7
Conclusão e Trabalhos Futuros

Esta tese apresentou um método de otimização estatística de buscas para estruturas hierárquicas do tipo 2^d–*trees*. Essa otimização requer apenas que os nós da estrutura possam ser acessados em tempo constante e indexados por um critério que satisfaça quatro requisitos. Um exemplo completo dessa otimização foi descrito para *hashed* 2^d–*trees*, tendo seus nós armazenados numa *hash table* e indexados pelo método de códigos de Morton. Os quatro requisitos foram provados neste caso. Além disso, operações de inteiros eficientes para melhorar o desempenho dessa estrutura foram apresentadas em dimensão qualquer. O método foi experimentado para $d = 3$, comparado com a representação clássica de uma 2^d–*tree*, o novo método reduz, tanto o tempo de execução por um fator médio de 4 vezes, como o consumo de memória utilizada, em média, metade da memória exigida pela representação clássica. No caso de uma 2^d–*tree* aberta, o método mantém o mesmo consumo de memória, mas melhora, em média, 15 vezes o tempo de execução.

Este trabalho pode ser aprimorado integrando mais ainda a estrutura hierárquica com a *hash table*. Por exemplo, pode-se definir uma maneira de ajustar seu tamanho automaticamente, de acordo com o dado e com a escolha do usuário em dar preferência à memória ou ao tempo de execução. Novas funções de *hash* que distribuam uniformemente as chaves na *hash table* podem ser criadas. Outras funções de *hash* tais como *linear hashing* ou *spiral hashing* deverão ser usadas quando é exigido um comportamento dinâmico da estrutura com exaustivas inserções e remoções. Para um armazenamento em disco, devem ser usadas funções de *hash* que preservem a proximidade das chaves para garantir boa coerência nos *buckets*. No caso do uso de listas encadeadas, critérios podem ser escolhidos para o gerenciamento das listas, definindo as posições por acesso e melhorando o *cache*.

Este trabalho pode ser estendido estudando modelos de custo mais precisos e adaptando-os a outras estruturas de busca, tais como *kd-tree* e o *particionamento binário do espaço*. Neste caso, devem ser definidas indexações compatíveis para cada uma.