Depth-First Search *versus* Jurema Search on GPU Branch-and-Bound Algorithms: a case study

Tiago Carneiro Ricardo Nobre∗ Marcos Negreiros⋆
Gustavo Augusto Lima de Campos
Mestrado Acadêmico em Ciências da Computação (MACC), Universidade Estadual do Ceará, Brazil
∗Serviço Federal de Processamento de Dados, Brazil
⋆Mestrado Profissional em Computação Aplicada, Universidade Estadual do Ceará, Brazil

Abstract

Branch-and-Bound (B&B) is a general problem solving paradigm and it has been successfully used to prove the optimality of combinatorial optimization problems. The development of GPU-based parallel Branch-and-Bound algorithm is a brandnew and challenging topic on high performance computing and combinatorial optimization, motivated by GPU's high performance and low cost. This work presents a strategy designed to parallelize Jurema search-based B&B algorithms on GPUs, evaluated for the Asymmetric Traveling Salesman Problem, called *Juremal*. Jurema search is a search mainly based on DFS concepts, developed to mitigate DFS-B&B flaws. Results show the search strategy chosen (DFS or Jurema) is not critical. But it is necessary to develop a trigger mechanism to determine when the process must be halted, and how the remaining search space must be redistributed. Strategies to reduce serialization of instructions are also need, in order to obtain higher speedups in GPU DFS-B&B based algorithms.

Keywords:: Branch-and-Bound, ATSP, CUDA, Depth-first Search, GPGPU

Author's Contact:

{carneiro,gustavo}@larces.uece.br
ricardo.nobre@serpro.gov.br
negreiro@graphvs.com.br

1 Introduction

The Branch-and-Bound (B&B) is a general problem solving paradigm and it has been successfully used to prove the optimality of computational combinatorial optimization problems, such as Traveling Salesman Problem (TSP), Capacitated Vehicle Routing Problem, Job Shop Scheduling, etc. All B&B algorithms follow three main concepts: branching, bounding and pruning. Branching is a step that breaks the problem, represented by a node, into sub problems, represented by its nodes sons. The recursive application of the branching stage produces a tree called B&B tree. The second concept is the concept of bounding, i.e., the use of upper and lower bounds to guide the search strategy and evaluate sub problems. The last concept is the act of eliminating large portions of the solutions space, called pruning.

The problems the B&B method commonly solves are NP-Hard, which means these problems are computation-intensive problems, since a large solutions space must be fully evaluated in order to prove the optimality. To deal with these possibly large solutions space, B&B algorithms has been successfully parallelized along the years.

Historically, a parallel B&B algorithm may belong to one of these three classes [Gendron and Crainic 1994]: algorithm that parallelizes the nodes evaluation and produces the same tree as the equivalent serial algorithm; centralized master-slave algorithm, where a central entity distributes nodes to the processing unities; parallel B&B where a lot of B&B threes are constructed in parallel by independent B&Bs that share upper bounds among them and perform load balancing. This class is known as *Collegial B&B*.

The development of GPU-based parallel Branch-and-Bound algorithm is a brandnew and challenging topic on high performance computing and combinatorial optimization, motivated by GPU's high performance and low cost. Parallel search in GPUs is a challenging topic due to GPUs inherent characteristics and the irregularity of the trees produced by Depth-first search (DFS).

This work presents a parallel strategy to parallelize Jurema search-based B&B algorithms on GPUs, evaluated for the Asymmetric Traveling Salesman Problem, called *Juremal*. The Jurema search here used is a search mainly based on DFS concepts, developed to mitigate DFS-B&B flaws. Results show that in GPU-based DFS-B&B algorithms, unlike serial B&B, the search is not critical, but the development on triggering mechanisms and strategies to reduce serialization of instructions are need, in order to obtain higher speedups.

The remainder of this paper is organized as follows. Section 2 presents the related works. Section 3 presents a brief introduction about Jurema search. Section 4 presents the proposed GPU B&B schema based on Jurema search, called Juremal. Section 5 presents details about computational evaluation of the method. In Section 6 conclusions and future remarks are considered.

2 Related Work

[Thompson et al. 2002] is the pioneer work of solving combinatorial optimization problem by using GPUs. Using only graphical API and low level languages, solved instances of max-sat problem employing genetic algorithms.

[El Baz et al. 2010] is the pioneer in studies about GPU-Based B&B. Its aim is to verify how the value of threads per block may affect the overall computation. [Chakroun et al. 2011] presents a parallel B&B for solving Flow-Shop scheduling problem. They got good speedups on performing bound evaluation in parallel and its aim is to reduce instructions divergence. [Kurowski and Mackowiak 2011] has few information about the implementation of the parallel B&B presented, once its objective is to provide an framework designed to use volunteer's GPU for solving TSP instances, and solved a 24 cities symmetric instance. It's important to empathize that all mentioned GPU B&B are parallelized by performing lower-bound evaluation in parallel (master-slave).

Clausen and Perregaard (1999) showed that Best-first Search (BeFS) is not superior than DFS in parallel B&B. Concurrent DFS might find new solutions quickly, and allied with synchronization of upper bounds and DFS's inherent advantages, such low space complexity possibility of postpone bound evaluation, results in a efficient pruning process. Carneiro et al. (2011a) presents a new parallel schema for implementing parallel DFS-B&B (Collegial) on GPUs. The aim of this work is to use the GPU to exploit the advantages of DFS in parallel B&B algorithms, and evaluate the solutions space in parallel. This work showed that in a theoretical worst case, GPU-based DFS-B&B might be dozen of times faster than the equivalent serial algorithm. Carneiro et al. (2011b) complements the work Carneiro et al. (2011a), and showed that in GPU DFS-B&B the amount of nodes generated initially by the serial portion is crucial to the performance of a GPU DFS-B&B algorithm, and that instances with lower bound distant from the value of an optimal solution are the most suited to be processed by a GPU DFS-B&B.

The size of the instance solved by all B&B algorithms present in this section, and the great disparity between the serial state-of-the-art algorithms for solving combinatorial optimization problems and the related GPU B&B algorithms, evidence the early stage of the GPU B&B algorithm's development.

3 Jurema Search

Generally, to determine the way in which nodes will be branched, B&B algorithms use Depth-first search (DFS) techniques, where the most recently generated sub-problem is explored first, or Best-first search (BeFS), that explores the most promising sub problem first.

BeFS is optimal in the number of nodes branched [Grama and Kumar 1993], but, in order to prove the optimality, this search strategy needs to keep a large number of nodes in a priority queue, which makes the BeFS's space complexity exponential in the search depth and suitable only for smaller or easier instances [Zhang and Korf 1993].

Depth-first search, in other hand, has space complexity linear in the search depth and can find new upper bounds quickly [Zhang 2000]; these new solutions may not be an optimal solution, but these new upper bounds make the bounds tighter, leading to a more effective pruning process, which makes DFS suitable to harder instances. The greatest flaw of DFS is the search can get stuck in unpromising branches and generate a B&B tree greater than BeFS one.

Jurema [Pessoa and Negreiros 2010] is a search strategy conceived to mitigate Depth-first search weakness by using the best of Best-first search and Depth-first search. Each node discovered by the breadth stage of BeFS search is evaluated, and the most promising node will be explored first, as a DFS root. This search is performed from a leaf to the root, in order to find new solutions quickly and to narrow the bounds, leading in a more efficient pruning process. Thus, Juremal needs an initial solution, called Guide Solution. The related method showed to be far superior than DFS-B&B, in what concerns to the resulting tree's size and time to prove optimality, mainly when the initial upper bound is quite distant to the optimal solution.

4 *Juremal* for solving the ATSP

Carneiro et al. (2011a) presented a new parallel procedure designed to process combinatorial GPU B&B algorithms, and evaluated to the Asymmetric Traveling Salesman Problem (ATSP). This schema is divided in two steps: Active Set creation and concurrent DFS-B&B. The Active Set creation performs B&B sequentially till a specific depth, saving the current path in B&B tree as nodes into the *Active Set*. The Active Set is a set that keeps nodes evaluated, but not yet branched. On this schema, each node in the *Active Set* is a DFS-B&B root R_i, and the Active Set will be concurrently processed by a massive number of DFS-B&Bs. The massively parallel version of Jurema Method, unlike Carneiro et al (2001a), is divided in tree major steps: Active Set creation, construction of the pool of Guide Solutions and concurrent Jurema Searches, justifying the chosen name, *Juremal*. Juremal means, in Portuguese, "a wood of Juremas".

Juremal, as Carneiro et al. (2011a)'s, starts performing serial B&B till a specific depth of the solutions space, informed by the programmer, based on instance's or program's characteristics, creating the initial Active Set, as shown in Figure 1.

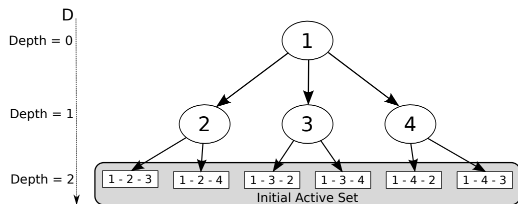

Figure 1: *Initial Active Set population by a serial DFS-B&B.*

Once Jurema search needs a initial solution, called guide solution, to perform the leaf-to-root B&B, each node belonging to the Active Set, i.e., incomplete Hamiltonian Cycles and its properties, will be basis of a constructive heuristic for solving the ATSP, in order to generate the Guide Solutions. These solutions will be stored in a pool of Guide Solutions, as show in Figure 2. Constructive heuristic used in current implementation was the Nearest Neighbor, who always inserts in the sub-cycle the cheapest neighbor.

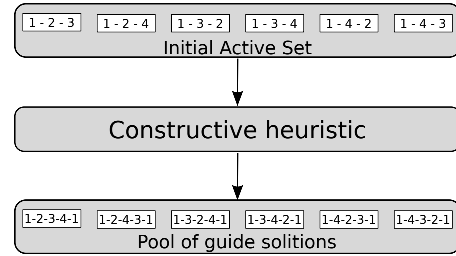

Figure 2: *Figure that illustrates the creation of the pool of guide solutions.*

Consider x the depth of the solution space that must be evaluated by the serial DFS-B&B in order to generate the initial Active Set, and consider n the ATSP's instance's size, then, the *Active Set* will contain, at most, $\frac{(n-1)!}{x!}$ nodes. Each one of these nodes represents an incomplete Hamilton cycle containing the start city plus x different cities. Thus, the pool of guide solutions will contain the size of the Active Set solutions. Each guide solution g_i of the pool also keeps its own cost as an information too.

After the creation of the pool of Guide Solutions, data is stored on GPU and the *kernel* is launched. The kernel is an non-recursive Jurema search with some minor modifications necessary to adapt the code to be processed by GPU. In Juremal, as shown in Figure 3, each thread T_i will evaluated the sub-solution space S_i using Guide Solution g_i as guide. A high level algorithm for Juremal is show in Algorithm 1. Details about the kernel are presented as follows.

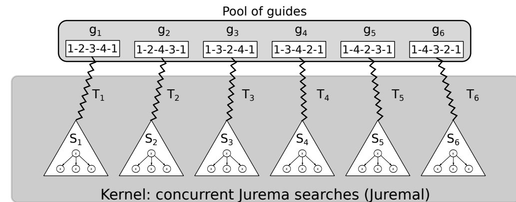

Figure 3: *Concurrent Jurema searches: Juremal.*

Algorithm 1: Juremal's high level algorithm.

Data: instance I
Result: An optimal solution for I and the amount of solutions found.
1 $p = get_GPU_properties()$;
2 $d = get_depth_of_solutions_space()$;
3 $z = calculate_initial_upper_bound(I)$;
4 $l = calculate_initial_lower_bound(I)$;
5 $A = generate_initial_active_set(p, z, l, d, I)$;
6 $G = generate_pool_of_guides(A)$;
7 $z = get_the_lower_guideSolution_value(G)$;
8 $t = calculate_threads_per_block(p, A)$;
9 $b = calculate_number_of_blocks(t)$;
10 $allocate_GPU_data(I, G)$;
`// Kernel.`
11 $juremal <<< b, t >>> (z, l, G, I)$;
12 $synchronize_GPU_CPU_data(z)$;

4.1 Kernel

As previously mentioned, Juremal performs a massive number of Jurema searches in parallel. This kernel is an adapted version of Jurema Method for solving the ATSP [Pessoa and Negreiros 2010], with few modifications. Serial Jurema sorts nodes discovered by the breadth stage of BeFS, in order to explore the most promising regions of the solutions space first. In a GPU, this may cause threads divergences, once each pool of nodes may have different characteristics. Furthermore, sorting requires extra memory for control. Thread's memory is a scarce class of memory in GPU's memory organization. In a worse scenario, only one thread of the warp may

need to evaluate its pool of nodes, an unwanted situation for GPUs, once they process in warps of threads. So, Juremal, unlike serial Jurema, is not a combination of DFS and BeFS, but a combination of DFS plus pure Breadth-first search. Algorithm 2 shows a high level algorithm for Juremal's kernel.

Algorithm 2: Juremal's kernel.

Data: instance I, pool of Guide Solutions G, actual upper bound z.
Result: amount of solutions found by all *threads* and an optimal solution for I.

```
1  ⋃ = z;
2  se thread_id == 0 então
3      block_upper_bound = ⋃;
4  fim
5  local variables initialization.;
6  synchthreads();
   // Region processed by thread thead_id:
7  local_cost = G[thead_id].cost;
8  localGuideSolution = G[thead_id];
9  Start the non-recursive Jurema Search using G[thead_id] as Guide
   Solution;
10 Synchronization of variables that will be returned to CPU;
11 synchthreads();
```

Synchronization among threads is something really costly in GPU computing. To deal with this trade-off, in our schema the threads are divided in blocks, and each block has its own upper bound shared by its threads (which is quite faster in the GPU architecture). Threads evaluate the solution space based on block's upper bound, and blocks synchronize upper bounds among them less often.

Juremal's kernel needs $O(4 * N)$ extra local memory per thread, where N is the ATSP instance's size, in order to perform the search. Carneiro et al. (2011b) GPU DFS-B&B has space complexity $O(2 * N)$. These complexities are equivalents, both are in $O(N)$, but the extra memory required by Juremal may result in less threads running concurrently. Concerning to the use of global memory, Juremal has space complexity $O(|A| * N)$, Carneiro et al.'s (2011b) has $O(|A| * D)$, where D is the depth of the search space, informed by the programmer, explored by the Active Set creation.

Juremal creates a massively number of initial solutions; so, an advantage of this method, when compared to GPU DFS-B&B, is Juremal can start the search with an solution closer to an optimal solution, and, as a consequence, explore a smaller solutions space.

5 Experimental Results

Carneiro et al. (2011) showed that for a DFS-based B&B algorithm, the size of the solutions space that must be fully evaluated, in order to prove optimality, is a critical issue, and small instances with low lower bounds are ideal to be processed by GPUs. Thus, an ideal scenario was created. The Assignment Problem, used by all implementations as a lower bound to the ATSP solution, in this section evaluated, has no restriction that forbids sub-cycle creations, as show in the formulation bellow.

(AP): Minimize $\sum_{i,j} c_{ij} x_{ij}$

Subject to:

$$\sum_{j=1}^{n} x_{ij} = 1, \quad j = \{1, ..., n\} \tag{1}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad i = \{1, ..., n\} \tag{2}$$

$$x_{ij} \geq 0, \quad i, j = \{1, ..., n\} \tag{3}$$

, where c_{ij} is the cost to assign city i to city j.

When using the ATSP as a lower bound, is necessary to set the values of the cost matrix, in elements belonging to the main diagonal, to infinity, so the AP can't choose this elements and the lower bounds to the ATSP higher. For some instances by Cirasella et al. (2001), when this pre-processing is not done, the lower bound returned by the AP is notably low, resulting in loose bounds and in a huge solutions space.

Cirasella et al. (2001) generates instances using properties found in real-world situations. The selected instances were: Coin (14 to 18 cities), Flow (14 to 18 cities). The instances *coin* are those of collecting money from pay phones in a grid-like city, and the instances *flow* are those from no-wait flow shop for processing of heated materials that must not be allowed to cool down. According to Johnson et al. (2004), instances made by Cirasella et al. (2001) are commonly harder than merely random instances.

Five parallel implementations are considered in this section: three DFS-B&B and two Juremas. Two DFS-B&B are multicore and other is Carneiro et al. (2011b) GPU DFS-B&B (DFS-MP). One multicore DFS-B&B (DFS-PE) is statically allocated, very similar to the GPU DFS-B&B. The other has the load dynamically allocated (DFS-PD), but with simple rules. See Carneiro et al. (2011b) for more informations. Jurema multicore (Jurema-P) uses the evaluated nodes as a pool and shares load among processors.

All implementations were conceived to prove optimality of Asymmetric Traveling Salesman Problem's instances. The Traveling Salesman Problem (TSP) is the problem to find a shortest Hamiltonian cycle though a given number of cities, in such way each city is visited only once. It is the most well-known and studied combinatorial optimization problem of the history, and has plenty real-world applications [Laporte 2006]. The versions of the TSP can be defined as symmetric, if the cost matrix is symmetric ($\forall[i, j] : c_{ij} = c_{ji},$), asymmetric otherwise ($\exists[i, j] : c_{ij} \neq c_{ji}$). The asymmetrical case is a more general way of representing TSP, and its instances are frequently harder to solve than the symmetric case's instances [Zhang 2004].

The Assignment Problem (AP) was employed as a lower bound and, as an upper bound, Karp's modified patching procedure (KPP) was employed. All implementations use Tucker's (1994) schema, that uses AP's dual variables in order to produce the residual cost matrix, to perform B&B. In Table 1 are present the gaps obtained by AP and KPP, in relation to the optimal solution of each instance.

Table 1: *Gaps obtained by AP and KPP, in relation to the optimal solution of each instance.*

Instance	AP	KPP
Coin14	-100,0%	17,24%
Coin15	-100,0%	9,67%
Coin16	-100,0%	3,22%
Coin17	-100,0%	0,0%
Coin18	-100,0%	0,0%
Flow14	-46,9%	1,40%
Flow15	-45,8%	4,24%
Flow16	-46,05%	4,17%
Flow17	-45,88%	7,57%
Flow18	-46,78%	7,76%

All implementations were made with $C++$. GPU DFS-B&B and Juremal, made with $C++$ and parallelized with NVidia's CUDA. Multicore implementations were parallelized by using OpenMP 3.0. All implementations were compiled by *GCC 4.4.6*.

The parameter employed to compare the implementations is the speedup. The speedup means the benefit of solving a problem in parallel, i.e., how many times the parallel code using n processors is faster than the serial code doing the same job [Akl 1989], and is represented as follows:

$$S(n) = \frac{t_s}{t_n} \tag{4}$$

where t_s is the serial time, and t_n the parallel time with *n* processors.

Experiments were conducted on an Intel Core i7 2600 (3.4 GHz), with eight cores; the system had 4 GB RAM and Ubuntu GNU Linux 11.10 OS (kernel 3.0.1). The GPU used to run CUDA codes was a NVidia GeForce GTX 580. This GPU has 512 stream processors at 1,544 MHz; 1.5 GB of GDDR5 RAM at 2.004 GHz; 384 bits memory interface.

Figure 4 shows the speedup obtained by each parallel implementation in relation to the serial DFS-B&B. In Figure 5 can be seen the average speedup obtained in relation to the serial DFS-B&B.

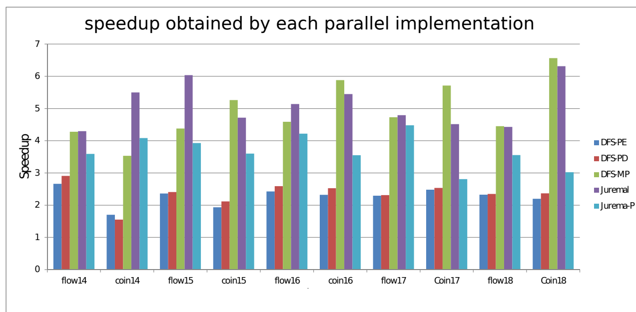

Figure 4: *Speedup obtained by each parallel implementation.*

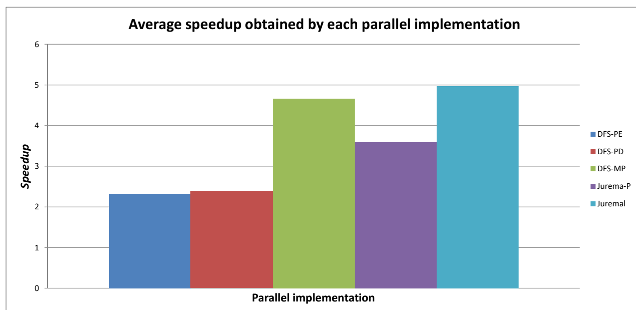

Figure 5: *Average speedup obtained by each parallel implementation.*

Figures 4 e 5 show that, unlike in serial and multicore B&B, in GPU-based B&B Jurema search is only slight better than DFS. Each one of the massively parallel implementations were better than the other in five instances. Juremal dominated all multicore implementations, but the massively parallel DFS-B&B was slower than Jurema multicore in coin15 stance. The reason is: coin15's initial upper bound, among all initial upper bounds, is the most distant form the optimal solution's value, an ideal scenario to Jurema search.

6 Conclusions and future research directions

Despite the slight superiority of Juremal, when compared to GPU DFS-B&B, we can conclude that unlike serial scenarios, where Jurema Search is notably superior than DFS-B&B, in GPU B&B the search strategy employed is not a critical factor. The main issue found is to create a way to reduce warp divergence, responsible to cause GPU sub-utilization and, as a consequence, low speedups.

According to Grama and Kumar [Grama and Kumar 1993], subtrees generated by DFS tend to be irregular, and work statically allocated tend to result in load imbalance among processors. Juremal uses DFS, hence, during its execution a lot of threads tend to be idle, once the solution space that the thread needs to evaluate is pruned or fully evaluated. To make an effective use of available processors, workload balance schema is a critical issue when planning a parallel B&B algorithm. The current GPU implementation has no trigger to determine when the process must be halted, blocks recalculated, and the remaining search space should be redistributed; so, at certain time of kernel's execution, warps may have few active threads. Therefore, a critical future work is to implement such mechanism, based on Karypis and Kumar [Karypis and Kumar 1994] triggering and redistributing schema for SIMD systems, to provide a more efficient use of GPU's processors and, as a direct consequence, better speedups.

References

AKL, S. 1989. *The design and analysis of parallel algorithms*. Old Tappan, NJ (USA); Prentice Hall Inc.

CARNEIRO, T., MURITIBA, A. E., NEGREIROS, M., AND DE CAMPOS, G. A. L. 2011. A new parallel schema for branch-and-bound algorithms using gpgpu. *Computer Architecture and High Performance Computing, Symposium on 0*, 41–47.

CARNEIRO, T., MURITIBA, A. E., NEGREIROS, M., AND DE CAMPOS, G. A. L. 2011. Solving atsp hard instances by new parallel branch and bound algorithm using gpgpu. *Iberian Latin American Congress on Computational Methods in Engineering (CILAMCE)*.

CHAKROUN, I., BENDJOUDI, A., MELAB, N., ET AL. 2011. Reducing thread divergence in gpu-based b&b applied to the flow-shop problem.

CLAUSEN, J., AND PERREGAARD, M. 1999. On the best search strategy in parallel branch-and-bound: Best-first search versus lazy depth-first search. *Annals of Operations Research 90*, 1–17.

EL BAZ, D., DUMAS, L., BOYER, V., ELKIHEL, M., AND ENJALBERT, J. 2010. Parallélisation de méthode doptimisation entière sur gpu. *Onzieme Congres de la Societe Francaise de Recherche Operationnelle et d'Aide a la Decision*.

GENDRON, B., AND CRAINIC, T. 1994. Parallel branch-and-bound algorithms: Survey and synthesis. *Operations Research*, 1042–1066.

GRAMA, A., AND KUMAR, V. 1993. A survey of parallel search algorithms for discrete optimization problems. *ORSA Journal on Computing 7*.

KARYPIS, G., AND KUMAR, V. 1994. Unstructured tree search on simd parallel computers. *Parallel and Distributed Systems, IEEE Transactions on 5*, 10, 1057–1072.

KUROWSKI, K., AND MACKOWIAK, M. 2011. Parallel branch and bound method for solving traveling salesman problem using cpu and gpgpu volunteer computing and the xmpp protocol. *The 25th IEEE International Conference on Advanced Information Networking and Applications (AINA-2011)*.

LAPORTE, G. 2006. A short history of the traveling salesman problem. *Canada Research Chair in Distribution Management, Centre for Research on Transportation (CRT) and GERAD HEC Montréal, Canada*.

PESSOA, T., AND NEGREIROS, M. 2010. jurema, a new branch & bound anytime algorithm for the asymmetric travelling salesman problem. In *XLIII Simpsio Brasileiro de Pesquisa Operacional*.

THOMPSON, C., HAHN, S., AND OSKIN, M. 2002. Using modern graphics architectures for general-purpose computing: A framework and analysis. In *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, IEEE Computer Society Press, 306–317.

TUCKER, A. 1994. *Applied combinatorics*, 3rd ed. Wiley.

ZHANG, W., AND KORF, R. 1993. Depth-first vs. best-first search: New results. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, JOHN WILEY & SONS LTD, 769–769.

ZHANG, W. 2000. Depth-first branch-and-bound versus local search: A case study. In *proceedings of the National Conference on Artificial Intelligence*, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 930–936.

ZHANG, W. 2004. Phase transitions and backbones of the asymmetric traveling salesman problem. *Journal of Artificial Intelligence Research 21*, 1, 471–497.