iteração	gr17	bays29	swiss42	gr48	berlin52	pr76	gr120	ch150	si175	a280
1	17,90	94,60	373,10	534,50	594,90	1845,70	6341,10	11359,40	12868,00	37060,00
2	14,50	107,90	351,60	477,50	614,80	1903,00	5426,10	14108,10	12786,00	35383,30
3	17,60	106,50	337,70	522,80	601,80	1410,30	5888,50	11031,90	12819,00	35753,30
4	19,80	101,40	362,30	562,40	597,20	1908,10	6245,80	13507,90	12795,00	39186,70
5	21,50	103,80	340,60	533,00	595,20	1730,10	6598,90	13643,50	12836,00	35916,70
6	23,40	97,10	358,80	543,60	589,40	1391,80	6724,10	13808,50	12812,00	36413,30
7	24,00	105,20	353,50	508,30	587,20	1506,70	5965,60	11288,30	12854,00	36613,30
8	22,50	109,40	350,80	558,90	609,10	1676,30	5211,50	12301,90	12728,00	37450,00
9	14,60	93,90	346,30	548,80	614,90	1937,00	5231,00	11852,40	12778,00	35200,00
10	17,70	100,70	361,00	565,80	599,30	1940,00	5430,30	13588,10	12862,00	35360,00
11	22,30	113,70	349,90	551,80	601,00	1429,40	6506,20	12403,80	12810,40	35504,00
12	14,10	98,40	366,00	532,10	610,20	1944,40	5424,80	13943,40	12804,50	38747,70
13	22,80	104,40	334,20	533,00	584,70	1935,80	6457,30	11601,20	12850,40	38855,50
14	23,40	103,70	355,60	515,10	614,90	1637,80	5405,50	11864,40	12765,80	38406,50
15	19,30	104,90	345,70	557,00	595,30	1836,20	6669,20	11484,10	12772,60	35737,90
16	23,90	107,50	345,10	527,60	605,00	1419,50	5601,60	11454,10	12744,80	36362,60
17	23,20	106,60	357,40	485,80	619,70	1596,50	5318,30	13810,90	12860,90	36643,60
18	23,70	109,20	370,30	530,40	581,10	1909,70	5418,70	12881,00	12820,00	37961,90
19	19,80	101,20	363,40	551,80	586,30	1831,70	6091,70	12784,60	12795,20	35882,80
20	23,40	100,10	374,50	511,00	595,60	1937,80	5828,80	11482,10	12818,10	38120,60
21	17,70	103,30	336,20	551,10	592,10	1745,30	5604,50	13758,20	12804,70	35768,20
22	23,00	107,00	374,70	476,60	608,20	1352,70	6488,10	13016,40	12819,10	37861,20
23	17,50	108,10	337,00	528,60	592,50	1867,70	6035,20	12144,80	12804,50	37251,90
24	23,70	105,90	344,20	505,20	588,60	1921,20	5969,90	12666,00	12829,50	38341,00
25	23,90	103,40	344,70	531,80	596,00	1759,10	6647,20	12308,00	12801,90	38096,50
26	23,80	104,60	379,50	553,80	602,70	1809,50	5483,80	11261,20	12868,50	38818,20
27	24,00	107,90	376,80	575,50	585,50	1801,00	6352,50	11788,70	12758,40	38770,00
28	20,80	96,90	343,30	546,40	600,40	1578,30	6346,00	11413,70	12742,70	36638,70
29	24,00	113,40	344,90	558,50	612,20	1744,60	5657,10	11608,70	12743,70	38034,50
30	23,90	98,10	342,20	499,90	597,40	1438,20	6003,40	11788,50	12736,50	36117,50
$\overline{x}$	21,10	104,00	354,00	532,60	599,1	1724,8	5945,8	12398,5	12803,0	37075,2
S	3,20	5,00	13,40	26,00	10,3	198,9	487,5	980,0	41,1	1267,5

Resultados Experimentais: GRASP-Learning

Tabela A.5: Resultados Obtidos para a Metaheurística GRASP-Learning (Valor da Função Objetivo)

Nº	INSTÂNCIAS TSPLIB									
Iter.	gr17	bays29	swiss42	gr48	berlin52	pr76	gr120	ch150	si175	a280
1	2085,00	2036,00	1273,00	5486,00	8062,00	121440,00	9420,00	6811,00	20208,00	2967,00
2	2085,00	2020,00	1285,00	5524,00	8002,00	112930,00	9026,00	7207,00	18854,00	3013,00
3	2085,00	2031,00	1273,00	5355,00	8083,00	124540,00	9152,00	6848,00	19006,00	3002,60
4	2085,00	2020,00	1284,00	5491,00	8068,00	128260,00	9101,00	7038,00	21068,00	2999,00
5	2085,00	2028,00	1284,00	5349,00	8088,00	122850,00	8420,00	6871,00	20974,00	3008,00
6	2085,00	2020,00	1273,00	5488,00	8061,00	131440,00	9316,00	7003,00	21085,00	2979,90
7	2085,00	2031,00	1285,00	5479,00	8075,00	132930,00	9311,00	7075,00	20012,00	2986,00
8	2085,00	2033,00	1296,00	5393,00	7905,00	134540,00	8416,00	7331,00	21039,00	2977,50
9	2085,00	2038,00	1285,00	5479,00	8120,00	138260,00	8774,00	6857,00	21017,00	2968,20
10	2085,00	2026,00	1285,00	5511,00	8018,00	132850,00	8584,00	7230,00	20163,00	2970,70
11	2085,00	2034,00	1273,00	5446,00	8088,00	132850,00	8975,00	6975,00	22186,00	3006,30
12	2085,00	2035,00	1273,00	5479,00	8047,00	135890,00	8127,00	7086,00	20149,00	2982,90
13	2085,00	2034,00	1296,00	5578,00	8086,00	129470,00	8064,00	6927,00	20968,00	2988,10
14	2085,00	2026,00	1273,00	5444,00	7986,00	137210,00	8258,00	6871,00	21010,00	3009,50
15	2085,00	2020,00	1284,00	5321,00	8048,00	132850,00	8127,00	7040,00	20073,00	2977,90
16	2085,00	2020,00	1294,00	5493,00	8043,00	135920,00	8425,00	6994,00	21454,17	3007,90
17	2085,00	2028,00	1273,00	5563,00	8047,00	133380,00	8237,00	7100,00	20152,65	2997,30
18	2085,00	2034,00	1273,00	5479,00	8166,00	119540,00	8057,00	7122,00	19659,73	3008,40
19	2085,00	2034,00	1285,00	5592,00	8024,00	131660,00	8340,00	7026,00	20200,65	2989,50
20	2085,00	2028,00	1273,00	5384,00	8062,00	117940,00	8195,00	6882,00	19175,45	3013,10
21	2085,00	2034,00	1273,00	5486,00	8002,00	125920,00	8216,00	6799,00	19293,55	2984,10
22	2085,00	2046,00	1284,00	5524,00	8083,00	132850,00	8469,00	6947,00	21993,30	2991,50
23	2085,00	2039,00	1284,00	5355,00	8068,00	136930,00	8173,00	7219,00	22040,74	2975,70
24	2085,00	2033,00	1284,00	5491,00	8088,00	117690,00	8330,00	6859,00	20771,19	2990,40
25	2085,00	2020,00	1273,00	5349,00	8009,00	123900,00	8312,00	6946,00	9053,69	2986,40
26	2085,00	2028,00	1273,00	5481,00	8088,00	131750,00	8283,00	7029,00	19636,18	2998,30
27	2085,00	2033,00	1285,00	5379,00	8074,00	135250,00	8483,00	7033,00	20031,37	2998,90
28	2085,00	2031,00	1296,00	5245,00	8088,00	124640,00	8363,00	7013,00	21591,63	3011,60
29	2085,00	2034,00	1285,00	5385,00	8023,00	136130,00	8200,00	7138,00	18905,78	2976,20
30	2085,00	2036,00	1285,00	5254,00	8006,00	139410,00	9060,00	7111,00	18997,08	2973,00
$\bar{x}$	2085,00	2032,00	1281,40	5442,77	8053,60	129707,33	8540,47	7012,93	20358,97	2991,30
S	0,00	6,61	7,85	88,76	47,98	6967,84	421,17	134,14	993,42	14,54

Tabela A.6: Resultados Obtidos para a Metaheurística GRASP-Learning (Tempo de Processamento)

Nº	INSTÂNCIAS TSPLIB									
Iteração	gr17	bays29	swiss42	gr48	berlin52	pr76	gr120	ch150	si175	a280
1	18,92	70,23	119,80	133,39	150,52	737,62	2394,80	1728,70	8825,20	8582,30
2	18,63	72,49	113,58	121,23	151,35	698,25	2500,50	1782,30	9234,20	7962,10
3	18,50	71,22	109,00	120,50	144,28	721,00	2437,90	1755,40	9002,90	8550,60
4	18,83	70,84	119,77	123,06	144,64	697,19	2451,70	1855,60	9258,20	8895,60
5	18,80	71,11	120,67	127,12	152,09	716,93	2468,40	1813,30	9158,80	8534,50
6	19,22	41,36	75,43	121,78	158,20	733,79	2427,60	1738,60	9121,80	8265,70
7	19,17	40,93	77,31	123,66	164,67	697,81	2440,50	1908,00	9095,10	8716,60
8	17,17	40,75	78,56	123,53	160,93	721,42	2439,10	1772,90	9170,10	8758,20
9	20,70	41,88	78,87	126,50	154,69	701,61	2462,70	1765,50	9263,40	8196,90
10	17,36	41,56	80,05	124,28	158,82	726,33	2469,90	1744,10	9092,70	8147,90
11	16,68	42,09	80,30	126,41	153,80	695,98	2458,20	1742,00	9060,50	8717,80
12	16,37	40,16	77,92	124,76	146,01	698,25	2445,50	1672,20	9051,00	7954,30
13	16,44	40,36	76,17	120,75	154,17	707,60	2485,10	1644,60	8960,50	8688,90
14	16,26	41,95	77,44	126,69	151,94	703,76	2516,90	1793,20	9045,50	8030,50
15	16,00	42,49	77,43	127,33	157,74	714,53	2498,90	1813,10	9181,30	8229,60
16	16,34	42,16	81,12	131,06	159,11	710,81	2500,30	1809,80	9242,94	8684,60
17	15,87	41,60	77,25	130,18	156,79	725,65	2569,50	1729,50	9195,88	8412,20
18	16,52	40,63	77,68	129,38	155,48	706,64	2464,60	1760,50	9107,55	8364,30
19	16,26	39,68	77,73	128,78	151,90	728,98	2377,90	1786,90	9092,62	8374,50
20	16,11	41,56	75,56	120,71	176,81	726,02	2337,70	1673,80	9095,59	8242,20
21	16,11	40,99	80,48	133,13	167,81	719,69	2411,20	1834,30	9053,21	8433,80
22	15,92	41,02	73,37	121,74	173,68	748,26	2454,90	1576,60	9114,30	8435,20
23	16,04	41,60	86,84	127,73	160,96	754,18	2410,20	1679,40	9115,47	8724,20
24	16,39	40,71	76,10	144,93	160,11	727,77	2471,90	1833,50	9208,23	8702,20
25	16,41	43,26	75,37	133,20	150,90	730,64	2380,30	1634,60	9201,84	8560,20
26	15,90	41,42	75,63	125,91	158,10	715,63	2394,40	1673,90	9155,19	8310,40
27	16,17	41,04	77,20	137,94	152,48	746,18	2442,50	1787,30	9075,23	8718,30
28	16,01	41,11	78,49	126,54	151,79	716,96	2346,20	1797,30	9206,17	8455,90
29	15,88	42,03	82,22	132,21	151,66	708,82	2407,10	1787,20	9121,23	8284,40
30	15,82	41,85	81,98	127,73	159,31	751,05	2448,30	1704,60	9066,44	8338,20
$\bar{x}$	17,03	46,34	84,64	127,41	156,36	719,65	2443,82	1753,29	8830,08	8442,40
S	1,36	11,33	14,87	5,51	7,46	16,84	50,32	72,38	8972,44	250,49

Resultados Experimentais: Algoritmo Genético Tradicional

Tabela A.7: Resultados Obtidos para o Algoritmo Genético Tradicional (Valor da Função Objetivo)

Nº	INSTÂNCIAS TSPLIB									
Iter.	gr17	bays29	swiss42	gr48	berlin52	pr76	gr120	ch150	si175	a280
1	2090,00	2354,00	1739,00	6175,00	9276,00	187210,00	15372,00	18392,00	31039,00	13615,00
2	2085,00	2269,00	1622,00	7178,00	11781,00	200880,00	16288,00	20157,00	30239,00	13740,00
3	2103,00	2205,00	1841,00	6329,00	10060,00	140760,00	16273,00	21294,00	30576,00	13969,00
4	2153,00	2293,00	1444,00	7039,00	9635,00	211850,00	17844,00	19270,00	30464,00	13223,00
5	2158,00	2260,00	1717,00	7114,00	10388,00	199290,00	16939,00	20450,00	30378,00	13573,00
6	2085,00	2387,00	1607,00	7383,00	9635,00	169930,00	18194,00	18593,00	30904,00	13514,00
7	2085,00	2155,00	1436,00	6064,00	9491,00	190280,00	15026,00	20625,00	31070,00	13375,00
8	2090,00	2159,00	1688,00	6002,00	9162,00	191920,00	16463,00	19913,00	31094,00	14221,00
9	2136,00	2238,00	1469,00	6286,00	10171,00	208440,00	17294,00	19495,00	30842,00	14243,00
10	2085,00	2304,00	1637,00	6593,00	9571,00	173850,00	15053,00	19134,00	30497,00	13185,00
11	2085,00	2200,00	1644,00	6999,00	11475,00	182880,00	15702,00	20193,00	30952,00	13609,00
12	2085,00	2199,00	1612,00	6011,00	11534,00	202270,00	16495,00	20614,00	31064,00	13639,00
13	2167,00	2087,00	1442,00	7338,00	10378,00	180040,00	17761,00	19375,00	30970,00	13831,00
14	2085,00	2242,00	1585,00	7613,00	9913,00	194800,00	18477,00	20521,00	31930,00	13040,00
15	2149,00	2470,00	1737,00	6490,00	11272,00	194700,00	15979,00	19901,00	31257,00	13197,00
16	2085,00	2176,00	1704,00	7553,00	10157,00	195110,00	16598,00	19200,00	31427,00	14533,00
17	2085,00	2277,00	1748,00	6353,00	9394,00	207230,00	16045,00	19347,00	30268,00	13946,00
18	2085,00	2179,00	1697,00	7557,00	9283,00	204690,00	17318,00	19383,00	30150,00	14246,00
19	2090,00	2237,00	1616,00	6397,00	10635,00	185670,00	15640,00	20568,00	30307,00	13234,00
20	2085,00	2141,00	1570,00	6558,00	9483,00	176420,00	14943,00	17962,00	31067,00	13295,00
21	2153,00	2328,00	1545,00	8230,00	10178,00	186770,00	17380,00	19692,00	30487,00	13855,00
22	2085,00	2310,00	1504,00	7606,00	10966,00	189190,00	17221,00	19890,00	31608,00	13564,00
23	2085,00	2312,00	1695,00	7155,00	9718,00	210060,00	16246,00	18694,00	30999,00	12822,00
24	2085,00	2154,00	1679,00	6042,00	9487,00	204310,00	15383,00	20767,00	30965,00	13132,00
25	2085,00	2628,00	1691,00	6788,00	10629,00	165520,00	16766,00	19460,00	31225,00	14119,00
26	2085,00	2270,00	1524,00	7042,00	10787,00	181620,00	17615,00	20326,00	30343,00	13284,00
27	2085,00	2528,00	1656,00	6309,00	8894,00	175170,00	16413,00	20132,00	30832,00	13947,00
28	2167,00	2354,00	1423,00	7376,00	9461,00	190630,00	14429,00	19570,00	31931,00	13959,00
29	2163,00	2358,00	1541,00	6441,00	10594,00	196870,00	17132,00	17122,00	30489,00	13195,00
30	2085,00	2513,00	1613,00	7172,00	9461,00	191410,00	16133,00	21124,00	30455,00	14157,00
$\bar{x}$	2104,97	2269,50	1614,20	6839,77	10095,63	189659,00	16480,73	19705,47	30860,97	13642,07
S	31,76	125,02	107,01	594,59	773,94	15271,34	1022,71	935,08	476,65	432,47

Tabela A.8: Resultados Obtidos para o Algoritmo Genético Tradicional (Tempo de Processamento)

Nº	INSTÂNCIAS TSPLIB									
Iteração	gr17	bays29	swiss42	gr48	berlin52	pr76	gr120	ch150	si175	a280
1	48,93	54,24	55,51	55,36	54,16	89,81	131,98	142,03	151,67	204,23
2	48,65	54,38	54,58	56,70	57,39	89,90	132,22	143,18	152,26	206,38
3	49,52	51,37	55,78	54,59	55,08	85,80	132,06	143,11	152,32	206,45
4	47,86	49,65	54,74	56,65	55,10	90,28	132,19	142,65	152,14	205,52
5	48,27	55,07	55,43	56,55	57,18	90,27	131,71	142,22	151,31	205,65
6	49,37	52,74	54,43	56,87	55,57	89,48	132,45	140,90	151,21	205,81
7	47,83	51,70	53,15	54,79	53,08	89,55	131,55	142,70	151,45	206,33
8	51,81	51,41	53,96	54,32	55,21	89,86	132,24	142,94	152,40	205,89
9	52,24	50,46	53,33	56,52	54,90	89,99	131,83	142,71	152,37	205,63
10	49,71	50,36	55,40	56,64	54,48	89,95	131,12	142,19	151,47	205,23
11	47,71	50,21	55,86	56,35	56,95	89,16	131,75	142,64	151,77	205,66
12	48,00	51,63	50,84	54,24	57,24	89,44	132,23	141,63	151,97	206,05
13	48,35	50,24	55,00	56,03	56,93	88,89	132,66	142,05	151,69	206,02
14	47,53	50,20	54,79	56,82	54,13	89,94	132,78	142,47	152,54	205,77
15	48,14	54,50	55,86	55,56	57,45	89,97	132,27	142,87	152,23	206,69
16	48,87	50,13	54,50	56,10	55,88	89,78	132,18	142,54	152,31	206,75
17	48,69	51,47	53,49	54,59	55,81	90,05	131,49	142,68	152,53	206,77
18	48,02	54,49	56,09	56,93	52,76	89,70	132,15	141,73	151,73	205,71
19	49,67	51,08	54,88	54,70	55,92	89,29	132,45	142,10	151,29	206,35
20	48,13	51,92	54,80	56,55	55,06	89,10	131,78	142,36	151,71	206,21
21	48,84	55,25	52,99	56,83	54,07	88,99	132,29	142,91	151,93	205,76
22	49,28	52,96	51,66	56,58	56,49	89,83	132,37	142,51	152,53	205,18
23	48,08	53,55	52,77	56,73	56,40	90,13	132,03	141,96	152,59	205,66
24	47,91	50,40	55,73	56,25	52,72	90,11	132,08	142,15	151,23	205,57
25	48,21	54,64	54,85	56,06	57,34	88,98	132,38	142,49	151,95	205,89
26	48,50	49,82	54,21	56,87	56,76	88,80	132,11	142,60	150,87	206,05
27	49,56	54,23	55,88	53,33	53,46	88,87	132,08	142,95	152,06	206,40
28	48,70	54,75	53,53	56,46	55,84	89,24	130,47	142,84	152,47	205,65
29	47,88	54,61	56,29	56,78	57,05	89,92	132,33	142,39	151,12	206,14
30	47,61	50,67	55,23	56,89	56,19	89,52	132,17	142,71	150,82	206,17
$\overline{x}$	48,73	52,27	54,52	55,99	55,55	89,49	132,05	142,44	151,86	205,92
S	1,10	1,91	1,32	1,01	1,44	0,83	0,46	0,49	0,53	0,52

Resultados Experimentais: Algoritmo Genético-Learning

Tabela A.9: Resultados Obtidos para o Algoritmo Genético-Learning (Valor da Função Objetivo)

Nº	INSTÂNCIAS TSPLIB									
Iter.	gr17	bays29	swiss42	gr48	berlin52	pr76	gr120	ch150	si175	a280
1	2085,00	2314,00	1495,00	5973,00	8647,00	176290,00	11649,00	9549,00	29153,00	3811,00
2	2085,00	2173,00	1500,00	5860,00	8561,00	172460,00	12682,00	9627,00	30628,00	3913,00
3	2085,00	2390,00	1546,00	6025,00	8390,00	172800,00	12408,00	10041,00	27760,00	3878,00
4	2085,00	2221,00	1584,00	5768,00	8454,00	139130,00	13155,00	9547,00	29612,00	4073,00
5	2085,00	2153,00	1662,00	6663,00	8578,00	164580,00	12687,00	10062,00	28887,00	3789,00
6	2085,00	2077,00	1432,00	5952,00	9097,00	144330,00	12738,00	9761,00	29727,00	3773,00
7	2088,00	2152,00	1564,00	5635,00	8825,00	161600,00	13234,00	10075,00	29310,00	3862,00
8	2153,00	2378,00	1648,00	5443,00	8365,00	160810,00	12354,00	9098,00	29335,00	3727,00
9	2085,00	2368,00	1702,00	5900,00	8651,00	164650,00	12725,00	9944,00	28985,00	3686,00
10	2085,00	2137,00	1601,00	5758,00	8393,00	129280,00	9897,00	9720,00	29147,00	3840,00
11	2085,00	2414,00	1617,00	5901,00	8197,00	173020,00	12309,00	9601,00	29843,00	3975,00
12	2085,00	2233,00	1663,00	6235,00	8350,00	178150,00	11689,00	9950,00	28259,00	3811,00
13	2085,00	2307,00	1653,00	5890,00	8387,00	160940,00	12378,00	9459,00	29466,00	3871,00
14	2085,00	2315,00	1552,00	5828,00	9861,00	172250,00	12534,00	9976,00	29941,00	3957,00
15	2085,00	2264,00	1658,00	5799,00	9030,00	189540,00	11474,00	9320,00	28819,00	4055,00
16	2085,00	2292,00	1616,00	5902,00	8401,00	170770,00	11819,00	9056,00	29353,00	3852,00
17	2085,00	2388,00	1583,00	6412,00	8750,00	148650,00	12255,00	9212,00	29037,00	3714,00
18	2085,00	2221,00	1635,00	5609,00	9506,00	134850,00	11582,00	9135,00	29516,00	3872,00
19	2085,00	2406,00	1351,00	5764,00	8705,00	181760,00	12345,00	9716,00	29549,00	3822,00
20	2085,00	2191,00	1520,00	5647,00	8285,00	163980,00	11641,00	8521,00	29163,00	4032,00
21	2085,00	2205,00	1451,00	5916,00	9269,00	176560,00	12181,00	8946,00	29206,00	3705,00
22	2085,00	2208,00	1354,00	6469,00	9694,00	158100,00	12563,00	9642,00	28632,00	3765,00
23	2085,00	2289,00	1415,00	5807,00	9032,00	174440,00	12703,00	10401,00	29096,00	3662,00
24	2085,00	2208,00	1693,00	6168,00	8986,00	192680,00	12392,00	10375,00	29425,00	4015,00
25	2085,00	2180,00	1582,00	6679,00	9096,00	181590,00	11842,00	8971,00	28835,00	3999,00
26	2085,00	2065,00	1464,00	6396,00	9918,00	163790,00	12882,00	10618,00	30086,00	4008,00
27	2085,00	2108,00	1439,00	6129,00	9416,00	157570,00	11219,00	9191,00	29719,00	3732,00
28	2085,00	2339,00	1500,00	5963,00	8365,00	155270,00	12299,00	9909,00	28437,00	3674,00
29	2085,00	2208,00	1548,00	5734,00	9124,00	166430,00	11499,00	8948,00	29494,00	3869,00
30	2085,00	2360,00	1368,00	5812,00	8325,00	172180,00	13041,00	9999,00	29504,00	3838,00
$\bar{x}$	2087,37	2252,13	1546,53	5967,90	8821,93	165281,67	12205,87	9612,33	29264,13	3852,67
S	12,41	100,83	102,43	304,67	493,52	15040,40	686,88	498,24	565,87	119,63

Tabela A.10: Resultados Obtidos para o Algoritmo Genético-Learning (Tempo de Processamento)

Nº	INSTÂNCIAS TSPLIB									
Iteração	gr17	bays29	swiss42	gr48	berlin52	pr76	gr120	ch150	si175	a280
1	48,34	52,34	55,94	56,32	58,13	96,07	140,77	156,28	193,40	276,88
2	49,20	50,38	53,39	56,79	57,67	95,91	141,35	154,88	194,20	277,01
3	48,31	49,65	53,52	57,64	55,43	95,31	141,29	151,77	190,92	279,69
4	48,20	49,87	53,79	53,70	53,74	92,33	141,35	151,93	192,81	282,25
5	48,38	52,27	56,86	57,46	51,97	96,23	141,54	156,83	193,30	266,02
6	48,13	52,03	53,32	57,34	56,84	93,96	141,51	152,25	193,78	273,36
7	49,53	50,99	57,18	55,76	53,11	96,00	141,41	155,66	193,73	261,49
8	47,70	50,94	57,10	55,20	53,98	95,73	140,82	153,80	193,07	247,42
9	47,56	52,15	57,02	57,94	53,28	96,22	141,37	154,20	193,95	276,69
10	47,44	49,94	55,61	51,98	53,48	90,33	134,23	155,16	192,81	275,57
11	47,60	55,28	56,90	53,21	54,08	96,49	139,30	155,04	193,74	276,44
12	47,95	51,63	55,29	57,71	56,45	96,32	136,98	154,55	192,48	271,41
13	49,53	54,82	57,07	54,81	58,70	95,02	140,80	155,28	193,85	268,91
14	48,09	54,88	51,79	52,23	58,95	95,33	140,15	156,80	193,77	282,55
15	47,51	50,53	56,33	55,93	52,74	96,25	139,89	150,77	192,89	280,93
16	47,59	49,79	55,37	56,74	56,16	96,19	141,17	144,11	192,81	279,91
17	47,52	52,14	57,10	57,87	53,73	94,72	141,21	142,46	191,98	271,23
18	47,53	51,49	56,92	57,91	58,52	91,67	139,53	154,33	194,36	278,60
19	48,13	55,17	53,12	57,27	52,49	96,03	141,41	154,17	193,56	266,82
20	48,16	53,61	56,91	55,69	52,81	95,45	141,05	152,80	193,81	284,74
21	47,60	53,09	52,31	52,79	54,89	95,41	140,90	145,86	192,29	276,44
22	47,60	52,00	51,86	57,11	58,89	95,49	140,35	153,51	192,62	283,25
23	47,46	52,39	55,54	57,39	58,22	96,13	141,38	155,67	193,34	269,98
24	48,19	50,15	56,75	57,29	58,64	96,20	140,92	157,01	194,02	286,06
25	47,69	53,36	56,52	57,81	52,92	96,76	141,31	148,93	193,34	282,68
26	48,45	52,61	51,23	57,64	58,97	96,05	141,27	156,03	194,65	285,48
27	47,93	54,30	54,06	55,62	58,26	95,69	138,83	153,33	193,09	259,76
28	48,13	53,15	51,70	57,55	54,07	95,70	140,67	156,34	192,66	265,69
29	47,73	55,07	56,50	56,08	56,17	95,50	138,85	151,47	193,54	273,50
30	47,73	51,29	53,17	56,82	57,12	96,18	140,79	156,59	192,76	275,67
$\overline{x}$	48,03	52,24	55,01	56,19	55,68	95,36	140,41	153,26	193,25	274,55
S	0,57	1,74	2,01	1,78	2,40	1,47	1,56	3,69	0,77	8,65

Resultados Experimentais: Algoritmo Genético Learning Cooperativo

Tabela A.11: Resultados Obtidos para o Algoritmo Genético-Learning Cooperativo (Valor da Função Objetivo)

Nº	INSTÂNCIAS TSPLIB									
Iter.	gr17	bays29	swiss42	gr48	berlin52	pr76	gr120	ch150	si175	a280
1	2085,00	2227,00	1515,00	6005,00	8896,00	137730,00	9089,00	8994,00	25084,00	3672,00
2	2085,00	2156,00	1434,00	5651,00	8611,00	128630,00	8724,00	8705,00	24735,00	3910,00
3	2085,00	2190,00	1467,00	5788,00	8730,00	133910,00	8868,00	8845,00	24896,00	3846,00
4	2085,00	2177,00	1465,00	5764,00	8713,00	133600,00	8818,00	8829,00	24854,00	3987,00
5	2085,00	2211,00	1490,00	5965,00	8852,00	136480,00	8955,00	8949,00	24979,00	3857,00
6	2085,00	2205,00	1475,00	5893,00	8794,00	135890,00	8898,00	8878,00	24899,00	3787,00
7	2085,00	2174,00	1462,00	5763,00	8697,00	133270,00	8816,00	8817,00	24842,00	3971,00
8	2085,00	2053,00	1394,00	5379,00	8311,00	123220,00	8419,00	8512,00	24449,00	3808,00
9	2085,00	2210,00	1489,00	5930,00	8852,00	136460,00	8952,00	8947,00	24948,00	3956,00
10	2085,00	2173,00	1459,00	5710,00	8696,00	132990,00	8801,00	8815,00	24816,00	3960,00
11	2085,00	2192,00	1475,00	5828,00	8773,00	134210,00	8870,00	8852,00	24898,00	3968,00
12	2085,00	2208,00	1487,00	5908,00	8799,00	136130,00	8902,00	8911,00	24930,00	3672,00
13	2085,00	2222,00	1500,00	5980,00	8878,00	137610,00	9054,00	8980,00	25015,00	3678,00
14	2085,00	2194,00	1475,00	5854,00	8778,00	135050,00	8898,00	8852,00	24898,00	3765,00
15	2085,00	2093,00	1423,00	5516,00	8493,00	126270,00	8484,00	8636,00	24653,00	3893,00
16	2085,00	2166,00	1451,00	5700,00	8671,00	131820,00	8776,00	8802,00	24803,00	3918,00
17	2085,00	2226,00	1505,00	5984,00	8890,00	137670,00	9078,00	8990,00	25059,00	3846,00
18	2085,00	2219,00	1495,00	5979,00	8863,00	137090,00	9052,00	8967,00	24991,00	3735,00
19	2085,00	2172,00	1458,00	5706,00	8684,00	132970,00	8794,00	8815,00	24806,00	3790,00
20	2085,00	2211,00	1491,00	5974,00	8861,00	136810,00	8984,00	8965,00	24990,00	3679,00
21	2085,00	2078,00	1414,00	5401,00	8366,00	124590,00	8442,00	8610,00	24564,00	3855,00
22	2085,00	2163,00	1451,00	5682,00	8666,00	130570,00	8773,00	8797,00	24791,00	3568,00
23	2085,00	2209,00	1489,00	5922,00	8826,00	136290,00	8907,00	8935,00	24945,00	3568,00
24	2085,00	2034,00	1339,00	5308,00	8265,00	122600,00	8404,00	8462,00	24366,00	3666,00
25	2085,00	2080,00	1419,00	5475,00	8418,00	124620,00	8462,00	8630,00	24609,00	3866,00
26	2085,00	2148,00	1434,00	5651,00	8608,00	128180,00	8682,00	8681,00	24722,00	3609,00
27	2085,00	2122,00	1429,00	5545,00	8498,00	126630,00	8524,00	8660,00	24669,00	3893,00
28	2085,00	2188,00	1467,00	5769,00	8719,00	133730,00	8834,00	8832,00	24874,00	3782,00
29	2085,00	2159,00	1447,00	5672,00	8622,00	130120,00	8768,00	8761,00	24775,00	3679,00
30	2085,00	2134,00	1433,00	5645,00	8553,00	127870,00	8582,00	8672,00	24681,00	3857,00
$\bar{x}$	2085,00	2175,50	1457,73	5744,90	8679,43	132100,33	8787,00	8803,37	24818,03	3801,36
S	0,00	125,02	37,29	196,68	176,28	4726,23	204,02	143,98	172,71	249,08

Tabela A.12: Resultados Obtidos para o Algoritmo Genético-Learning Cooperativo (Tempo de Processamento)

Nº	INSTÂNCIAS TSPLIB									
Iteração	gr17	bays29	swiss42	gr48	berlin52	pr76	gr120	ch150	si175	a280
1	52,44	58,15	64,35	68,25	70,19	124,33	212,55	248,96	287,72	545,59
2	51,08	56,27	62,64	65,82	68,34	123,25	211,08	247,20	285,14	542,70
3	52,21	58,03	64,23	66,97	69,62	124,12	212,13	247,90	286,28	544,32
4	52,01	58,01	64,21	66,94	69,36	123,55	211,95	247,83	285,93	544,11
5	52,36	58,10	64,31	67,07	70,14	124,25	212,39	248,32	287,00	545,08
6	52,25	58,05	64,26	66,99	69,95	124,18	212,19	248,12	286,73	544,63
7	51,99	58,00	64,21	66,63	69,14	123,52	211,92	247,72	285,87	543,90
8	50,68	56,18	62,23	65,51	67,78	121,93	209,69	245,27	283,55	538,85
9	52,35	58,08	64,28	67,07	70,07	124,24	212,31	248,27	286,97	544,97
10	51,95	57,99	64,21	66,63	69,09	123,51	211,78	247,63	285,82	543,82
11	52,22	58,03	64,24	66,99	69,87	124,16	212,15	248,04	286,56	544,36
12	52,26	58,07	64,26	66,99	70,05	124,19	212,29	248,15	286,87	544,67
13	52,39	58,12	64,33	67,67	70,18	124,31	212,45	248,73	287,52	545,51
14	52,25	58,05	64,25	66,99	69,90	124,17	212,17	248,11	286,65	544,59
15	50,93	56,22	62,55	65,64	67,97	123,05	210,16	246,00	284,53	540,67
16	51,92	57,96	64,18	66,56	68,93	123,40	211,69	247,46	285,75	543,37
17	52,41	58,15	64,35	67,76	70,19	124,31	212,48	248,75	287,61	545,57
18	52,37	58,12	64,33	67,18	70,16	124,29	212,40	248,64	287,52	545,51
19	51,95	57,97	64,19	66,58	69,00	123,47	211,70	247,57	285,78	543,50
20	52,37	58,10	64,31	67,14	70,15	124,27	212,40	248,46	287,34	545,20
21	50,71	56,21	62,24	65,56	67,84	122,64	209,79	245,59	284,29	539,77
22	51,91	57,95	64,01	66,48	68,70	123,36	211,48	247,29	285,26	543,20
23	52,31	58,07	64,28	67,03	70,06	124,20	212,31	248,19	286,96	544,89
24	50,64	56,18	62,18	65,49	67,69	121,90	209,19	245,11	281,80	536,74
25	50,86	56,21	62,28	65,59	67,89	122,86	209,89	245,84	284,53	539,78
26	51,03	56,26	62,60	65,77	68,29	123,19	210,56	246,94	285,04	541,04
27	51,00	56,24	62,59	65,70	68,25	123,05	210,42	246,10	284,63	540,69
28	52,10	58,02	64,22	66,97	69,56	123,93	211,97	247,83	286,26	544,16
29	51,09	56,43	62,77	65,83	68,35	123,26	211,42	247,25	285,19	543,09
30	51,02	56,25	62,60	65,76	68,29	123,16	210,51	246,80	284,83	540,77
$\bar{x}$	51,77	57,45	63,66	66,59	69,17	123,60	211,51	247,47	285,86	543,17
S	0,65	0,87	0,87	0,75	0,90	0,69	1,00	1,07	1,36	2,31

Resultados Experimentais: Análise de Sobrevivência

Tabela A.13: Dados da Análise de Sobrevivência para os Algoritmos GRASP, GRASP Reativo e GRASP-Learning (Instância gr17) - Tempo de Censura: 4,13 s

GRASP				GRASP Reativo				GRASP-Learning			
t(s)	Frequência	S(t)	1-s(t)	t(s)	Frequência	S(t)	1-s(t)	t(s)	Frequência	S(t)	1-s(t)
0,16	1,00	0,97	0,03	0,08	1,00	0,97	0,03	0,22	2,00	0,93	0,07
0,30	1,00	0,93	0,07	0,09	1,00	0,93	0,07	0,23	1,00	0,90	0,10
0,56	1,00	0,90	0,10	0,14	1,00	0,90	0,10	0,24	2,00	0,83	0,17
0,63	1,00	0,87	0,13	0,25	1,00	0,87	0,13	0,25	1,00	0,80	0,20
0,88	1,00	0,83	0,17	0,28	1,00	0,83	0,17	0,45	1,00	0,77	0,23
0,92	1,00	0,80	0,20	0,28	2,00	0,77	0,23	0,47	1,00	0,73	0,27
0,94	1,00	0,77	0,23	0,34	1,00	0,73	0,27	0,66	2,00	0,67	0,33
1,31	1,00	0,73	0,27	0,34	1,00	0,70	0,30	0,66	1,00	0,63	0,37
1,34	1,00	0,70	0,30	0,49	1,00	0,67	0,33	0,69	2,00	0,57	0,43
1,49	1,00	0,67	0,33	0,50	1,00	0,63	0,37	0,75	1,00	0,53	0,47
1,58	1,00	0,63	0,37	0,59	1,00	0,60	0,40	0,89	2,00	0,47	0,53
1,70	1,00	0,60	0,40	0,64	1,00	0,57	0,43	0,91	2,00	0,40	0,60
1,92	1,00	0,57	0,43	0,69	1,00	0,53	0,47	0,94	1,00	0,37	0,63
2,33	1,00	0,53	0,47	0,72	1,00	0,50	0,50	1,16	1,00	0,33	0,67
2,36	1,00	0,50	0,50	0,78	1,00	0,47	0,53	1,33	1,00	0,30	0,70
2,70	1,00	0,47	0,53	0,80	1,00	0,43	0,57	1,36	1,00	0,27	0,73
3,23	1,00	0,43	0,57	0,92	1,00	0,40	0,60	1,58	1,00	0,23	0,77
3,36	1,00	0,40	0,60	0,94	1,00	0,37	0,63	1,59	1,00	0,20	0,80
3,53	1,00	0,37	0,63	1,08	1,00	0,33	0,67	1,61	1,00	0,17	0,83
3,92	1,00	0,33	0,67	1,31	1,00	0,30	0,70	1,63	1,00	0,13	0,87
4,05	1,00	0,30	0,70	1,38	1,00	0,27	0,73	2,19	1,00	0,10	0,90
4,73	1,00	0,27	0,73	1,55	1,00	0,23	0,77	2,50	1,00	0,07	0,93
-	-	-	-	1,72	1,00	0,20	0,80	3,92	1,00	0,03	0,97
-	-	-	-	1,73	1,00	0,17	0,83	4,31	1,00	0,00	1,00
-	-	-	-	1,88	1,00	0,10	0,90	-	-	-	-
-	-	-	-	1,91	1,00	0,07	0,93	-	-	-	-
-	-	-	-	2,34	1,00	0,03	0,97	-	-	-	-
-	-	-	-	4,83	1,00	0,00	1,00	-	-	-	-

Tabela A.14: Dados da Análise de Sobrevivência para os Algoritmos GRASP, GRASP Reativo e GRASP-Learning (Instância pr76) - Tempo de Censura: 91,38 s

GRASP				GRASP Reativo				GRASP-Learning			
t(s)	Frequência	S(t)	1-s(t)	t(s)	Frequência	S(t)	1-s(t)	t(s)	Frequência	S(t)	1-s(t)
15,66	1,00	0,97	0,03	5,56	1,00	0,97	0,03	7,03	1,00	0,97	0,03
16,39	1,00	0,93	0,07	5,66	1,00	0,93	0,07	7,13	1,00	0,93	0,07
16,61	1,00	0,90	0,10	6,08	1,00	0,90	0,10	7,22	1,00	0,90	0,10
18,41	1,00	0,87	0,13	6,11	1,00	0,87	0,13	7,42	1,00	0,87	0,13
27,19	1,00	0,83	0,17	6,17	1,00	0,83	0,17	7,56	1,00	0,83	0,17
30,99	1,00	0,80	0,20	7,53	1,00	0,80	0,20	7,63	1,00	0,80	0,20
35,08	1,00	0,77	0,23	7,83	1,00	0,77	0,23	7,83	1,00	0,77	0,23
47,61	1,00	0,73	0,27	9,98	1,00	0,73	0,27	8,02	1,00	0,73	0,27
47,95	1,00	0,70	0,30	10,86	1,00	0,70	0,30	8,25	1,00	0,70	0,30
49,98	1,00	0,67	0,33	11,63	1,00	0,67	0,33	8,28	1,00	0,67	0,33
50,61	1,00	0,63	0,37	17,44	1,00	0,63	0,37	8,45	1,00	0,63	0,37
50,70	1,00	0,60	0,40	18,03	1,00	0,60	0,40	8,48	1,00	0,60	0,40
53,66	1,00	0,57	0,43	22,45	1,00	0,57	0,43	8,49	1,00	0,57	0,43
54,98	1,00	0,53	0,47	24,39	1,00	0,53	0,47	8,53	1,00	0,53	0,47
62,36	1,00	0,50	0,50	27,03	1,00	0,50	0,50	8,55	1,00	0,50	0,50
64,17	1,00	0,47	0,53	30,86	1,00	0,47	0,53	8,61	1,00	0,47	0,53
64,89	1,00	0,43	0,57	32,88	1,00	0,43	0,57	8,64	1,00	0,43	0,57
65,23	1,00	0,40	0,60	36,28	1,00	0,40	0,60	8,77	1,00	0,40	0,60
65,34	1,00	0,37	0,63	37,74	1,00	0,37	0,63	8,88	1,00	0,37	0,63
70,66	1,00	0,33	0,67	38,44	1,00	0,33	0,67	8,92	1,00	0,33	0,67
80,47	1,00	0,30	0,70	40,89	1,00	0,30	0,70	9,27	1,00	0,30	0,70
84,33	1,00	0,27	0,73	41,83	1,00	0,27	0,73	9,42	1,00	0,27	0,73
122,61	1,00	0,23	0,77	45,56	1,00	0,23	0,77	9,48	1,00	0,23	0,77
-	-	-	-	46,94	1,00	0,20	0,80	9,56	1,00	0,20	0,80
-	-	-	-	50,14	1,00	0,17	0,83	10,45	1,00	0,17	0,83
-	-	-	-	54,22	1,00	0,13	0,87	14,39	1,00	0,13	0,87
-	-	-	-	55,34	1,00	0,10	0,90	15,72	1,00	0,10	0,90
-	-	-	-	58,13	1,00	0,07	0,93	16,09	1,00	0,07	0,93
-	-	-	-	61,91	1,00	0,03	0,97	17,16	1,00	0,03	0,97
-	-	-	-	141,70	1,00	0,00	1,00	17,83	1,00	0,00	1,00

Tabela A.15: Dados da Análise de Sobrevivência para os Algoritmos Genético Tradicional, Genético-Learning e Genético-Learning Cooperativo (Instância gr17) - Tempo de Censura: 51,10 s

Genético Tradicional				Genético-Learning				Genético-Learning Cooperativo			
t(s)	Frequência	S(t)	1-s(t)	t(s)	Frequência	S(t)	1-s(t)	t(s)	Frequência	S(t)	1-s(t)
10,77	1,00	0,97	0,03	3,69	1,00	0,97	0,03	1,96	1,00	0,97	0,03
34,24	1,00	0,93	0,07	3,82	1,00	0,93	0,07	1,98	1,00	0,93	0,07
37,67	1,00	0,90	0,10	3,89	1,00	0,90	0,10	2,16	1,00	0,90	0,10
39,71	1,00	0,87	0,13	3,95	1,00	0,87	0,13	2,31	1,00	0,87	0,13
40,42	1,00	0,83	0,17	4,00	1,00	0,83	0,17	2,56	1,00	0,83	0,17
44,88	1,00	0,80	0,20	4,26	1,00	0,80	0,20	2,65	1,00	0,80	0,20
47,54	1,00	0,77	0,23	4,43	1,00	0,77	0,23	2,65	1,00	0,77	0,23
47,82	1,00	0,73	0,27	4,65	1,00	0,73	0,27	2,67	1,00	0,73	0,27
47,83	1,00	0,70	0,30	4,68	1,00	0,70	0,30	2,83	1,00	0,70	0,30
48,04	1,00	0,67	0,33	4,91	1,00	0,67	0,33	3,05	1,00	0,67	0,33
48,32	1,00	0,63	0,37	5,28	1,00	0,63	0,37	3,24	1,00	0,63	0,37
48,39	1,00	0,60	0,40	5,32	1,00	0,60	0,40	3,56	1,00	0,60	0,40
48,44	1,00	0,57	0,43	6,11	1,00	0,57	0,43	3,87	1,00	0,57	0,43
48,57	1,00	0,53	0,47	6,94	1,00	0,53	0,47	4,32	1,00	0,53	0,47
48,75	1,00	0,50	0,50	8,99	1,00	0,50	0,50	5,12	1,00	0,50	0,50
48,76	1,00	0,47	0,53	9,84	1,00	0,47	0,53	5,14	1,00	0,47	0,53
49,00	1,00	0,43	0,57	10,06	1,00	0,43	0,57	5,97	1,00	0,43	0,57
49,22	1,00	0,40	0,60	10,64	1,00	0,40	0,60	5,99	1,00	0,40	0,60
49,49	1,00	0,37	0,63	10,79	1,00	0,37	0,63	6,39	1,00	0,37	0,63
50,12	1,00	0,33	0,67	14,70	1,00	0,33	0,67	6,45	1,00	0,33	0,67
52,56	1,00	0,30	0,70	23,10	1,00	0,30	0,70	6,92	1,00	0,30	0,70
-	-	-	-	38,60	1,00	0,27	0,73	6,99	1,00	0,27	0,73
-	-	-	-	39,18	1,00	0,23	0,77	7,06	1,00	0,23	0,77
-	-	-	-	40,07	1,00	0,20	0,80	7,14	1,00	0,20	0,80
-	-	-	-	40,09	1,00	0,17	0,83	7,40	1,00	0,17	0,83
-	-	-	-	45,45	1,00	0,13	0,87	9,14	1,00	0,13	0,87
-	-	-	-	48,64	1,00	0,10	0,90	9,25	1,00	0,10	0,90
-	-	-	-	48,90	1,00	0,07	0,93	9,65	1,00	0,07	0,93
-	-	-	-	48,96	1,00	0,03	0,97	12,74	1,00	0,03	0,97
-	-	-	-	48,98	1,00	0,00	1,00	28,20	1,00	0,00	1,00

Tabela A.16: Dados da Análise de Sobrevivência para os Algoritmos Genético Tradicional, Genético-Learning e Genético-Learning Cooperativo (Instância pr76) - Tempo de Censura: 87,61 s

Genético Tradicional				Genético-Learning				Genético-Learning Cooperativo			
t(s)	Frequência	S(t)	1-s(t)	t(s)	Frequência	S(t)	1-s(t)	t(s)	Frequência	S(t)	1-s(t)
84,35	1,00	0,97	0,03	26,03	1,00	0,97	0,03	18,48	1,00	0,97	0,03
85,10	1,00	0,93	0,07	31,06	1,00	0,93	0,07	21,33	1,00	0,93	0,07
85,86	1,00	0,90	0,10	56,36	1,00	0,90	0,10	23,64	1,00	0,90	0,10
85,96	1,00	0,87	0,13	59,98	1,00	0,87	0,13	25,01	1,00	0,87	0,13
86,30	1,00	0,83	0,17	63,38	1,00	0,83	0,17	26,81	1,00	0,83	0,17
86,52	1,00	0,80	0,20	66,68	1,00	0,80	0,20	28,03	1,00	0,80	0,20
86,61	1,00	0,77	0,23	68,63	1,00	0,77	0,23	28,37	1,00	0,77	0,23
86,91	1,00	0,73	0,27	71,07	1,00	0,73	0,27	30,15	1,00	0,73	0,27
87,25	1,00	0,70	0,30	72,11	1,00	0,70	0,30	30,40	1,00	0,70	0,30
87,33	1,00	0,67	0,33	72,54	1,00	0,67	0,33	31,19	1,00	0,67	0,33
87,71	1,00	0,63	0,37	72,69	1,00	0,63	0,37	32,59	1,00	0,63	0,37
-	-	-	-	73,18	1,00	0,60	0,40	32,81	1,00	0,60	0,40
-	-	-	-	73,29	1,00	0,57	0,43	34,98	1,00	0,57	0,43
-	-	-	-	75,06	1,00	0,53	0,47	36,80	1,00	0,53	0,47
-	-	-	-	75,81	1,00	0,50	0,50	42,03	1,00	0,50	0,50
-	-	-	-	77,61	1,00	0,47	0,53	43,67	1,00	0,47	0,53
-	-	-	-	79,25	1,00	0,43	0,57	45,20	1,00	0,43	0,57
-	-	-	-	85,96	1,00	0,40	0,60	46,06	1,00	0,40	0,60
-	-	-	-	87,80	1,00	0,37	0,63	47,00	1,00	0,37	0,63
-	-	-	-	-	-	-	-	59,15	1,00	0,33	0,67
-	-	-	-	-	-	-	-	59,69	1,00	0,30	0,70
-	-	-	-	-	-	-	-	64,65	1,00	0,27	0,73
-	-	-	-	-	-	-	-	67,12	1,00	0,23	0,77
-	-	-	-	-	-	-	-	82,70	1,00	0,20	0,80
-	-	-	-	-	-	-	-	84,44	1,00	0,17	0,83
-	-	-	-	-	-	-	-	90,60	1,00	0,13	0,87

Apêndice B

Publicações

Listagem das Publicações Relacionadas ao Trabalho

Publicações: Artigos em Congressos Nacionais

LIMA JÚNIOR, F. C.; DÓRIA NETO, Adrião Duarte; MELO, Jorge Dantas de.

1. Uso da Metaheurística GRASP com Aprendizagem por Reforço na Resolução do Problema do Caixeiro Viajante. In: XVI Congresso Brasileiro de Automática, 2006, Salvador-BA. Anais CBA/CLCA, 2006.

Abstract

Techniques of optimization known as metaheuristics have gotten success in the resolution of innumerable classified problems as NP-Hard. These methods use non deterministic approach that finds very good solutions without, however, guarantee the determination of the global optimum. Beyond the inherent difficulties to complexity that characterizes the optimization problems, the metaheuristics still face the dilemma of the exploitation - exploration, which consists of choosing between a greedy search and a wider exploration of the solution space. A way to guide such algorithms in search of better solutions is to supply them with more knowledge through learning of the environment. This way, this work proposes the use of a technique of Reinforcement Learning - Q-learning Algorithm - as algorithm for the constructive phase of GRASP metaheuristic applied to the symmetrical traveling salesman problem.

Keywords: Reinforcement Learning, GRASP, Combinatorial Optimization, Q-learning Algorithm, Traveling Salesman Problem.

Publicações: Artigos em Congressos Internacionais

SANTOS, João Queiroz dos; LIMA JÚNIOR, F. C.; MAGALHÃES, Rafael Marrocos; DÓRIA NETO, Adrião Duarte; MELO, Jorge Dantas de.

1. A Parallel Hybrid Implementation Using Genetic Algorithm, GRASP and Reinforcement Learning. In: 2009 International Joint Conference on Neural Networks (IJCNN), Atlanta - USA. Acontecerá de 14 a 19 de Junho de 2009.

Abstract

In the process of searching for better solutions, a metaheuristic can be guided to regions of promising solutions using the acquisition of information on the problem under study. In this work this is done through the use of reinforcement learning. The performance of a metaheuristic can also be improved using multiple search trajectories, which act competitively and/or cooperatively. This can be accomplished using parallel processing. Thus, in this paper we propose a hybrid parallel implementation for the GRASP metaheuristics and the genetic algorithm, using reinforcement learning, applied to the symmetric traveling salesman problem.

Keywords: Parallel Processing, GRASP Metaheuristics, Genetic Algorithm, Q-learning Algorithm.

LIMA JÚNIOR, F. C.; DÓRIA NETO, Adrião Duarte; MELO, Jorge Dantas de.

2. Using the Q-learning Algorithm in the Constructive Phase of the GRASP and Reactive GRASP Metaheuristics. In: 2008 International Joint Conference on Neural Networks (IJCNN), 2008, Hong kong. IJCNN2008 Proceedings, 2008.

Abstract

Currently many non-tractable considered problems have been solved satisfactorily through methods of approximate optimization called metaheuristic. These methods use non-deterministic approaches that find good solutions which, however, do not guarantee the determination of the global optimum. The success of a metaheuristic is conditioned by capacity to adequately alternate between exploration and exploitation of the solution space. A way to guide such algorithms while searching for better solutions is supplying them with more knowledge of the solution space (environment of the problem). This can to be made in terms of a mapping of such environment in states and actions using Reinforcement Learning. This paper proposes the use of a technique of Reinforcement Learning - Q-Learning Algorithm - for the constructive phase of GRASP and Reactive GRASP metaheuristic. The proposed methods will be applied to the symmetrical traveling salesman problem.

Keywords: Reactive GRASP, Q-learning Algorithm, Traveling Salesman Problem

LIMA JÚNIOR, F. C.; DÓRIA NETO, Adrião Duarte; MELO, Jorge Dantas de.

3. Using Q-learning Algorithm for Initialization of the GRASP Metaheuristic and Genetic Algorithm. In: 2007 International Joint Conference on Neural Networks (IJCNN), 2007, Orlando. IJCNN2007 Proceedings, 2007.

Abstract

Techniques of optimization, known as metaheuristics, have achieved success in the resolution of many problems classified as NP-Hard. These methods use non-deterministic approaches that find good solutions which, however, do not guarantee the determination of the global optimum. Beyond the inherent difficulties related to the complexity that characterizes the optimization problems, the metaheuristics still face the dilemma of the exploitation - exploration, which consists of choosing between a greedy search and a wider exploration of the solution space. A way to guide such algorithms during the search of better solutions is supplying them with more knowledge through the learning of the environment. This way, this work proposes the use of a technique of Reinforcement Learning - Q-Learning Algorithm - for the constructive phase of GRASP metaheuristic and to generate the initial population of a Genetic Algorithm. The proposed methods will be applied to the symmetrical traveling salesman problem.

Keywords: GRASP Metaheuristic, Genetic Algorithm, Reinforcement Learning.

LIMA JÚNIOR, F. C.; DÓRIA NETO, Adrião Duarte; MELO, Jorge Dantas de.

4. Proposal for Improvement of GRASP Metaheuristic and Genetic Algorithm Using the Q-Learning Algorithm.. In: International Conference on Intelligent Systems Design and Applications (ISDA), 2007, Rio de Janeiro. Proceedings of ISDA'07, published IEEE Computer Society, 2007.

Abstract

Currently many non-tractable considered problems have been solved satisfactorily through methods of approximate optimization called metaheuristic. These methods use non-deterministic approaches that find good solutions which, however, do not guarantee the determination of the global optimum. The success of a metaheuristic is conditioned its capacity to adequately alternate between exploration and exploitation of the solutions space. A way to guide such algorithms during the searching for better solutions is supplying them with more knowledge of the environment. This work proposes the use of a technique of Reinforcement Learning - Q-Learning Algorithm - for the constructive phase of GRASP metaheuristic and also as generator of the initial population for the Genetic Algorithm. The proposed methods will be applied to the symmetrical traveling salesman problem.

Keywords: Traveling Salesman Problem, Genetic Algorithm, GRASP Metaheuristic, Q-learning Algorithm.

Publicações: Artigos em Revista ou Capítulo de Livro

LIMA JÚNIOR, F. C.; DÓRIA NETO, Adrião Duarte; MELO, Jorge Dantas de.

1. Using Reinforcement Learning to Improve the GRASP Metaheuristic and Genetic Algorithm. Elsevier - NEUROCOMPUTING. (Em fase de correção)

Abstract

Optimization techniques known as metaheuristics have been successful in resolving many problems classified as NP-Complete. These methods use non deterministic approaches that provide excellent solutions; however, they do not guarantee the determination of the global optimum. In addition to the inherent difficulties related to the complexity that characterizes optimization problems, metaheuristics still faces the exploration/exploitation dilemma, which consists of choosing between a greedy search and a wider exploration of the solution space. One way to guide such algorithms during the search for better solutions is to supply Using Reinforcement Learning to Improve the GRASP Metaheuristic and Genetic Algorithm them with more knowledge through the learning of the environment. Therefore, this work proposes to use the Reinforcement Learning Technique - Q-Learning Algorithm - as an exploration/exploitation strategy for the metaheuristics GRASP (*Greedy Randomized Adaptive Search Procedure*) and Genetic Algorithm. The metaheuristic GRASP is used *Q-Learning* as a substitute for the traditional greedy-random algorithm in the construction phase. In the Genetic Algorithm *Q-Learning* was used to generate an initial high fitness population, and after a determinate number of generations, it was also used to supply one of the parents to be used in the genetic crossover operator. Both the algorithms were applied successfully to the symmetrical Traveling Salesman Problem, which was modeled as a non-stationary Markov decision process.

Keywords: GRASP Metaheuristic, Genetic Algorithm, Reinforcement Learning, Q-Learning Algorithm

SANTOS, João Queiroz dos; LIMA JÚNIOR, F. C.; MAGALHÃES, Rafael Marrocos; DÓRIA NETO, Adrião Duarte; MELO, Jorge Dantas de.

2. A Parallel Hybrid Implementation Using Genetic Algorithms, GRASP and Reinforcement Learning for the Salesman Traveling Problem. Capítulo de Livro. In Computational Intelligence in Expensive Optimization Problems. (Em fase de Revisão)

Abstract

Many problems formerly considered intractable have been satisfactorily resolved using approximate optimization methods called metaheuristics. These methods use a non-deterministic approach that finds good solutions, despite not ensuring the determination of the overall optimum. The success of a metaheuristic is conditioned on its capacity of alternating properly between the exploration and exploitation of solution spaces. During the process of searching for better solutions, a metaheuristic can be guided to regions of promising solutions using the acquisition of information on the problem under study. In this study this is done through the use of reinforcement learning. The performance of a metaheuristic can also be improved using multiple search trajectories, which act competitively and/or cooperatively. This can be accomplished using parallel processing. Thus, in this paper we propose a hybrid parallel implementation for the GRASP metaheuristics and the genetic algorithm, using reinforcement learning, applied to the symmetric traveling salesman problem.

Keywords: Parallel Processing, GRASP, Genetic Algorithm, Q-learning Algorithm.