within 10% of the bounds, as is, to be feasible.

within the domain of the solution, the objective function is minimized by the feasible solution. In this case, the objective function is shown to be approximately 10% of the maximum possible value for the objective function. The objective function is defined as

\[ \min \{ f(x) \} \]

subject to

\[ g(x) \leq 0 \]

where \( f(x) \) is the objective function and \( g(x) \) are the constraint functions.

The objective function is shown to be approximately 10% of the maximum possible value for the objective function. The objective function is defined as

\[ \min \{ f(x) \} \]

subject to

\[ g(x) \leq 0 \]

where \( f(x) \) is the objective function and \( g(x) \) are the constraint functions.

The objective function is shown to be approximately 10% of the maximum possible value for the objective function. The objective function is defined as

\[ \min \{ f(x) \} \]

subject to

\[ g(x) \leq 0 \]

where \( f(x) \) is the objective function and \( g(x) \) are the constraint functions.

The objective function is shown to be approximately 10% of the maximum possible value for the objective function. The objective function is defined as

\[ \min \{ f(x) \} \]

subject to

\[ g(x) \leq 0 \]

where \( f(x) \) is the objective function and \( g(x) \) are the constraint functions.

The objective function is shown to be approximately 10% of the maximum possible value for the objective function. The objective function is defined as

\[ \min \{ f(x) \} \]

subject to

\[ g(x) \leq 0 \]

where \( f(x) \) is the objective function and \( g(x) \) are the constraint functions.

The objective function is shown to be approximately 10% of the maximum possible value for the objective function. The objective function is defined as

\[ \min \{ f(x) \} \]

subject to

\[ g(x) \leq 0 \]

where \( f(x) \) is the objective function and \( g(x) \) are the constraint functions.

The objective function is shown to be approximately 10% of the maximum possible value for the objective function. The objective function is defined as

\[ \min \{ f(x) \} \]

subject to

\[ g(x) \leq 0 \]

where \( f(x) \) is the objective function and \( g(x) \) are the constraint functions.

The objective function is shown to be approximately 10% of the maximum possible value for the objective function. The objective function is defined as

\[ \min \{ f(x) \} \]

subject to

\[ g(x) \leq 0 \]

where \( f(x) \) is the objective function and \( g(x) \) are the constraint functions.

The objective function is shown to be approximately 10% of the maximum possible value for the objective function. The objective function is defined as

\[ \min \{ f(x) \} \]

subject to

\[ g(x) \leq 0 \]

where \( f(x) \) is the objective function and \( g(x) \) are the constraint functions.
Table 4.1  
Results on 42 input-output matrices

<table>
<thead>
<tr>
<th>Problem</th>
<th>Initial m</th>
<th>Cuts added</th>
<th>Stages</th>
<th>Final m</th>
<th>Iterations</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t59d11</td>
<td>67</td>
<td>340</td>
<td>14</td>
<td>318</td>
<td>85</td>
<td>17.47</td>
</tr>
<tr>
<td>t59fl1</td>
<td>76</td>
<td>232</td>
<td>11</td>
<td>247</td>
<td>64</td>
<td>11.71</td>
</tr>
<tr>
<td>t59h11</td>
<td>80</td>
<td>224</td>
<td>15</td>
<td>258</td>
<td>77</td>
<td>13.42</td>
</tr>
<tr>
<td>t59n11</td>
<td>73</td>
<td>374</td>
<td>16</td>
<td>311</td>
<td>78</td>
<td>15.23</td>
</tr>
<tr>
<td>t65b11</td>
<td>93</td>
<td>358</td>
<td>20</td>
<td>342</td>
<td>113</td>
<td>23.65</td>
</tr>
<tr>
<td>t65d11</td>
<td>105</td>
<td>297</td>
<td>18</td>
<td>302</td>
<td>85</td>
<td>16.36</td>
</tr>
<tr>
<td>t65fl1</td>
<td>95</td>
<td>353</td>
<td>25</td>
<td>359</td>
<td>124</td>
<td>24.36</td>
</tr>
<tr>
<td>t65h11</td>
<td>97</td>
<td>389</td>
<td>16</td>
<td>406</td>
<td>98</td>
<td>20.59</td>
</tr>
<tr>
<td>t65l11</td>
<td>77</td>
<td>174</td>
<td>10</td>
<td>192</td>
<td>56</td>
<td>9.50</td>
</tr>
<tr>
<td>t65n11</td>
<td>104</td>
<td>412</td>
<td>17</td>
<td>420</td>
<td>94</td>
<td>21.58</td>
</tr>
<tr>
<td>t65w11</td>
<td>101</td>
<td>324</td>
<td>14</td>
<td>340</td>
<td>69</td>
<td>13.48</td>
</tr>
<tr>
<td>t69e11</td>
<td>75</td>
<td>345</td>
<td>19</td>
<td>327</td>
<td>109</td>
<td>21.82</td>
</tr>
<tr>
<td>t70b11</td>
<td>86</td>
<td>275</td>
<td>12</td>
<td>286</td>
<td>80</td>
<td>15.40</td>
</tr>
<tr>
<td>t70d11b</td>
<td>132</td>
<td>363</td>
<td>11</td>
<td>404</td>
<td>66</td>
<td>14.38</td>
</tr>
<tr>
<td>t70f11</td>
<td>104</td>
<td>337</td>
<td>14</td>
<td>387</td>
<td>68</td>
<td>11.92</td>
</tr>
<tr>
<td>t70h11</td>
<td>85</td>
<td>426</td>
<td>19</td>
<td>414</td>
<td>111</td>
<td>24.02</td>
</tr>
<tr>
<td>t70k11</td>
<td>73</td>
<td>345</td>
<td>16</td>
<td>362</td>
<td>90</td>
<td>18.56</td>
</tr>
<tr>
<td>t70l11</td>
<td>75</td>
<td>264</td>
<td>14</td>
<td>239</td>
<td>69</td>
<td>12.33</td>
</tr>
<tr>
<td>t70n11</td>
<td>94</td>
<td>346</td>
<td>20</td>
<td>353</td>
<td>98</td>
<td>20.12</td>
</tr>
<tr>
<td>t70u11</td>
<td>63</td>
<td>173</td>
<td>11</td>
<td>185</td>
<td>56</td>
<td>6.47</td>
</tr>
<tr>
<td>t70w11</td>
<td>110</td>
<td>240</td>
<td>12</td>
<td>303</td>
<td>62</td>
<td>11.39</td>
</tr>
<tr>
<td>t70x11</td>
<td>105</td>
<td>282</td>
<td>13</td>
<td>325</td>
<td>66</td>
<td>10.01</td>
</tr>
<tr>
<td>t74d11</td>
<td>125</td>
<td>319</td>
<td>13</td>
<td>369</td>
<td>71</td>
<td>14.71</td>
</tr>
<tr>
<td>t75d11</td>
<td>124</td>
<td>351</td>
<td>14</td>
<td>398</td>
<td>72</td>
<td>14.92</td>
</tr>
<tr>
<td>t75e11</td>
<td>100</td>
<td>442</td>
<td>16</td>
<td>411</td>
<td>99</td>
<td>19.66</td>
</tr>
<tr>
<td>t75f11</td>
<td>88</td>
<td>418</td>
<td>17</td>
<td>428</td>
<td>102</td>
<td>22.28</td>
</tr>
<tr>
<td>t75h11</td>
<td>81</td>
<td>295</td>
<td>12</td>
<td>320</td>
<td>69</td>
<td>13.38</td>
</tr>
<tr>
<td>t75l11</td>
<td>86</td>
<td>352</td>
<td>18</td>
<td>345</td>
<td>101</td>
<td>21.48</td>
</tr>
<tr>
<td>t75n11</td>
<td>92</td>
<td>254</td>
<td>16</td>
<td>286</td>
<td>73</td>
<td>13.46</td>
</tr>
<tr>
<td>Mean</td>
<td>92</td>
<td>321</td>
<td>15</td>
<td>332</td>
<td>83</td>
<td>16.33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem</th>
<th>Initial m</th>
<th>Cuts added</th>
<th>Stages</th>
<th>Final m</th>
<th>Iterations</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>diw56n54</td>
<td>210</td>
<td>721</td>
<td>23</td>
<td>683</td>
<td>119</td>
<td>53.92</td>
</tr>
<tr>
<td>diw56n58</td>
<td>206</td>
<td>625</td>
<td>16</td>
<td>636</td>
<td>80</td>
<td>34.33</td>
</tr>
<tr>
<td>diw56n62</td>
<td>205</td>
<td>876</td>
<td>21</td>
<td>776</td>
<td>112</td>
<td>61.21</td>
</tr>
<tr>
<td>diw56n66</td>
<td>204</td>
<td>817</td>
<td>16</td>
<td>729</td>
<td>89</td>
<td>42.98</td>
</tr>
<tr>
<td>diw56n67</td>
<td>211</td>
<td>809</td>
<td>17</td>
<td>768</td>
<td>113</td>
<td>63.13</td>
</tr>
<tr>
<td>diw56n72</td>
<td>213</td>
<td>710</td>
<td>16</td>
<td>716</td>
<td>90</td>
<td>42.23</td>
</tr>
<tr>
<td>diw56r54</td>
<td>211</td>
<td>723</td>
<td>22</td>
<td>706</td>
<td>100</td>
<td>37.87</td>
</tr>
<tr>
<td>diw56r58</td>
<td>207</td>
<td>632</td>
<td>19</td>
<td>626</td>
<td>83</td>
<td>35.55</td>
</tr>
<tr>
<td>diw56r66</td>
<td>206</td>
<td>755</td>
<td>15</td>
<td>703</td>
<td>81</td>
<td>37.50</td>
</tr>
<tr>
<td>diw56r67</td>
<td>217</td>
<td>816</td>
<td>18</td>
<td>759</td>
<td>94</td>
<td>48.11</td>
</tr>
<tr>
<td>diw56r72</td>
<td>206</td>
<td>731</td>
<td>17</td>
<td>711</td>
<td>91</td>
<td>43.84</td>
</tr>
<tr>
<td>Mean</td>
<td>209</td>
<td>747</td>
<td>18</td>
<td>710</td>
<td>96</td>
<td>45.52</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem</th>
<th>Initial m</th>
<th>Cuts added</th>
<th>Stages</th>
<th>Final m</th>
<th>Iterations</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbm1974</td>
<td>225</td>
<td>928</td>
<td>17</td>
<td>895</td>
<td>112</td>
<td>79.08</td>
</tr>
<tr>
<td>sbm1975</td>
<td>231</td>
<td>1034</td>
<td>19</td>
<td>965</td>
<td>114</td>
<td>92.97</td>
</tr>
<tr>
<td>Mean</td>
<td>228</td>
<td>981</td>
<td>18</td>
<td>930</td>
<td>113</td>
<td>86.03</td>
</tr>
</tbody>
</table>

A PRIMAL-DUAL INTERIOR POINT CUTTING PLANE METHOD FOR THE LINEAR ORDERING PROBLEM

JOHN E. MITCHELL* AND BRIAN BORCHERS

Abstract. A cutting plane algorithm for solving the linear ordering problem is described. This algorithm uses the primal-dual interior point method to solve the linear programming relaxations which methods has been attempted to identify cutting planes early and it stores an old feasible point which is used to help recenter when cutting planes are added. Computational results are described for some real-world problems; the algorithm appears to be somewhat competitive with a simplex-based cutting plane algorithm.

1. Introduction. The linear ordering problem is an \( NP \)-hard combinatorial optimization problem which has many real-world applications, including the triangulation of input-output matrices in economics. The polyhedral structure of the problem has been investigated by Grötschel, Jünger, and Reinelt [4,7,11]. They described a simplex-based cutting plane algorithm for solving the linear ordering problem and were able to solve sets of European Community and (West) German input-output tables. They showed that almost all of these problems could be solved using just simple bounds and 3-dicycle inequalities.

We used the primal-dual logarithmic barrier method as described by Lustig, Marsten and Shanno [9]. This is a barrier function method which works on the primal and dual linear programs simultaneously, applying Newton's method to the sum of the objective function and the barrier function. All parameters were as described in [9], except as detailed in Section 3.

It is necessary to be able to exploit a "warm start" in the interior point method if it is to be used efficiently in a cutting plane algorithm. Usually, the solution to one relaxation is on the boundary of the feasible region, and interior point methods generally do not perform well when started from close to the boundary. We used two strategies to overcome this problem. The first was to attempt to identify cutting planes early, so the iterates do not get too close to the boundary. The second was to store a strictly feasible interior point from which the algorithm was restarted when cutting planes were added. As can be seen from the computational results in section 4, the method appeared to be able to efficiently exploit warm starts.

Computational work on solving integer programming problems using interior point methods has been done by Borchers [2], Kaliski and Ye [8], Mitchell and Todd [10], and Resende and Veiga [12], among others. For interesting theoretical results, see the papers cited above and also Goffin and Vial [3] and den Hertog et al. [5], which discuss the related problem of using an interior point method in a column generation setting.

2. The Linear Ordering Problem. The linear ordering problem can be stated as follows:

Given a set of objects \( V \), and rewards \( \delta_{ij} \) and \( \delta_{ji} \) for each pair of objects \( i, j \) in \( V \), find a complete ordering for the objects which max-

* Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180 USA – mitchell@rpi.edu. Research partially supported by ONR Grant number N00014-96-J-1714.

† Department of Mathematics, New Mexico Tech, Socorro, NM 87801 USA – bercher@nmt.edu
projections was about 20%, 40% and 50% of the total run time, increasing as the size of the problems increased.

Acknowledgement. We are very grateful to Michael Jünger for supplying us with the input-output matrices analyzed in this paper. We are also grateful to Eberhard Kranich for his bibliography on interior point methods.

REFERENCES


3. Parallel Best First Search. The A* algorithm is a well known search algorithm that can use problem-specific heuristic information to prune search space. As discussed in [10], A* is essentially a "best-first" branch-and-bound (B&B) algorithm. A number of researchers have investigated parallel formulations of A*/B&B algorithms [14,15]. An important component of A*/B&B algorithms is the priority queue which is used to maintain the "frontier" (i.e., unexplored) nodes of the search graph in the heuristic order. In the sequential A*/B&B algorithm, in each cycle a most promising node from the priority queue is removed and expanded, and the newly generated nodes are added to the priority queue.

In most parallel formulations of A*, different processors concurrently expand different frontier nodes. Conceptually, these formulations can be viewed to differ in the data structures used to implement the priority queue of the A* algorithm. Some formulations are suited only for shared-memory architectures, whereas others are suited for distributed-memory architectures as well. The effectiveness of different parallel formulations is also strongly dependent upon the characteristics of the problem being solved. We have investigated a number of parallel formulations of A*[15]. Some of these formulations are new, and the others are very similar to the ones developed by other researchers. We have tested the performance of these formulations on the 15-puzzle, the traveling salesman problem(TSP), and the vertex cover problem (VCP) on the BBN Butterfly(TM) multiprocessor. The results for the 15-puzzle and VCP are very similar, but very different from results obtained for the TSP. The reason is that the TSP and VCP generate search spaces that are qualitatively different from each other, even though both problems are NP-hard problems. We have also performed a preliminary analysis of the relationship between the characteristics of the search spaces and their suitability to various parallel formulations[15].

4. Speedup Anomalies in Parallel Search. In parallel DFS, the speedup can differ greatly from one execution to another, as the actual parts of the search space examined by different processors are determined dynamically, and can be different for different executions. Hence, for some execution sequences the parallel version may find a solution by visiting fewer nodes than the sequential version thereby giving superlinear speedup, whereas for others it may find a solution only after visiting more nodes resulting in sublinear speedup. It may appear that on the average the speedup would be either linear or sublinear. This phenomenon of speedup greater than P on P processors in isolated executions of parallel DFS has been reported by many researchers, for a variety of problems and is referred to by the term speedup anomaly. In [20], we present analytical models and experimental results on the average case behavior of parallel backtracking. We consider two types of backtrack search algorithms: (i) simple backtracking (which does not use any heuristic information); (ii) heuristic backtracking (which uses heuristics to order and prune search). We present analytical models to compare the average number of nodes visited in sequential and parallel search for each case. For simple backtracking, we show that the average speedup obtained is (i) linear when distribution of solutions is uniform and (ii) superlinear when distribution of solutions is non-uniform. For heuristic backtracking, the average speedup obtained is at least linear (i.e., either linear or superlinear), and the speedup obtained on a subset of instances (called difficult instances) is superlinear. We also present experimental results over many synthetic and practical problems on various parallel machines, that validate our theoretical analysis.
The problem of determining the maximum clique problem is an important problem in graph theory. The objective is to find the largest complete subgraph within a given graph. To solve this problem, various algorithms have been developed, including the Bron-Kerbosch algorithm, which is a recursive algorithm that explores all maximal cliques in a graph. The algorithm works by maintaining a list of candidates and a list of selected cliques. It recursively tries to extend a candidate clique by adding vertices that are adjacent to all vertices in the current candidate clique. This process is repeated until no more candidates can be extended, and the maximum clique is the one with the maximum size. The Bron-Kerbosch algorithm is efficient and can handle large graphs, but it still requires significant computational resources for dense graphs with thousands of vertices.
Then, \( A(n, w, d) \) is the size of a maximum clique of \( J(n, w, d) \). It is known (see [4]) that a code of size \( n \) and weight \( w \) can correct \( w - \frac{d}{2} \) errors. All known lower bounds and exact values for constant weighted codes with distance between 2 and 18 are given in [4]. The interested reader can refer to this paper for all details.

3. Problems Arising from Keller’s Conjecture. A family of hypercubes with disjoint interiors whose union is \( R^n \) is a tiling. Lattice tiling is a tiling for which the centers of the hypercubes form a lattice.

In the beginning of the century, Minkowski conjectured that in a lattice tiling of \( R^n \) by translates of a unit hypercube, there exist two cubes that share a \((n - 1)\)-dimensional face. About fifty years later, Hajós proved Minkowski’s conjecture. At 1930, Keller suggested that Minkowski’s conjecture holds even in the absence of the lattice assumption. Ten years later Perron proved correctness of Keller’s conjecture for \( n \leq 6 \). Since then, many efforts have been devoted to prove or disprove this conjecture. Recently, a combinatorial equivalent of the conjecture was given by Corrádi and Szabó [6] which states that: There is a counterexample to Keller’s conjecture if and only if there exists \( n \in N \) for which the graph \( \Gamma_n \) has a clique of size \( 2^n \). The class of Keller’s graphs \( \Gamma_n \) is described next.

3.1. Keller Graphs. Consider the set \( V_n = \{ (d_1, \ldots, d_n) : d_i \in \{0, 1, 2, 3\} \text{ for } 1 \leq i \leq n \} \) and define the graph \( \Gamma_n \) with vertex set \( V_n \) and edge set as follows: Given a graph \( G \), let \( (v, u) \in G \) denote that vertices \( v, u \) are adjacent in \( G \). Two nodes \( v = (d_1, \ldots, d_n) \) and \( u = (d'_1, \ldots, d'_n) \) in \( V_n \) are adjacent iff \( d_i - d'_i \equiv 2 \mod 4 \) for some \( 1 \leq i \leq n \) and \( d_i \neq d'_i \) for some \( 1 \leq j \leq n \) and \( i \neq j \).

Clearly, \( \Gamma_n \) has \( 4^n \) vertices. It is not hard to show that every vertex \( v \in V_n \) has degree \( d(v) = 4^n - 3^n - n \). Also, one can verify that if \( E(\Gamma_n) \) is the edge set of \( \Gamma_n \), then \( \Gamma_n \) has density \( \frac{\#E(\Gamma_n)}{\#(V_n) \cdot (\#V_n - 1)} = \frac{4^n - 3^n - n}{4^n - 1} \) where \( K_n \) denotes the complete graph on \( m \) vertices. A quick calculation shows that \( \Gamma_7 \) has density exceeding 0.666 and \( \Gamma_{15} \) has density over 0.8866! Recently it has been proved that for \( n \geq 10 \) the clique size of \( \Gamma_n \) is \( c(\Gamma_n) = 2^n \). This means that Keller’s conjecture fails for \( n \geq 10 \) but it remains open for \( n \in \{7, 8, 9\} \). The following table (see [6]) gives the maximum clique size of the graphs \( \Gamma_n \) for \( n \leq 6 \).

<table>
<thead>
<tr>
<th>( n )</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c(\Gamma_n) )</td>
<td>2</td>
<td>5</td>
<td>12</td>
<td>28</td>
<td>57</td>
</tr>
</tbody>
</table>

Values of \( c(\Gamma_n) \) for \( n \leq 6 \)

4. Problems Arising From Fault Diagnosis. Over the last years large multiprocessor systems have gained an essential role in computing. For this matter, reliability of these systems is very important and diagnosis of faulty machines becomes a problem. The classical approach on fault diagnosis originated by Preparata, Metze and Chien twenty years ago by a model known as PMC (see [3]). In this model it is assumed that each unit can test some other units and that fault-free units always detect correct results while faulty units can output any result. Also, it is assumed that the number of faulty units does not exceed a certain bound \( t \) (if cannot exceed the number of neighbors of any unit) and gathered results are transmitted and processed by a monitoring unit from which we ascertain the state of the system.

MIMD multiprocessor. In these parallel formulations, the search space is dynamically assigned among a number of processors. These formulations differ in the methods used for distributing work among processors. To study the effectiveness of these formulations, we have applied these to several of problems. Parallel formulations of IDA* for solving the 15 puzzle problem yielded near linear speedup on Sequent Balance up to 30 processors and on the Intel Hypercube 2™ and BBN Butterfly 2™ up to 128 processors [16,17,19]. Parallel formulation of PODEM, which is the best known sequential algorithm for solving the test pattern generation problem, provided linear speedup on a 128 processor Symult™ [5]. Parallel depth-first branch-and-bound for floorplan optimization for VLSI circuits yielded linear speedups on a 1024 processor Ncube 2™, a 128 processor Symult™ and a network of 16 SUN workstations [3]. Linear speedups were also obtained for parallel DFS for the tautology verification problem for up to 1024 processors on the Ncube/10/2™ and the Ncube/2/2™ [4,7,11]. In [8], we have presented new methods for load balancing of unstructured tree computations on large-scale SIMD machines such as CM-2™. The analysis and experiments show that our new load balancing methods provide good speedups for parallel DFS on SIMD architectures. In particular, their scalability is no worse than that of the best load balancing schemes on MIMD architectures.

Scalability Analysis. From experimental results for a particular architecture and a range of processors alone, it is difficult to ascertain the relative merits of different parallel algorithm-architecture combinations. This is because the performance of different schemes may be altered in different ways by changes in hardware characteristics (such as interconnection network, CPU speed, speed of communication channels etc.), number of processors, and the size of the problem instance being solved [12]. Hence any conclusions drawn from experimental results on a specific parallel computer and program instance are rendered invalid by changes in any one of the above parameters. Scalability analysis of a parallel algorithm and architecture combination has been shown to be useful in extrapolating these conclusions [13,17].

We have developed a scalability metric, called isoefficiency, which relates the problem size to the number of processors necessary for an increase in speedup in proportion to the number of processors used [17]. In general, for a fixed problem size \( W \), increasing the number of processors \( P \) causes a decrease in efficiency because parallel processing overhead will increase while the sum of time spent by all processors in meaningful computation will remain the same. Parallel systems that can maintain a fixed efficiency level while increasing both \( W \) and \( P \) are defined as scalable [13]. If \( W \) needs to grow as \( f(P) \) to maintain an efficiency \( E \), then \( f(P) \) is defined to be the isoefficiency function for efficiency \( E \) and the plot of \( f(P) \) with respect to \( P \) is defined to be the isoefficiency curve for efficiency \( E \). An important feature of isoefficiency analysis is that it succinctly captures the effects of characteristics of the parallel algorithm as well as the parallel architecture on which it is implemented, in a single expression. The isoefficiency metric has been found to be very useful in characterizing scalability of a number of algorithms [18].

Scalability Analysis of Parallel Depth-First Search. The primary reason for loss of efficiency in our parallel formulations of DFS is the communication over-

---

1 Our work on tautology verification, test pattern generation and floorplan optimization received honorable mention for the Gordon Bell Award for outstanding research in practical parallel computing [6]. This is significant considering that this award has historically been given to researchers working on numerical problems.

2 CM-2 is a registered trademark of the Thinking Machines Corporation.
In the context of optimization problems, parallel algorithms are often utilized to enhance the efficiency and speed of solving complex problems, particularly those involving large datasets or intricate search spaces. This paper aims to explore and analyze the efficacy of parallel algorithms in addressing specific optimization challenges.


table

1. **Introduction**
   - Description of the problem and motivation.

2. **Related Work**
   - Overview of previous research and existing parallelization techniques.

3. **Proposed Methodology**
   - Detailed explanation of the parallel algorithm.

4. **Experiments and Results**
   - Presenting case studies and performance metrics.

5. **Conclusion**
   - Summary of findings and future research directions.

This structured approach will allow for a comprehensive understanding of the parallel algorithms and their impact on optimization problems.
6. Remarks. Test problems are of practical importance in evaluating algorithms or heuristics and performing computational comparisons. In this note we presented a number of test instances that have interesting applications. Computational results, as well as the codes of the test generators that we describe here, are presented in detail in [7].

REFERENCES


in practice, lower bounds do not turn out as sharp as they could potentially be (for instance, at least 2 out of the 3 test problems for which we report a positive duality gap, could be solved to optimality, at node 0 of a search tree, with a branch and cut algorithm that uses the same inequalities as we do). The main attraction here is that our lower bounds are still very sharp and the time to compute them is, relatively speaking, low. One factor that helps explain this is the low computational complexity associated with the detection of violated inequalities. This may prove even more attractive for applications where the separation problem, for a given type of valid inequality, in an equivalent branch and cut algorithm, could only be tackled, due to their computational complexity, by approximation algorithms.

REFERENCES

### Table 1: Example Data

<table>
<thead>
<tr>
<th>Year</th>
<th>Sales</th>
<th>Expenses</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020</td>
<td>1200</td>
<td>900</td>
<td>300</td>
</tr>
<tr>
<td>2021</td>
<td>1300</td>
<td>1000</td>
<td>300</td>
</tr>
<tr>
<td>2022</td>
<td>1400</td>
<td>1100</td>
<td>300</td>
</tr>
</tbody>
</table>

### Figure 2: Flowchart Diagram

```
start
  decision x
    branch x = 0
      decision y
        branch y = 1
          action z
            decision w
              branch w = 0
                action x
              branch w = 1
                action y
            decision v
              branch v = 0
                action w
              branch v = 1
                action v
end
```

### Proposition 3:

If $a > b$ and $c < d$, then $a + c > b + d$.
The basic interior point method used by Kamath et al. [2] and Karmarkar et al. [5] to solve problem (2.1) involves several steps. First, problem (2.1) is transformed into the equivalent continuous quadratic programming problem

\[
\begin{align*}
\text{Maximize } & x^T x \\
\text{subject to } & \tilde{A} x \leq \tilde{b} \\
& -e \leq x \leq e
\end{align*}
\]

where \( \tilde{A} \in \mathbb{R}^{m \times n} \) and \( \tilde{b} \in \mathbb{R}^m, x \in \mathbb{R}^n, e \in \mathbb{R}^n \) and \( e^T = (1, 1, \ldots, 1) \).

The feasible region of (2.2) is the polytope \( \mathcal{P} \) where

\[
\mathcal{P} = \{ x \in \mathbb{R}^n \mid \tilde{A} x \leq \tilde{b} \text{ and } -e \leq x \leq e \}
\]

Problem (2.2) is NP-complete [8].

The general strategy to solve (2.2) is to choose an interior point \( x^0 \in \mathcal{P} \) and generate a sequence of interior points \( x^k \) such that the potential function \( \psi(x) \) given by

\[
\psi(x) = \log \sqrt{n - x^2} - \eta \sum_{j=1}^m \log d_j(x)
\]

is minimized. In equation (2.3), \( d_j(x) = b_j - \sum_{i=1}^n a_{ij} x_i \) and \( \eta \) is a parameter which is set to \( \frac{1}{2} \) in [5,2]. If \( \Delta x \) is descent direction of \( \psi(x) \) around \( x^k \), then a possible sequence of interior points can be generated by updating

\[
x^{k+1} = x^k + \alpha \Delta x,
\]

where \( \alpha \) is a step size.

Karmarkar et al. [5] show that the steepest descent direction \( \Delta x \) can be determined by solving the following quadratic programming problem:

\[
\begin{align*}
\text{Minimize } & \frac{1}{2} (\Delta x)^T H \Delta x + h^T \Delta x \\
\text{subject to } & (\Delta x)^T A D^2 A^T (\Delta x) \leq r^2 \leq 1
\end{align*}
\]

(2.5)

where the Hessian \( H \) is

\[
H = -\frac{2}{f_0} I - \frac{4}{f_0^2} x^k (x^k)^T + \frac{1}{n} A D^2 A^T,
\]

and the gradient is

\[
h = -\frac{1}{f_0} x^k + \frac{1}{n} A D e.
\]

Problem (2.5) is solved in polynomial time [9].

**Table 4.1**

| Problem | \( |E| \) | \( |P| \) | \( \text{max } C_2 \) | \( \text{max } C_3 \) | Cycles | Bounds  | Optimal Value | CPU time (secs) |
|---------|--------|--------|-----------------|-----------------|--------|--------|----------------|-----------------|
| 1       | 10     | 10     | 5               | 1               | 1      | 85     | 85             | 85              | 16               |
| 2       | 37     | 37     | 117             | 2               | 1      | 144    | 144            | 144             | 37               |
| 3       | 83     | 83     | 114             | 1               | 1      | 754    | 754            | 754             | 20               |
| 4       | 125    | 125    | 113             | 1               | 1      | 1079   | 1079          | 1079            | 20               |
| 5       | 125    | 125    | 113             | 1               | 1      | 1579   | 1579          | 1579            | 5                |
| 6       | 1000   | 5      | 50               | 1               | 1      | 55     | 55             | 55              | 147              |
| 7       | 10     | 10     | 54               | 2               | 1      | 102    | 102            | 102             | 102              |
| 8       | 83     | 83     | 54               | 1               | 1      | 509    | 509            | 509             | 167              |
| 9       | 125    | 125    | 48               | 2               | 1      | 707    | 707            | 707             | 306              |
| 10      | 250    | 250    | 78               | 1               | 1      | 1093   | 1093          | 1093            | 35               |
| 11      | 2500   | 5      | 61               | 2               | 1      | 32     | 32             | 32              | 201              |
| 12      | 10     | 10     | 50               | 1               | 1      | 46     | 46             | 46              | 216              |
| 13      | 83     | 83     | 43               | 3               | 1      | 258    | 258            | 258             | 431              |
| 14      | 125    | 125    | 298              | 1               | 1      | 323    | 323            | 323             | 191              |
| 15      | 250    | 250    | 179              | 1               | 1      | 556    | 556            | 556             | 130              |
| 16      | 12500  | 5      | 48               | 1               | 1      | 11     | 11             | 11              | 176              |
| 17      | 10     | 10     | 77               | 1               | 1      | 18     | 18             | 18              | 192              |
| 18      | 83     | 83     | 50               | 1               | 1      | 113    | 113            | 113             | 952              |
| 19      | 125    | 125    | 192              | 1               | 1      | 146    | 146            | 146             | 171              |
| 20      | 250    | 250    | 0                | 0               | 1      | 267    | 267            | 267             | 200              |

The upper bounds quoted are the ones generated before the end of the first pre-processing/Lagrangean relaxation cycle. The other headings are self-explanatory. Problems in groups C1, C2, C3, D1, and D2 have 2, 6, 1000 and 2500 vertices, respectively. The algorithm is coded in Fortran and run on a Sun SparcStation 2.

As it can be appreciated from results, the lower bounds we derive are quite sharp, and for 57 of the 60 problems involved, lower bounds equaled optimal SPG solution values. This compares with a figure of 15 out of 60 for Beasley [2] (in same cases we have managed to close gaps of almost 30% between Beasley's lower bounds and optimal solution values). In terms of CPU times, for the 57 problems mentioned above, in a rough comparison with the times quoted by Beasley (based on results in Dongarra [5]), we are always faster (in some cases more than 400 times faster). Our results also appear competitive (using results in Dongarra [5] yet again) with the ones quoted by Chopra, Gorges, and Rao [3], for the same set of test problems. It appears that one algorithm does not dominate the other in terms of performance.

The upper bounds we generated turned out equally sharp. For 5 of the problems considered, optimal solutions were generated before the end of the first pre-processing/Lagrangean relaxation cycle. In addition, for problem E18, to which an optimal solution is unknown, we generated the best known upper bound.

We intend to implement an enumeration scheme, based on the lower bounding approach described in this paper, to attempt to solve to optimality the 3 test problems for which positive duality gaps are reported.

**5. Conclusions.** Lagrangean relaxation algorithms that incorporate cutting-planes, in the same vein as we describe here, could prove quite attractive to tackle other combinatorial optimization problems. We feel that this holds true despite the fact that,
### Table 4.1 Computational results with STAR: Part 1

<table>
<thead>
<tr>
<th>Problem</th>
<th>Vars</th>
<th>Clauses</th>
<th>( E(n(c)) )</th>
<th>Algorithm Type</th>
<th>Instances</th>
<th>Iterations</th>
<th>Inits</th>
<th>Min</th>
<th>Mean</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Global</td>
<td>local</td>
<td>1-iter</td>
<td>Min</td>
<td>Mean</td>
<td>Max</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>5.126</td>
<td>[2] (( a = 0.5 ))</td>
<td>89</td>
<td>11</td>
<td>/</td>
<td>1</td>
<td>46.2</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>97</td>
<td>3</td>
<td>60</td>
<td>1</td>
<td>8.6</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>95</td>
<td>5</td>
<td>60</td>
<td>1</td>
<td>1.9</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>200</td>
<td>5.103</td>
<td>[2] (( a = 0.5 ))</td>
<td>82</td>
<td>18</td>
<td>/</td>
<td>1</td>
<td>136.2</td>
<td>1002</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>98</td>
<td>2</td>
<td>35</td>
<td>1</td>
<td>22.7</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>92</td>
<td>8</td>
<td>35</td>
<td>1</td>
<td>3.1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>400</td>
<td>7.029</td>
<td>[2] (( a = 0.5 ))</td>
<td>81</td>
<td>19</td>
<td>/</td>
<td>1</td>
<td>412.6</td>
<td>2070</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>0</td>
<td>59</td>
<td>1</td>
<td>20.4</td>
<td>181</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>0</td>
<td>59</td>
<td>1</td>
<td>2.8</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>800</td>
<td>10.06</td>
<td>[2] (( a = 0.5 ))</td>
<td>86</td>
<td>14</td>
<td>/</td>
<td>1</td>
<td>44.2</td>
<td>1320</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>0</td>
<td>82</td>
<td>0</td>
<td>12.8</td>
<td>289</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>0</td>
<td>82</td>
<td>0</td>
<td>1.8</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>9.996</td>
<td>[2] (( a = 0.5 ))</td>
<td>80</td>
<td>20</td>
<td>/</td>
<td>1</td>
<td>119.4</td>
<td>2350</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>0</td>
<td>82</td>
<td>1</td>
<td>12.3</td>
<td>343</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>0</td>
<td>82</td>
<td>1</td>
<td>1.8</td>
<td>19</td>
<td></td>
</tr>
</tbody>
</table>

The results are summarized in Tables 4.4. For each test, we give the number of variables, the number of clauses, the number of literals per clause, the number of instances which reach the global minimum, the number of instances which reach a local minimum, the minimum iteration number, the mean iteration number, the maximum iteration number, the number of calls to the HARWELL linear system solver MA31 and the CPU time used. The problems were run on a MIPS machine at the University of Waterloo with the optimization flag -O.

We also compared our results to the results in [2]. Since that work was done using a Alliant FX/80 parallel/vector processing computer, it was possible for the researchers to test larger problems. For instance, problems containing 1000 variables and 2000 clauses were solved. We are able to solve fairly large problems containing 500 variables and 1000 clauses. We felt that it would not be fair to compare running times on different computing environments. However, we have compared the number of iterations obtained using the different algorithms; that is, with and without line search.

We can make the following observations about the test results:

- We were able to reproduce the results in [2] for the case where the step size is fixed; that is, \( a = 0.5 \). The interior point method produced satisfactory results for the majority of instances. Our implementation achieves more satisfactory truth assignments than [2]. Large step sizes produce local minima for more instances than small step size.
- All problem classes had at least one instance for which the algorithm produced a satisfactory solution in one iteration.
- The problems become harder as the expected number of literals per clause decrease. It is clear that less feasibility is allowed in determining the value of the variable.

of cost 0, attached to it. Denote by \( G' = (V', E') \) the resulting expanded graph where \( |V'| = |V| + 1 = n' \) and let \( S' \) be a spanning tree of \( G' \). Taking \( x \) as the incidence vector of \( S' \), let \( x_i = 1 \) if \( e \in E \) and \( S' \) is set \( S \) and 0 otherwise. Beasley's extended formulation of the SPG consists of a SST for \( G' \) subject to \( C_1 = \{ x_i, x_{i+1} + 1 : i \in V \setminus T \} \), where \( \delta(i) \) is the set of edges of \( E \) with one endpoint in the vertex indexed by \( i \). Constraints \( C_1 \) split feasible spanning trees into two halves and, if all edges associated with the artificial vertex \( n + 1 \) are eliminated (note that those edges would have a combined cost of 0), a Steiner tree for the original graph \( G \) would emerge. Beasley relaxed \( C_1 \) in a Lagrangean fashion to obtain a valid lower bound for the SPG based on the solution of the resulting unconstrained SST problem.

2.1. Valid Inequalities. Since any Steiner vertex in a SPG solution must have an edge degree of at least 2, a valid inequality that imposes this restriction is given by \( C_2 = \{ x_i : e \in \delta(i) \} + 2x_{n+1} \geq 2 : i \in V \setminus T \} \).

A lifting of the subtour elimination constraints would lead to \( C_3 = \{ x_i (E(W)) + \sum x_{i+1} : i \in W \setminus T \setminus W \subseteq [n] \} + 1 : W \subseteq V \setminus T, W \cap T = \emptyset \} \), where \( E(W) \) denotes the edges with both endpoints in the set of vertices, and \( x_i(W) \) denotes the sum of variables for the edges in the set. Constraints \( C_3 \) are an adaptation, within this context, of inequalities introduced independently by Goemans [7], Lucena [8], and Margot, Prodon, and Liebling [10], for another extended formulation of the SPG. In effect, this other extended formulation involves, in addition to binary variables \( x_{i,e} \in E \), binary variables for vertices in \( V \setminus T \). Denoting those vertex variables \( y_i, i \in V \setminus T \), one has \( x_{i+1} = 1 - y_i, i \in V \setminus T \). Therefore, variables for the edges of an artificial vertex in Beasley's formulation are no different from variables \( y_i, i \in V \setminus T \), for the other formulation (there is a correspondence of one to one between them). Goemans [7] has shown how one can obtain facets defining valid inequalities for the Steiner tree polytope by projecting constraints equivalent to \( C_3 \), for the other formulation, onto the space of variables \( x_{i,e} \in E \).

Goemans [7] and Margot, Prodon, and Liebling [10] introduced another lifting of the subtour elimination constraints that, in this context, can be expressed as \( C_4 = \{ x_i (E(W)) + x_{i+1} : i \in W \setminus T, k \in W \setminus \set{k} \subseteq [n] \} + 1 : W \subseteq V \setminus T, W \cap T = \emptyset, k \in W \cap T \} \) is valid for the SPG.

Polyhedral results similar to the ones described above for \( C_3 \) can also be derived for \( C_4 \) (c.f. Goemans [7]). One should notice that inequalities \( C_1 \), introduced by Beasley, are a special case of inequalities \( C_3 \) and \( C_4 \) for \( |W| = 2 \).

Finally consider a partition of \( V \) into \( V_1 \) and \( V_2 \) where \( V_1 \cap T \neq \emptyset \) and \( V_2 \cap T \neq \emptyset \). Let \( E(V_1, V_2) \) be the set of edges with one endpoint in \( V_1 \) and another endpoint in \( V_2 \). Then \( C_5 = \{ x_i (E(V_1, V_2)) : i \in V \setminus T, V_1 \cap T \neq \emptyset, V_2 \cap T \neq \emptyset, V_1 \cap V_2 = \emptyset \} \) is valid for the SPG. Inequalities \( C_5 \) have been used by Chopra, Gorres and Rao [3] and Lucena and Beasley [9] in their branch and cut algorithms for the SPG. Those inequalities have been shown by Chopra and Rao [4] to be facet-defining for the Steiner tree polytope.

3. Lagrangean Relaxation, Upper Bounds and Reduction Tests. We first solve the SST problem for graph \( G' \). Then constraints of types \( C_2, C_3, C_4 \), and \( C_5 \) that are violated in this SST solution are identified in linear time. Note that this contrasts with the general case where the separation problems associated with constraints \( C_2, C_4 \) and \( C_5 \) could only be solved in \( O(n^4) \) time (that involves solving a number of network flow problems). Violated inequalities, when first detected, are dualized in a Lagrangean fashion. In practical terms, it is as if those constraints have already been dualized (with a multiplier of value 0 being associated with them). We use the Subgradient Method, as suggested by Fisher [11], to optimize the Lagrangean
to the original Karmarkar algorithm for solving 0-1 integer programs is proposed. This
approach adds a line search to the existing interior point method to determine opti-
mal step size rather than using a fixed step size within (0, 1). Computational results
show that the modified interior point approach with optimal step size reduced the
number of iterations by a factor of 5-15, and the number of calls to the linear system
solver by a factor of 2-6, with only about one percent extra time for line search. For
a majority of instances of the Satisfiability problem, satisfiability was proven in less
than 3 iterations.

REFERENCES
experience with an interior point algorithm on the Satisfiability problem," Annals of Op-
Mathematics 114 (1990) 297-308.
to solve computationally difficult set covering problems," Mathematical Program-
ing 52 (1991) 597-618.
[8] S.A. Vavasis, "Quadratic programming is in NP," Information Processing Letters 36 (1990) 73-
77.
[9] Y. Ye, "On the interior algorithms for nonconvex quadratic programming," to appear Mathemat-

SPECIAL ISSUE ON COMPUTATIONAL ASPECTS OF
COMBINATORIAL OPTIMIZATION
EDITORIAL

This issue of the COAL Bulletin includes a collection of invited papers that reflects
recent research in solving NP-hard combinatorial optimization problems.

In the first paper, Lucena studies the use of cutting-planes to strengthen a La-
grangean relaxation lower bound for the Steiner Problem in Graphs. Computational
results on 60 problems show that the approach improves significantly the results of

The paper by Kumar and Ananth provides an overview of parallel algorithms for
solving combinatorial optimization problems. They discuss parallel depth first and
best first search, as well as speedup anomalies and scalability issues.

In the paper by Mitchell and Borchers, an interior point primal-dual cutting plane
algorithm for solving the linear ordering problem is given. Promising computational
results, on real-world problems, are presented.

Pardalos and Vairaktarakis present instances of maximum clique problems from
various applications. The availability of such collections of test problems will facilitate
the efforts of comparing performance and correctness of the many proposed algorithms
and heuristics for solving the maximum clique problem.

Shi, Vannelli and Vlach describe an improvement in the implementation of Kar-
markar's interior point algorithm for integer programming. They apply their imple-
mentation on the randomly generated instances of Satisfiability described in Kamath
et al. (1990) and show that much improvement can be achieved by using a dynamic
step size.

We would like to thank the editors for inviting us to prepare this special issue and
to thank the contributors for helping us to produce this excellent collection of papers.

Panos M. Pardalos
University of Florida

Mauricio G.C. Resende
AT&T Bell Laboratories
This issue will be addressed in the Bulletin.

The Bulletin's primary objective is to provide a vehicle for the rapid dissemination of

BULLETIN OBJECTIVES

EX OFFICIO MEMBERS

USA

Canada

Europe

Japan

The Committee on Algorithms is involved in computer science developments in mathematical

COAL OBJECTIVES

PROCEEDINGS OF THE

Committee on Algorithms of the Mathematical
### Mathematical Programming Society
### Committee on Algorithms
### Bulletin

<table>
<thead>
<tr>
<th>NO. 21</th>
<th>JENS CLAUSEN</th>
<th>FAIZ A. AL-KHAYYAL</th>
<th>EDITORS</th>
</tr>
</thead>
<tbody>
<tr>
<td>November 1992</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SPECIAL ISSUE ON COMPUTATIONAL ASPECTS OF COMBINATORIAL OPTIMIZATION**

Panos M. Pardalos  
Mauricio G. C. Resende  

**Guest Editors**

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editorial</td>
<td>1</td>
</tr>
<tr>
<td>Steiner Problem in Graphs: Lagrangean Relaxation and Cutting-Planes</td>
<td>2</td>
</tr>
<tr>
<td>Abilio Lucena</td>
<td></td>
</tr>
<tr>
<td>Parallel Algorithms for Discrete Optimization Problems</td>
<td>8</td>
</tr>
<tr>
<td>V. Kumar and Grama Y. Ananth</td>
<td></td>
</tr>
<tr>
<td>A Primal-Dual Interior Point Cutting Plane Method for the Linear</td>
<td>13</td>
</tr>
<tr>
<td>Ordering Problem</td>
<td></td>
</tr>
<tr>
<td>John E. Mitchell and Brian Borchers</td>
<td></td>
</tr>
<tr>
<td>Test Cases for the Maximum Clique Problem</td>
<td>19</td>
</tr>
<tr>
<td>Panos M. Pardalos and George Vairaktarakis</td>
<td></td>
</tr>
<tr>
<td>An Improvement on Karmarkar's Algorithm for Integer Programming</td>
<td>23</td>
</tr>
<tr>
<td>C.-J. Shi, A. Vannelli, and J. Vlach</td>
<td></td>
</tr>
</tbody>
</table>