A Fast and Efficient Algorithm for Topology-Aware Coallocation

Valentin Kravtsov[1], Martin Swain[2], Uri Dubin[1],
Werner Dubitzky[2], and Assaf Schuster[1]

[1] Technion-Israel Institute of Technology, Technion City, 32000, Haifa, Israel
svali_ds@cs.technion.ac.il
[2] University of Ulster, Cromore Road, Coleraine, BT52 1SA, Northern Ireland, UK

Abstract. Modern distributed applications require coallocation of massive amounts of resources. Grid level allocation systems must efficiently decide where these applications can be executed. To this end, the resource requests are described as labeled graphs, which must be matched with equivalent labeled graphs of available resources. The coallocation problem described in the paper has real-world requirements and inputs that differ from those of a classical graph matching problem. We propose a new algorithm to solve the coallocation problem. The algorithm is especially tailored for medium to large grid systems, and is currently being integrated into the QosCosGrid system's allocation module.

1 Introduction

The problem we are tackling is a maximal allocation of a labeled *requests* graph to a labeled *offers*[1] graph. This problem is also referred as graph matching. The allocation must satisfy both the constraints of the nodes (machines) and the constraints of the links (network). In our setup, the allocation can be nonoptimal in terms of the allocation size; however, all allocations must obey all computing and network constraints.

The motivation for our work comes from real-world scientific applications, including complex systems simulations. Complex systems simulations include highly parallel applications such as large cellular automata; molecular dynamics simulations; combinations of coarse and fine-grained parallel applications, such as distributed evolutionary algorithms for optimizing parameters; techniques such as parallel tempering, where molecular dynamics simulations are combined with Monte Carlo algorithms; and agent-based models where both the frequency of communication between agents and the number of agents is highly variable and may change with time [1]. Such applications rely on the *coallocation* of large numbers of reliable resources. This requirement has traditionally been met by supercomputing facilities, but some applications researchers are now looking to computational grids as a more economic computing resource.

[1] In this paper we use the terms "offers" and "available machines" interchangeably, assuming that only available machines are offered by resource providers.

M. Bubak et al. (Eds.): ICCS 2008, Part I, LNCS 5101, pp. 274–283, 2008.
© Springer-Verlag Berlin Heidelberg 2008

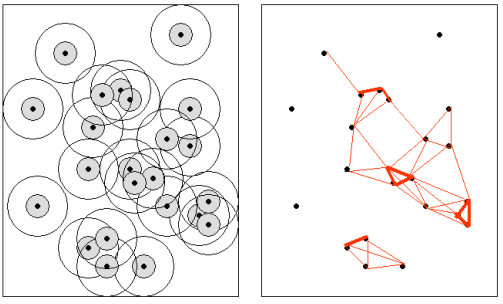

Fig. 1. A parallelized agent-based model (left) with the agent interactions represented as a graph (right)

Quasi-opportunistic supercomputing is a new approach, designed to enable the execution of demanding parallel applications on massive nondedicated resources in grid environments [2].

Fig. 1 shows an approach for parallelizing an agent-based model in which each agent interacts with others within a certain distance and must be aware of those agents that are within that distance. In Fig. 1, on the left, each black dot represents an agent, the light gray circle represents the distance for definite interactions, and the outer circle indicates possible future interactions. These interactions can be represented by a graph, as shown on the right, and it is this graph which depicts the properties and the topology of the required resources.

The matching methods in the literature can be divided into two broad categories: the first contains exact matching methods that require a strict correspondence among the two objects being matched or at least among their subparts. Algorithms that solve these problems for the general graphs are exponential in the worst case. The second category defines inexact matching methods, where a matching can occur even if the two graphs being compared are structurally different, relaxing to some extent the given constraints [3]. Our case can be seen as a mixture of both categories: as in exact matching, we must not violate any of the constraints, but as in inexact matching, nonoptimal allocation sizes are permissible. Forgoing this optimality requirement allows us to provide an efficient algorithm for resource coallocation, which in practice delivers results that are reasonably close to the optimum.

The problem described above can be very hard to solve even with heuristic algorithms in real grids. As real-world grids may consist of thousands of machines, and we are planning to simultaneously allocate hundreds to thousands of jobs, the number of edges in the offers and requests graphs might be of an order of 10^6. Thus, even light heuristic algorithms which are linear in the product of the number of edges are almost useless when dealing with the computation time of $O(10^{12})$.

To reduce the problem complexity, we propose a simplified version, which we call the *clustered topology-aware coallocation problem* (CTAAP). In this problem, the offered machines are aggregated into a relatively small number of

homogeneous clusters. Each cluster contains identical machines, interconnected by identical links. This formulation does not account for the differences between the machines in the clusters, but significantly reduces the problem size. Unfortunately, even the reduced problem is still NP-complete with no approximation available.

In this paper we propose a new heuristic algorithm, the CTAAP-Solver, which solves the CTAAP problem. In our solution, we execute *graduated assignment graph matching* [4] once, and use its output as a starting point for the greedy search procedure. During this greedy search, we repeatedly execute an algorithm for weighted bipartite graph matching, steering it towards a feasible solution that does not violate any constraint.

This paper is organized as follows. Related work is summarized in section 2. In section 3 we discuss the problem definition and its intractability. In this section we also formalize the problem and give the details of our CTAAP-Solver algorithm and its complexity. Experimental results are given in section 4.

2 Related Work

Exact matching. Most of the algorithms for exact graph matching are based on some form of tree search with backtracking. The first important algorithm of this family was by Ullmann [5] in 1976. Ullmann's algorithm is widely known and, despite its age, is still widely used and is probably the most popular graph matching algorithm. A more recent algorithm for both isomorphism and subgraph isomorphism is the VF algorithm, by Cordella et al. [6]. The authors define a heuristic that is based on the analysis of the sets of nodes adjacent to the ones already considered in the partial mapping. This heuristic computes quickly, leading to a significant improvement over Ullmann's and other algorithms in many cases. However, the worst case run time of Ullmann's algorithm is $\Theta(N!N^3)$, and $\Theta(N!N)$ for the VF algorithm.

Inexact matching. Tree search with backtracking can also be used for inexact matching. In [7], the A^* algorithm is used with a fast and simple heuristic that takes into account only the future cost of unmatched nodes. A radically different approach is to cast graph matching, which is inherently a discrete optimization problem, into a continuous, nonlinear optimization problem. One of the pioneering works for this approach is that of Fischler and Elschlager [8]. In [9], a new matching algorithm based on a probabilistic relaxation framework is proposed, which introduces the definition of a Bayesian graph edit distance. Gold and Rangarajan [4] presented the graduated assignment graph matching (GAGM) algorithm. In this algorithm a technique known as graduated nonconvexity is employed to avoid poor local optima [3]. However, the inexact matching algorithms that we are aware of do not guarantee that constraints will not be violated.

3 The Topology-Aware Coallocation Algorithm

3.1 Topology-Aware Coallocation: Definition and Analysis

Our coallocation model assumes an "à la operating systems" scheduling system, meaning that the time axis is divided into discrete (potentially inconstant and long) time slots, and the decision about which processes to execute in a certain time slot is made repeatedly by an allocation management system. In this paper, we consider only a certain time slot where the quantitative values of computing resources are assumed to be constant. Thus we ignore the time index in the following discussion.

The mathematical model will be defined in the next subsection, while here we will discuss the intractability of the presented problem. Matching that does not account for link constraints – known also as bipartite graph matching – is a well-studied problem [10], with a variety of efficient (polynomial) solving algorithms [11], [12]. However, matching that takes into account the links between the nodes, which is a general graph-matching problem, becomes NP-hard. Even the simplified form of the problem defined above – the CTAAP, where offered computing machines are grouped into homogeneous clusters, is still NP-complete with no approximation even for a constant number of clusters. This can be shown by reducing the *independent set* (IS) problem to the CTAAP. IS is defined as follows:

- Input: Graph $G = (V, E)$ and a positive integer k.
- Question: Is there a subset $V' \subseteq V$ of size k such that no two vertexes in V' are joined by an edge in E?

The reduction is as follows: given a graph G of n vertexes, we will treat it as a requests graph. We will create an offers graph with two clusters, one of size k with no links between the nodes, and another cluster with $n - k$ nodes, all of which are interconnected and also connected to all the nodes of the first cluster. There is a solution of CTAAP that can allocate all the requests to offers iff there is a solution to the IS problem. Not only is the IS NP-complete [13] but, as was shown by [14], no polynomial time algorithm can approximate it within a factor of $n/2^{(\log n)^{1-\epsilon}}$ for any $\epsilon > 0$, unless $NP = ZPP$. Given that fact, it is clear that CTAAP cannot be solved even approximately by polynomial time algorithms.

3.2 Clustered Topology-Aware Coallocation: Model Formalization

Specifying the topology request. The *request* for n tasks is presented as a graph $G_R = (V, E)$, where $|V| = n$ and v_i denotes a request for a single resource (machine).

A positive vector $C = [c_1, c_2, \ldots, c_n]$ represents request properties, where c_i denotes the minimal quantitative properties for the required computational resource v_i (e.g. FLOPS). Different quantitative properties might be described by multiple property vectors. For example, if vector C represents the minimal

CPU requirements, then the minimal memory requirements of the n tasks are represented by the positive vector $M = [m_1, m_2, \ldots, m_n]$.

The properties of edges $e \in E$ are represented by an n-by-n adjacency matrix B, where b_{ij} refers to the connectivity level between a user's tasks v_i and v_j. Usually, this matrix is symmetric, and $\forall i : 1 \leq i \leq n, b_{ii} = 0$. Matrix B represents the communication bandwidth between tasks as estimated by the user.

Specifying the resource offer. Analogously, an offer of m *clusters* of identical machines is denoted as a graph $\hat{G_O} = (\hat{V}, \hat{E})$,where $|\hat{V}| = m$.

The individual properties of the identical machines in the clusters are denoted as $\hat{C} = [\hat{c_1}, \hat{c_2}, \ldots, \hat{c_m}]$ (CPU) and $\hat{M} = [\hat{m_1}, \hat{m_2}, \ldots, \hat{m_m}]$ (memory).

A capacity vector $\hat{CAP} = [\hat{cap}_1, \hat{cap}_2, \ldots, \hat{cap}_m]$ denotes the number of available machines in each cluster; an m-by-m adjacency matrix $\hat{B}$ represents the edges' properties (e.g., currently available communication bandwidth within and between the m clusters in the grid), assuming identical connectivity properties between all the machines in each cluster.

In the offers graph there are usually self-loops. The self-loop of the node $\hat{v_i}$ denotes the connectivity level between the machines in cluster i: $\hat{b_{ii}} \neq 0$. The bandwidth could be estimated between two adjacent (physically connected) clusters or between two distant but connected clusters using maximum flow techniques.

The allocation matrix. We are interested in finding the n-by-m allocation matrix X, in which the term $x_{ij} = 1$ represents an allocation of a requested task v_i to an offered resource $\hat{v_j}$. Several constraints must hold for a correct coallocation:

$$\forall i : 1 \leq i \leq n, \sum_{j=1}^{m} x_{ij} \leq 1, \tag{1}$$

denoting that one requested task can be mapped to at most one offered resource;

$$\forall j : 1 \leq j \leq m, \sum_{i=1}^{n} x_{ij} \leq \hat{cap}_j, \tag{2}$$

denoting that an offered cluster j can serve at most $\hat{cap}_j$ tasks;

$$\forall i, j : 1 \leq i \leq n, 1 \leq j \leq m, x_{ij} c_i \leq \hat{c_j} \wedge x_{ij} m_i \leq \hat{m_j} \tag{3}$$

denoting that the individual (computation/memory) properties of a request must fit the properties of the matched offer;

$$\forall i, j, k, l : 1 \leq j, l \leq m, 1 \leq i, k \leq n, x_{ij} b_{ik} x_{kl} \leq \hat{b_{jl}}, \tag{4}$$

denoting that the pairwise (connectivity) properties of any pair of requests must fit the properties of the matched offers pair; and

$$\forall i, j : 1 \leq i \leq n, 1 \leq j \leq m, x_{ij} \in \{0, 1\}, \tag{5}$$

denoting that the decision is binary, where 1 indicates an allocation of a requested task v_i to an offered resource $\hat{v_j}$.

Different objective functions will express different "global welfare" schemas. Here we are interested in maximizing the system utilization: $\max(\sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij})$.

3.3 An Algorithm for Clustered Topology-Aware Coallocation

Our algorithm consists of three procedures. The first one finds a weights matrix X by executing a modified version of the graduated assignment graph matching [4] algorithm. Matrix X contains values between 0 and 1, which denote the "profitability" of each allocation X_{ij}. An extra row and column are added to hold the slack variables (this augmented matrix is denoted by $\widetilde{X}$). By incorporating slack variables, the graph matching algorithm can handle outliers (spurious or missing nodes or links) in a statistically robust manner. As β is constantly increased, only one number in each row and up to $c\hat{a}p_j$ numbers in each column approach 1, while all the others approach 0.

Input: vectors $C, M, \hat{C}, \hat{M}, C\hat{A}P$, matrixes $B, \hat{B}$, edge compatibility function $F(e_r, e_o) \rightarrow \mathbb{R} \mid e_r \in G_R, e_o \in G_O$
Output: weights matrix X
$\beta \leftarrow \beta_0$, $\widetilde{X} \leftarrow 1 + \epsilon$;
while $\beta \leq \beta_f$ **do**
  while X *does not converge AND #iterations* $\leq I_0$ **do**
    $Q_{ij} \leftarrow (c_i > \hat{c_j} \vee m_i > \hat{m_j})$? $0 : \sum_{k=1}^{N}\sum_{l=1}^{M} X_{kl}F(e_{ik}, e_{jl})$;
    $X_{ij} \leftarrow \exp(\beta Q_{ij})$;
    while $\widetilde{X}$ *does not converge AND #iterations* $\leq I_1$ **do**
      $\widetilde{X}^1_{ij} \leftarrow \frac{\widetilde{X}_{ij}}{\sum_{j=1}^{M+1}\widetilde{X}_{ij}}$; *// update* $\widetilde{X}$ *by normalizing across rows*
      $\widetilde{X}_{ij} \leftarrow \min(1, \frac{\widetilde{X}^1_{ij}}{\sum_{i=1}^{N+1-(c\hat{a}p_j-1)} smallest_i})$; *// normalizing across columns,* $smallest_i$ *stands for* i_{th} *smallest element in column* j
    end
  end
  $\beta \leftarrow \beta \cdot \beta_r$;
end
return X ;

Algorithm 1. Step 1 – inexact graph matching

In the second procedure we address equations 1-3 and 5 only (i.e., computational and capacity constraints). Discarding equation 4, we have an instance of a weighted bipartite matching problem, modeled as follows: $G' = (V', E')$, where $V' = V \cup \hat{V}$, and $E' = (v_i, \hat{v_j}) | v_i \in V \wedge \hat{v_j} \in \hat{V} \wedge c_i \leq \hat{c_j} \wedge m_i \leq \hat{m_j}$, where we use weights computed by procedure 1: $w((v_i, \hat{v_j})) = X_{ij}$. To solve the

maximum weighted bipartite problem, we use a slightly modified version of the LEDA implementation [15]. The resulting allocation suits constraints 1-3 and 5 but might violate constraint 4.

Input: vectors $C, M, \hat{C}, \hat{M}, C\hat{A}P$, weights matrix X
Output: allocation matrix A
$A \leftarrow$ solve maximum_weighted_bipartite_matching$(C, M, X, \hat{C}, \hat{M}, C\hat{A}P)$;
return A;

Algorithm 2. Step 2 – weighted bipartite graph matching

In the last procedure, we have to make sure that no connectivity constraints were violated by procedure 2. To do so, we analyze all the allocation pairs (X_{ij} and X_{kl}), counting how many connectivity violating allocation pairs each allocation X_{ij} appears in. If no connectivity violations were detected, the algorithm terminates. Otherwise, the "worst" allocation X_{ij} (the one that appears in the most violating pairs) is removed from the allocation matrix, from then on forcing $X_{ij} = 0$, and step 2 is repeated.

Input: allocation matrix A, vectors $C, M, \hat{C}, \hat{M}, C\hat{A}P$ and matrixes $B, \hat{B}$
Output: final allocation matrix
$problem\_cost_{ij} \leftarrow ZERO\_MATRIX$;
$curr\_alloc \leftarrow \{(i,j) | i \in \{1..n\} \wedge j \in \{1..m\} \wedge A_{ij} = 1\}$;
forall $(i,j) | (i,j) \in curr\_alloc$ **do**
  forall $(k,l) | (k,l) \in curr\_alloc \wedge (k,l) \neq (i,j)$ **do**
    if $B_{ik} > \hat{B}_{jl}$ **then** *// Allocation violates edge constraints*
      $problem\_cost_{ij}$++ ; $problem\_cost_{kl}$++ ;
    end
  end
end
if *(problem_cost is ZERO_MATRIX)* **then**
  return A ;
else
  $(i,j) \leftarrow$ index of the biggest number in $problem\_cost$;
  $A_{ij} \leftarrow 0$;
  Go To Procedure 2 ;
end

Algorithm 3. Step 3 – cleanup

3.4 Algorithm Complexity

In order to analyze the complexity of the entire algorithm, we will analyze each one of its three steps. Here we will assume that the number of clusters in the grid is M and the number of jobs is N.

Step 1: the normalization across rows takes $O(N^2M^2)$, while the normalization across columns takes $O(N^2M^2\log(N))$. The overall complexity of step 1 is $O(N^2M^2\log(N))$.

Step 2: the constructed graph G' has $O(N+M)$ vertexes and $O(NM)$ edges. Using the algorithm proposed in [15], the overall complexity of this step is $O((N+M)^2\log(N+M)+NM(N+M)) = O(N^2\log(N+M)+N^2M)$, assuming that $N \gg M$.

Step 3: as there is a maximal number of N allocations, the analysis of the correctness of all given allocation pairs takes $O(N^2)$ time.

Overall: In the worst case, steps 2 and 3 are repeated NM times; thus the overall time complexity in the worst case is $O(N^3M\log(N+M)+N^3M^2)$. However, we expect the average performance to be much better. It is also important to note that the algorithm is polynomial in the number of clusters only, regardless of the number of actual machines in those clusters.

4 Experimental Results

In order to evaluate the performance of the CTAAP-Solver algorithm, we performed a series of experiments to estimate both its quality and speed. The results for the CTAAP-Solver algorithm are compared to the optimal results calculated by an integer programming technique. The integer programming model that we used is based on the five equations listed above. The fourth quadratic equation was replaced by a series of equivalent linear equations. The system of five equations, including the modified equation 4, was fed into integer-programming solvers GLPK [16] and CBC [17], which provided an optimal solution. The following values for the constants were used in all the experiments: $\beta_0 = 0.5, \beta_f = 10, \beta_r = 1.075, I_0 = 4$, and $I_1 = 30$.

Fig. 2 describes the results of the first experiment, in which the CTAAP-Solver algorithm results are compared with the optimal solution. The request graph of 50 nodes has computing and network properties as random values in the range of 1...100 (all the random numbers mentioned in this text have uniform distribution in the given range). The offered graph consisted of 5 homogeneous clusters, each with a random capacity in the range of 1...11. In five consistent tests composed of 100 independent runs, we increased the offered properties ranges from 1...100 to 1...200, then to 1...300, etc.

The results depicted in Fig. 2 show that as the chances of a single request to be mapped to a single available resource increase, our algorithm performs better. The "range" itself is of no importance: a request can be mapped to an offer iff the offer's properties are not lower than the request's properties. Only the order of the requests and offers is important, and not the values themselves. In the first point of the graph, the chances of a single requested machine to be mapped to a specific available machine are 50.5% (both offer and request are integers, randomized in a range of 1...100). In the second test, these chances increase

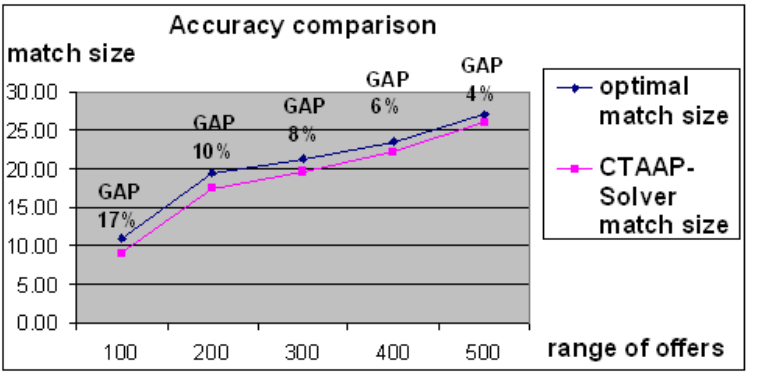

Fig. 2. The success rate of the CTAAP-Solver algorithm

(as the offer is randomized in the range of 1...200, but the request is still randomized in the range of 1...100) and thus becomes $1/2+1/2\cdot 0.505 \approx 75\%$, while in the third experiment it is $2/3+1/3\cdot 0.505 \approx 83\%$, and so on.

Another experiment, the results of which are given in Fig. 3, compares the runtime of the CTAAP-Solver algorithm with the runtime of one of the best open-source integer programming solvers – CBC2 [17].

In this experiment the size of the requests and offers graphs was constantly increased. The computing and network properties were random values in the range of 1...100. The offers graph consisted of 5 homogeneous clusters, with a random capacity in the range of 1 and 1/5 of the size of the requests graph. The computing and network properties were random numbers in the range of 1...200.

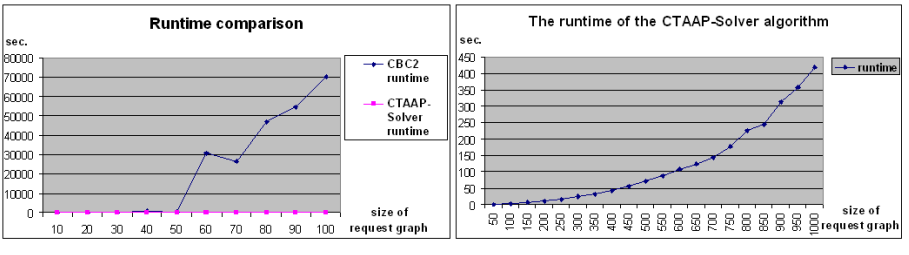

Fig. 3. Left: a comparison of the runtime of the CTAAP-Solver algorithm vs. CBC2 optimized exhaustive search. Right: the runtime of the CTAAP-Solver algorithm.

5 Conclusions

Here we have presented a new algorithm that provides a fast and efficient solution to the topology-aware coallocation problem. This algorithm is currently used as an important building block in the QosCosGrid scheduling system.

Acknowledgments. The work in this paper was supported by EC grant QosCosGrid IST FP6 STREP 033883.

References

1. Charlot, M., et al.: The QosCosGrid project: Quasi-Opportunistic Supercomputing for Complex Systems Simulations. In: Description of a general framework from different types of applications. Ibergrid conference, Centro de Supercomputacion de Galicia (GESGA) (2007)
2. Kravtsov, V., Carmeli, D., Schuster, A., Yoshpa, B., Silberstein, M., Dubitzky, W.: Quasi-Opportunistic Supercomputing in Grids, Hot Topic Paper. In: IEEE International Symposium on High Performance Distributed Computing, Monterey Bay, California, USA (2007)
3. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty Years of Graph Matching in Pattern Recognition. International Journal of Pattern Recognition and Artificial Intelligence 18(3), 265–298 (2004)
4. Gold, S., Rangarajan, A.: A Graduated Assignment Algorithm for Graph Matching. IEEE Trans. Pattern Anal. Mach. Intell. 18(4), 377–388 (1996)
5. Ullman, J.R.: An algorithm for subgraph isomorphism. J. Assoc. Comput. Mach. 23, 31–42 (1976)
6. Cordella, L.P., Foggia, P., Sansone, C., Tortorella, F., Vento, M.: Graph matching: a fast algorithm and its evaluation. In: 14th Int. Conf. Pattern Recognition, pp. 1582–1584 (1998)
7. Gregory, L., Kittler, J.: Using graph search techniques for contextual colour retrieval. In: Joint IAPR Int. Workshops SSPR and SPR, pp. 186–194 (2002)
8. Fischler, M., Elschlager, R.: The representation and matching of pictorial structures. IEEE Trans. Computing 22, 67–92 (1973)
9. Myers, R., Wilson, R.C., Hancock, E.R.: Bayesian graph edit distance. IEEE Trans. Patt. Anal. Mach. Intell. 22, 628–635 (2000)
10. Lovasz, L., Plummer, M.D.: Matching Theory. Elsevier Science Publishing Company, New York (1986)
11. Blum, N.: A Simplified Realization of the Hopcroft-Karp Approach to Maximum Matching in General Graphs. Univ. of Bonn, Computer Science V, 85232-CS (2001)
12. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms, The Bellman-Ford algorithm, pp. 588–592. MIT Press and McGraw-Hill, New York, USA (2001)
13. Pardalos, P.M., Xue, J.: The maximum clique problem. Journal of Global Optimization 4(3), 301–328 (1994)
14. Khot, S.: Improved Inapproximability Results for MaxClique, Chromatic Number and Approximate Graph Coloring. In: Proceedings of the 42nd IEEE Symposium on the Foundations of Computer Science 600, Washington, DC, USA (2001)
15. Mehlhorn, K., Naeher, S.: The LEDA Platform of Combinatorial and Geometric Computing. Cambridge University Press, Cambridge (1999)
16. Andrew, M.: GNU Linear Programming Kit, Version 4.22. `http://www.gnu.org/software/glpk/glpk.html`
17. Bonami, P., et al.: Research Report RC 23771, An Algorithmic Framework For Convex Mixed Integer Nonlinear Programs. IBM T. J. Watson Research Center, Yorktown, USA (2005)