DISTRIBUTED HAMILTONIAN PATH SEARCH ALGORITHM

DISTRIBUOVANÝ ALGORITMUS PRO HĽADANIE HAMILTONOVSKEJ CESTY

Karol Grondžák[1]

Summary: Many practical problems of transportation can be transformed to the problem of finding Hamiltonian path or circle. It was proven, that this problem is NP-complete and thus can be very time-consuming for practical problem size. In this paper we present a distributed algorithm to search for Hamiltonian path in a graph.

Key words: vehicle routing, Hamiltonian path, Message Passing Interface

Anotácia: Mnohé praktické problémy v doprave môžu byť transformované na problém hľadania hamiltonovskej cesty alebo kružnice. Je dokázané, že tento problém je NP-úplný, takže môže byť časovo náročný pre úlohy riešené v praxi. V tomto príspevku prezentujeme distribuovaný algoritmus pre hľadanie hamiltonovských ciest na grafoch.

Kľúčové slová: smerovanie vozidiel, hamiltonovská cesta, Message Passing Interface

1. INTRODUCTION

Hamiltonian path is well-known and often used concept of graph theory. Let us consider undirected and unweighted graph $\mathbf{G} = (V, E)$, where V is a set of vertices (nodes) and E is set of edges. Edge is a tuple $\{u, v\}$ such that $u \in V, v \in V$. Hamiltonian path is a path in an undirected graph $\mathbf{G}$ which visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is defined as a cycle in an undirected graph $\mathbf{G}$ which visits each vertex exactly once and starting and final vertex are identical. Determining whether such paths and cycles exist in graphs is the Hamiltonian path problem. It can be formulated as decision-making combinatorial problem ([1]).

To define combinatorial algorithm, let us consider a discrete set X, a function $F : X \to \Re$, and a set of feasible solutions S, where $S \subseteq X$. The feasible solution is defined in terms of given constraints specific for a particular problem. Decision-making combinatorial problem is the task to construct elements of set X and find (if possible) at least on element x belonging to set S, i.e. $x \in X \wedge x \in S$. Common technique to search for solution is backtracking algorithm ([1]).

[1] Ing. Karol Grondžák, PhD., Univerzity of Žilina, Faculty of Management Science and Informatics, Department of Informatics, Univerzitná 8215/1, 010 26 Žilina, Tel.: +421 41 5134173, Fax: +421 41 5134055, E-mail: Karol.Grondzak@fri.uniza.sk

2. DISTRIBUTED HAMILTONIAN PATH ALGORITHM DESIGN

As was mentioned above, backtracking algorithm is used to solve the problem of finding Hamiltonian path. General backtracking algorithm can be formulated as follows. Let us consider n sets:

$$E_k = \{e_1^k, e_2^k, \ldots, e_{m_k}^k\}, k = 1, \ldots n, \quad (1)$$

where m_k is a number of elements of the set E_k and e_i^k is i-th element of the set E_k. Then the above mentioned set X is defined as Cartesian product of sets $E_i, \forall i$:

$$X = E_1 \times E_2 \times \cdots \times E_n, \quad (2)$$

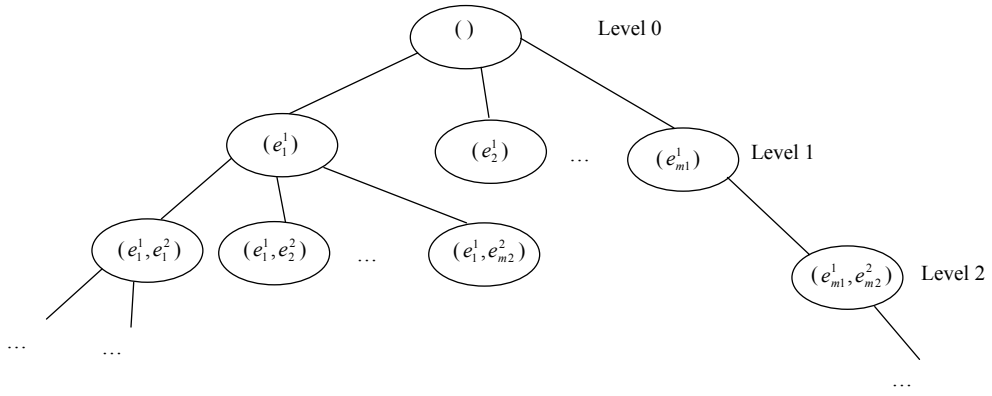
with cardinality $Q = \prod_{i=1}^{n} m_i$. It is obvious that the set X consist of n-tuples:

$$\left(x_1^c, x_2^c, \ldots x_n^c\right), c = 1,2,\ldots Q, \quad (3)$$

such that $x_i^c \in E_i$. These n-tuples are constructed recursively extending the set of $(n-1)$-tuples.

The backtracking algorithm starts with an empty n-tuple. In the stage i the $(i-1)$-tuple is extended using elements of the set E_i. Newly obtained i-tuples are then checked for feasibility and then expanded to $(i+1)$-tuples using elements from set E_{i+1}.

This process is actually a depth-first search of a search-tree. Nodes of the search tree at i-th level consist of i-tuples. An example of the search-tree is in Fig. 1.



Source: Author

Fig. 1 - Example of a search-tree

This general backtracking algorithm can be applied to search for Hamiltonian path. For graph with n vertices, we are looking for a list of vertices $(v_0, v_1, \ldots v_{n-1})$ such that each tuple $\{v_{i-1}, v_i\} \in E, i = 1,2,\ldots n-1$ and for $i \neq j, v_i \neq v_j$. We start with list containing starting vertex. Then this list is expanded by another vertex of graph G. The feasibility of this particular solution is checked. If it is not feasible, another expansion is not performed and this solution is discarded. The process of construction of search-tree can be time-consuming for deep trees.

2.1 Parallel computer architectures

Since the beginning of computer era there were efforts to speed-up calculations performed on computers. One approach is to increase performance of computer components, e.g. frequency of CPU or main memory. This approach is limited by physical properties of used materials. Another approach is to divide calculation into independent portions and perform calculations in parallel.

Two main architectures are used in this approach, considering the organization of main memory. If all the processors share main memory, we call this shared memory architecture. Communication among processes running on different processors is performed using main memory. For this architecture, OpenMP standard ([2]) was defined.

If the processors have separate main memory, we call this distributed memory architecture. In this case communication among processes running on different processors must be performed by exchanging messages. For this architecture, Message Passing Interface (MPI) ([3]) was proposed by group of computer hardware producers.

2.2 Parallel Backtracking Algorithm

Analysis of backtracking algorithm reveals a possibility for speed-up. Nodes of search-tree of the backtracking algorithm can be distributed among different CPUs and searched in parallel. This approach is suitable for both shared memory and distributed memory architectures ([4]). In general, it can be realized by master-worker paradigm. One process (master) maintains the pool of tasks and distributes them among worker processes. Worker processes ask master process for task and after processing it inform master about results. General master-worker algorithm is shown in Fig. 2. This approach is fully applicable to the problem of Hamiltonian path search.

```
WHILE workers not finished
DO
  wait for a message from worker
  IF solution found
    notify all workers
    finish
  ELSE
  IF request for a job
    send job to worker
END
```

(a)

```
REPEAT
  ask for a job from master
  IF there is a job
    process assigned job
    IF solution found
      notify master
 UNTIL there is a job
```

(b)

Source: Author

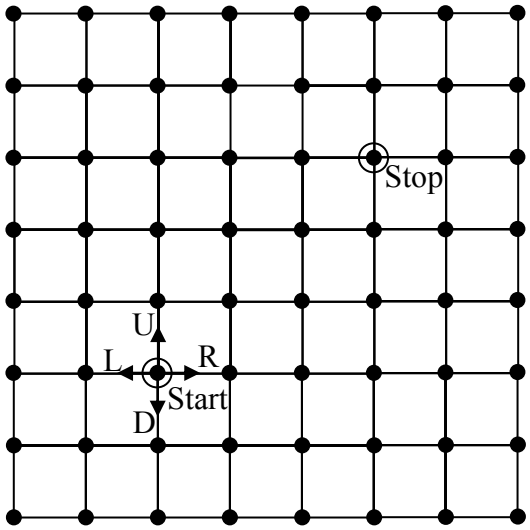
Fig. 2 – Pseudo-code for master-worker paradigm algorithm

3. EXPERIMENT AND OBTAINED RESULTS

Algorithm proposed in previous parts was implemented on a cluster of 20 commodity personal computers. Each computer was equipped with dual core processor Intel D 3.0GHz and 1024MB of main memory; running Debian GNU/Linux operating system. Computers

were connected using standard Ethernet 1GB/s infrastructure. Because of the distributed memory architecture of used hardware, OpenMPI ([5]) was used as tool for communication between processes. It is a freely available implementation of MPI specification for GNU/Linux operating system.

The properties of proposed algorithm were studied on the following problem. Let us consider rectangular area divided into regular mesh (Fig. 3). There are two special vertices – Start and Stop. We are looking for a Hamiltonian path starting in vertex Start and ending in vertex Stop. If Start and Stop vertices are identical, we are searching for Hamiltonian circle. It can represent a problem of routing small vehicle performing maintenance of the area. Or it can be a problem of routing the arm of some robot performing some tasks in vertices of graph (e.g. drill hole in each position).



Source: Author

Fig. 3 – Topology of studied problem

Because of the regular nature of this problem, we can represent the solution in the form of direction which should vehicle take from actual position. There are at most four possible directions from a vertex – left (L), right (R), up (U) or down (D). Then for a problem of size N (amount of nodes in one direction), the maximum possible number of combinations is:

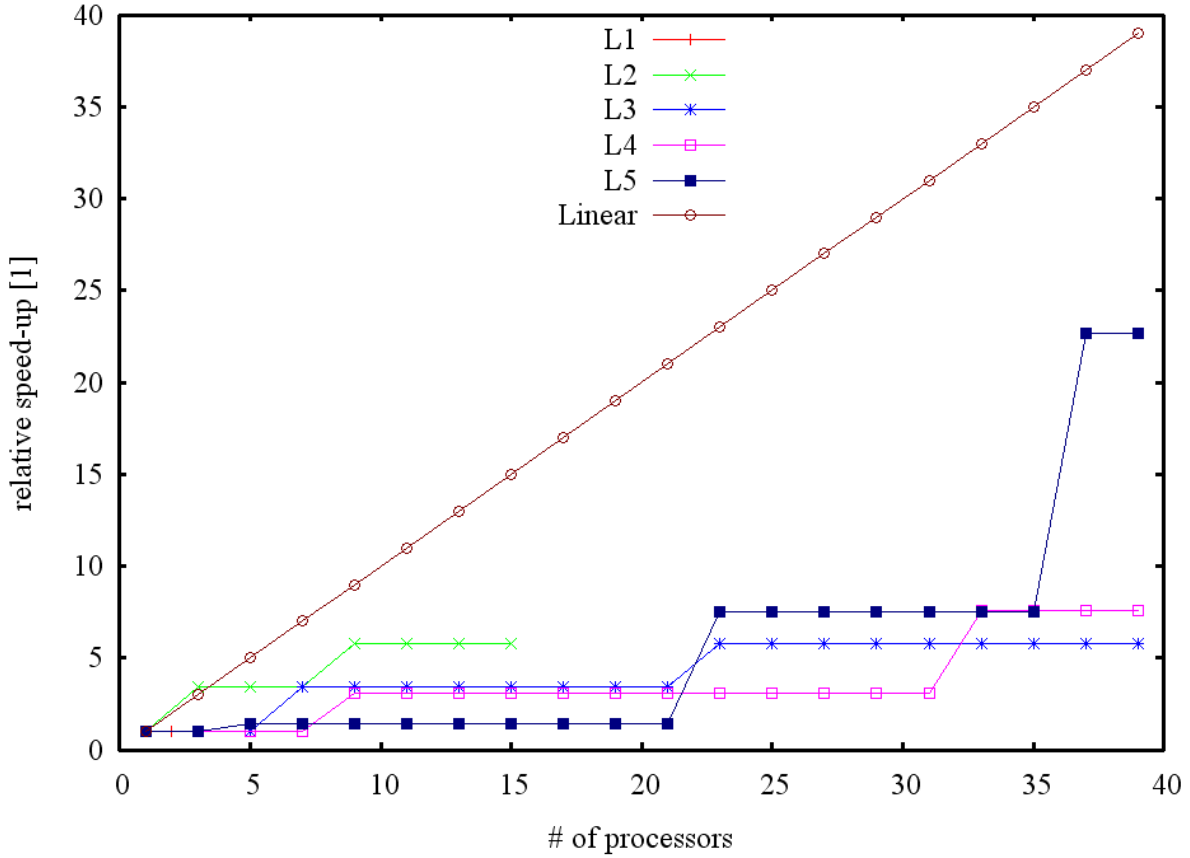
$$P_{\max} = 4^{N^2-1}. \quad (4)$$

Many combinations are not feasible solution and can be pruned during the process of solution construction. Backtracking algorithm starts with empty list and vehicle in starting position. In first step vehicle can move to one of four directions, so at the first level of search-tree we get four lists containing one direction (L, R, U or D). Next step would be to expand this list by second move. It can be (theoretically) again one of four directions. This will result in a 16 list of two directions. In each step we have to check some constrains:

- we can not step out of mesh
- we can not enter vertex, which has already been entered in previous steps
- we can not enter Stop vertex, if it is not last step (if this holds, we have found solution)

These conditions allow us to prune non-feasible partial paths without having to construct and evaluate whole combination.

Two different aspects of algorithm were studied. First was the performance of algorithm with respect to the number of processors involved in calculation. When involving more processors we expect the algorithm to be faster. Optimal achievable speed-up is linear (if we do not consider some special cases). This can be achieved, when the problem is divided into equal independent parts which can be processed by different processors in parallel. This would be true, if all the sub-trees of search-tree were of the same depth. In reality the search-tree is usually unbalanced (as showed our experiments). For a fixed number of processors available it is reasonable to divide task into smaller parts. Then we can expect that processors which will be assigned shallower sub-trees of search-tree will process larger amount of tasks and processors which will be assigned deeper sub-trees will process less tasks. This was the second aspect to study – the influence of number of tasks to the algorithm performance.



Source: Author

Fig. 4 – Relative speed-up vs. number of processors involved in calculation

The results presented at Fig. 4. were obtained for the same configuration of the problem. They differ by the level of search-tree, at which the tasks were distributed to worker processes. For level value 1, there are four sub-trees, which can be distributed among at most four processes (4^1). For level 5, there are $4^5 = 1024$ sub-trees to distribute and process by worker processes. It can be seen, that as the level increases, the relative speed-up increases as

well. It is also clearly seen, that the speed-up is nonlinear, due to fact, that search-tree is highly unbalanced. It can be also seen, that the more processors is involved in calculation, the faster it is. For convenience also optimal linear speed-up is shown (line Linear).

4. CONCLUSION

In this contribution we have presented a proposed distributed algorithm for Hamiltonian path search. The performance of newly proposed algorithm was compared with serial algorithm. We have demonstrated the fact, that search-tree for the studied problem is unbalanced. The potential for speeding-up of Hamiltonian path search was demonstrated. To optimally use the potential of distributed architecture, some technique of load balancing should be applied.

REFERENCES

[1] KUČERA, L. *Kombinatorické algoritmy.* Praha: SNTL, 1989. 286 pp. ISBN 04-009-89.
[2] *OpenMP* [online]. c2010 [cit. 2010-11-19] Available from <http://openmp.org/wp/>
[3] *MPI Documents* [online]. c2010 [cit. 2010-11-19] Available from: <http://www.mpi-forum.org/docs/docs.html>.
[4] WILKINSON, B., ALLEN, M. *Parallel Programming*. Upper Saddle River: Pearson Education, 2005. 467pp. ISBN: 0-13-140563-2
[5] *OpenMPI* [online] c2010 [cit. 2010-11-19] Available from <http://www.open-mpi.org/>