SAGL: A New Heuristic for Multi-Robot Routing with Complex Tasks

Hong Xu*, T. K. Satish Kumar*, Dylan Johnke†, Nora Ayanian* and Sven Koenig*
*University of Southern California, Los Angeles, CA 90089, USA
†Cornell University, Ithaca, NY 14853, USA
hongx@usc.edu, {tkskwork, dylanjohnke}@gmail.com, {ayanian, skoenig}@usc.edu

***Abstract*—In this paper, we study the Complex Routing Problem (CRP), where several homogeneous robots need to visit given task locations to accomplish complex tasks in a cooperative setting. Each task location hosts a task. The complexity level of a task is defined as the number of robots that need to be simultaneously present at its location to accomplish it. The robots need to be routed so that all tasks get accomplished with minimal makespan. We present a new centralized algorithm, called SAGL, for solving the CRP heuristically. SAGL is inspired by the application of linear programming duality to the Steiner Forest Problem. It makes less restrictive assumptions than the state-of-the-art distributed Approach with Reaction Functions and scales better in both the complexity levels of tasks and the number of complex tasks (whose complexity levels are greater than one), although it results in somewhat larger makespans.**

I. Introduction

In this paper, we study the Complex Routing Problem (CRP), where several homogeneous robots need to visit given task locations to accomplish complex tasks in a cooperative setting. Each task location hosts a task. The complexity level of a task is defined as the number of robots that need to be simultaneously present at the location to accomplish it. A task (and its location) is called simple if its complexity level is one and complex otherwise. The robots need to be routed so that all tasks get accomplish with minimal makespan (the time when the last task gets accomplished). The initial robot locations, the task locations and the complexity levels of tasks are known. All locations are embedded in a metric space where the triangle inequality holds, which is realistic since the locations are often embedded into the Euclidean plane with or without obstacles. The CRP arises in real-world applications. For example, multiple robots might be required to lift heavy pieces of debris in a search-and-rescue domain.

The Traveling Salesman Path Problem (finding a Hamiltonian path with a given start vertex) is a special case of the CRP where there is only one robot and all tasks are simple. Thus, all intractability and inapproximability results about the Traveling Salesman Path Problem carry over to the CRP. For example, the Traveling Salesman Path Problem and thus also the CRP are NP-hard to solve optimally, and at least the triangle inequality is needed to approximate them efficiently [1]. The CRP has previously been solved [2]. The Approach with Reaction Functions (ARF) [3] [4] is considered a state-of-the-art CRP approach for the minimization of both makespan and the sum of the travel distances of the robots [5], [6], [7], [8]. A reaction function characterizes the cost of a given robot for visiting a given complex task location at a given time in addition to all its assigned task locations. In each iteration of ARF, each robot submits to an auctioneer its reaction functions for all complex tasks, and the auctioneer then assigns more tasks to robots. ARF was inspired by auction-like approaches for a special case of the CRP where all tasks are simple [9]. On the other hand, game-theoretic approaches are typically used in a competitive setting [10], [11], where self-interested agents try to maximize their own utilities [12], [13].

In this paper, we present a new centralized algorithm, called SAGL (which stands for its 4 steps: Spanning tree construction, Task assignment, Global visitation order determination and Local visitation order determination), for solving the CRP heuristically. SAGL is inspired by the application of linear programming duality to design a two-approximation algorithm for the Steiner Forest Problem [14]. SAGL makes less restrictive assumptions than ARF and scales better in both the number of complex tasks and their complexity levels, although it results in somewhat larger makespans. For example, the runtime of ARF is exponential in the number of complex tasks assigned to each robot and their maximum complexity level, which is why ARF imposes upper bounds on these quantities [3]. The runtime of SAGL, on the other hand, is polynomial without such restrictions.

II. Problem Formulation

The CRP can be mathematically formulated as a septuple $\langle \mathcal{V}, \mathcal{R}, \mathcal{T}, d, \tau, \ell, c \rangle$, where

- $\mathcal{V}$ is the set of locations;
- $\mathcal{R}$ is the set of robots;
- $\mathcal{T}$ is the set of tasks;
- d is the distance function that maps two locations to a positive real number ($d : \mathcal{V} \times \mathcal{V} \to \mathbb{R}_{>0}$), which represents the distance from the first to the second location;
- τ is the traversal function that maps two locations to a positive real number ($\tau : \mathcal{V} \times \mathcal{V} \to \mathbb{R}_{>0}$), which represents the travel time from the first to the second location;
- ℓ is the function that maps a robot to its initial location and a task to its location ($\ell : \mathcal{R} \cup \mathcal{T} \to \mathcal{V}$); and
- c is the function that maps a task to a positive integer ($c : \mathcal{T} \to \mathbb{N}^+$), which represents its complexity level.

A candidate solution is an assignment of task locations to all robots along with a visitation order for each robot that determines in which order the robot has to visit its assigned task locations. Each location of some task $t \in \mathcal{T}$ has to be assigned to exactly $c(t)$ different robots. When a robot arrives at the location of task t, it waits until $c(t)$ robots (including itself) are present. The earliest such time is the completion time of the task since we assume without loss of generality that all tasks are accomplished instantaneously once all robots are present.[1] The makespan of a candidate solution is the largest completion time of any task. A candidate solution with a deadlock (where some robot waits forever for other robots) has an infinite makespan. A candidate solution with minimum makespan is called optimal. The task is to find a close-to-optimal candidate solution.

We impose the following restrictions in this paper that we intend to relax in future work: We assume that all distances satisfy the triangle inequality, as explained earlier. We also assume that all distances are symmetrical and that all robots always move with unit speed. Thus, the travel time between locations is equal to their distance. These restrictions allow us to embed a CRP instance into a complete undirected edge-weighted graph $G = \langle V_G, E_G \rangle$, where each vertex represents a task location or an initial robot location and each edge has a cost that represents the travel time between the two locations corresponding to its endpoint vertices.

III. Background

SAGL draws on ideas from the application of linear programming duality to the Steiner Forest Problem and from approximations of the Traveling Salesman Problem.

A. Steiner Forest Problem

The Steiner Forest Problem is defined as follows: Given an undirected edge-weighted graph $G = \langle V, E \rangle$ and a collection $S = \{S_1, S_2, S_3, \ldots, S_n\}$ of sets $S_i \subseteq V$, find a minimum edge-cost forest F such that, for all S_i and all $u, v \in S_i$, there exists a path connecting u and v in F. The Steiner Forest Problem can be formulated as an integer linear program where each edge $e \in E$ is associated with a $0/1$ variable x_e:

$$\begin{aligned} \min \ & \sum_{e \in E} c_e x_e \\ \text{s.t.} \ & \sum_{e \in \partial H} x_e \geq 1, \qquad \forall H \in \mathcal{H} \\ & x_e \in \{0, 1\}, \qquad \forall e \in E \end{aligned} \tag{1}$$

c_e is the cost of edge e, H is a subset of V which cuts some S_i into two parts, $\mathcal{H}$ is the set of all such subsets, and ∂H is the set of all edges which have exactly one endpoint vertex in H. A two-approximation of the Steiner Forest Problem can be obtained by relaxing the integer linear program to a linear program and then solving it using a primal-dual method [14]. Each constraint in the dual formulation corresponds to a variable in the primal formulation and thus also an edge in the graph. In this two-approximation algorithm, each iteration uniformly increases the values of all dual variables corresponding to every minimal unsatisfied set S_i until a dual constraint becomes tight. When a dual constraint becomes tight, the edge corresponding to this constraint is added to F and all dual variables are frozen. The two-approximation algorithm terminates when F is a feasible solution. Intuitively, each iteration of the two-approximation algorithm can be understood as expanding "discs of influence" around minimal unsatisfied sets until two of them "touch" each other along some edge, which is then added to F.

B. Traveling Salesman Problem

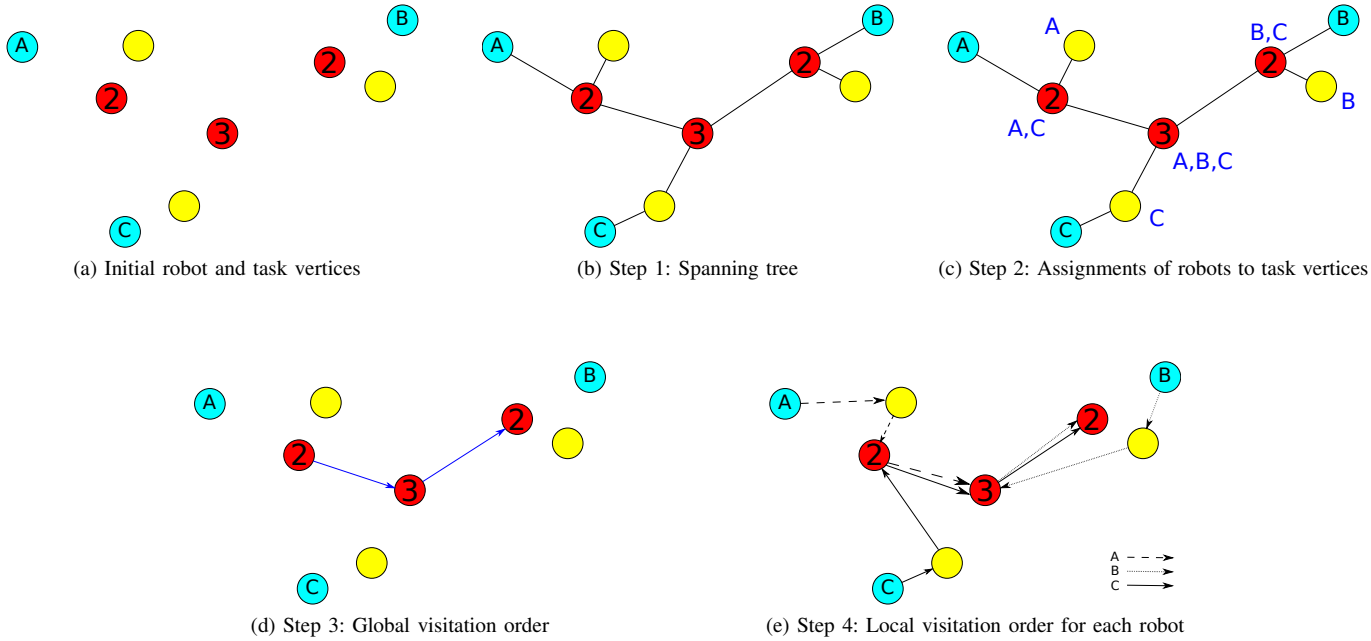
The Traveling Salesman Problem (TSP) is defined as follows: Given a complete undirected edge-weighted graph $G = \langle V, E \rangle$, find a minimum-cost cycle that includes all vertices exactly once. A TSP whose graph obeys the triangle inequality is called metric. A two-approximation of the metric TSP can be obtained by first finding a minimum spanning tree T of G and then performing a depth-first traversal with short-circuiting [15] on T [16], starting with an arbitrary vertex—resulting in our metric TSP solver. A TSP with the constraint that the cycle must be consistent with a given total visitation order of a subset of the vertices (the "constrained vertices") is called path-constrained. A three-approximation of the path-constrained metric TSP can be obtained as follows [17]—resulting in our path-constrained metric TSP solver:

1) Add an auxiliary vertex and connect it to all constrained vertices with zero-cost auxiliary edges.
2) Find a minimum spanning tree T on the resulting graph. The auxiliary edges from Step 1 are in T since they have zero costs.
3) Remove the auxiliary vertex and all auxiliary edges from T. None of the constrained vertices are connected to each other afterward since they were all connected via auxiliary edges to the auxiliary vertex. Thus, T now contains a number of connected components, each of which contains exactly one constrained vertex.
4) Perform a depth-first traversal with short-circuiting [15] on each connected component of T, starting with the constrained vertex. Connect the start and end vertices of the resulting paths to a cycle that obeys the given visitation order of the constrained vertices.

IV. SAGL: A New CRP Algorithm

The CRP consists of two interrelated subproblems, namely determining which robots should visit which task vertices and in which visitation order the robots should visit them. SAGL, our new heuristic CRP algorithm, decouples the solution of these two subproblems. Steps 1 and 2 solve the first subproblem, and Steps 3 and 4 solve the second subproblem. Figure 1 shows an example of the working of SAGL, which runs in polynomial time (since each step runs in polynomial time) and prevents deadlocks.

[1] If this is not the case, then the time needed to accomplish the task can be folded into the travel time.



(a) Initial robot and task vertices (b) Step 1: Spanning tree (c) Step 2: Assignments of robots to task vertices

(d) Step 3: Global visitation order (e) Step 4: Local visitation order for each robot

Fig. 1: Illustrates the working of SAGL. Cyan circles are initial robot vertices, yellow circles are simple task vertices, and red circles are complex task vertices labeled with the complexity levels of the tasks. (a) The figure shows the initial robot and task vertices on the Euclidean plane. (b) The figure shows the spanning tree resulting from Step 1. (c) The figure shows the assignment of robots to task vertices (in blue font) resulting from Step 2. (d) The figure shows the global visitation order of the complex task vertices resulting from Step 3. (e) The figure shows the local visitation order of its assigned task vertices for each robot resulting from Step 4.

Algorithm 1: Spanning tree construction.

Function `SpanningTree()`
  Output: a spanning tree of G.
  $M = \langle V, E\rangle := \langle V_G, \emptyset\rangle$;
  while *M has more than one connected component* **do**
    $E' :=$ edges in E connecting two connected components of M;
    $E := E \cup \{\text{argmin}_{(u,v)\in E'}\, d(u,v)/(g(u)+g(v))\}$;
  return M;
Function `g(u)`
  Input: u: a vertex of M.
  Output: the growth rate of u.
  $CC = \langle V, E\rangle :=$ the connected component of M containing u;
  return $\max\{1, \max_{t\in T|l(t)\in V}\{c(t)\} - |\{r \in R|l(r) \in V\}|\}$;

A. Step 1: Spanning Tree Construction

Step 1 builds a spanning tree M that Step 2 then uses to assign all task vertices to robots, see Algorithm 1. Step 1 is inspired by the application of linear programming duality to the Steiner Forest Problem [14]. A connected component in the following text corresponds to a minimal unsatisfied set, and the growth rate $g(u)$ of connected component u corresponds to the expansion rate of the disc of influence around the corresponding minimal unsatisfied set.

Step 1 initializes M to contain the vertices V_G and no edges. M thus contains $|V_G|$ connected components. Step 1 expands a disc during each iteration around each vertex with the growth rate of the vertex, which is the growth rate of the connected component that contains the vertex. It adds an edge to M between the first pair of vertices from different connected components whose discs touch (ties can be broken arbitrarily), which merges the two connected components into one and might change the growth rates of the vertices that it contains. Step 1 terminates when there is only one connected component. Thus, the resulting connected component is a spanning tree.

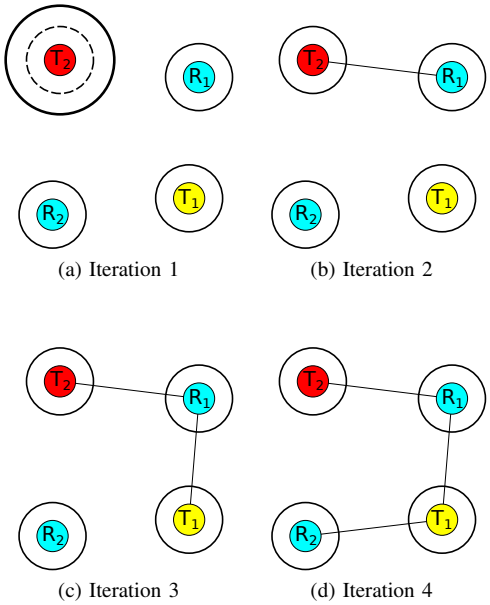
The growth rate of a connected component is defined to be the maximum complexity level of all tasks whose vertices are contained in it minus the number of robots whose initial vertices are contained in it. The idea behind this definition is that the robots whose initial vertices are contained in it will likely be assigned to the task vertices that are contained in it. The growth rate of the connected component, if it is not less than one, is thus the smallest number of additional robots that need to be assigned to the task vertices in it. If the growth rate is less than one, then it is set to one to ensure that Step 1 builds a spanning tree rather than a spanning forest.

B. Step 2: Task Assignment

Step 2 uses the spanning tree M from Step 1 to assign all task vertices to robots. The vertex of each task t is assigned to its nearest $c(t)$ initial robot vertices using the distances on the spanning tree.

C. Step 3: Global Visitation Order Determination

Step 3 determines a global visitation order D of the complex task vertices by running the metric TSP solver on only the



(a) Iteration 1 (b) Iteration 2

(c) Iteration 3 (d) Iteration 4

Fig. 2: Illustrates Step 1. Cyan circles are initial robot vertices, yellow circles are simple task vertices, and red circles are complex task vertices. Tasks T_1 and T_2 have complexity levels one and two, respectively. The outside circles represent the growth rates of the vertices. (a) The growth rate of the vertex of T_2 is two, and all other growth rates are one. The discs of the vertex of T_2 and the initial vertex of R_1 touch each other first. (b) The edge between these vertices is added to M to form a new connected component with one robot and one task with complexity level 2. Thus, the growth rates of all vertices in this connected component are one, resulting in all growth rates now being one. The discs of the vertex of T_1 and the initial vertex of R_1 touch each other next. (c) The edge between these vertices is added to M. All growth rates remain one. The discs of the vertex of T_1 and the initial vertex of R_2 touch each other next. (d) The edge between these vertices is added to M, resulting in the spanning tree.

complex task vertices (that is, on the subgraph of G induced by these vertices) and then removing the edge with the largest cost from the resulting cycle. It then uses the resulting path for the global visitation order. All robots have to visit their assigned complex task vertices in the global visitation order, which makes it impossible for them to wait for each other and thus prevents deadlocks. They can visit their assigned simple task vertices between their assigned complex task vertices.

D. Step 4: Local Visitation Order Determination

Step 4 determines the local visitation order of its assigned task vertices for each robot by running the path-constrained metric TSP solver on only its initial vertex and its assigned task vertices (that is, on the subgraph of G induced by these vertices) with the constraint that the cycle must be consistent with the global visitation order D of the complex task vertices from Step 3. It then removes the edge in the cycle that returns to the initial robot vertex and uses the resulting path (starting with the initial robot vertex) for the local visitation order. Each robot visits its assigned task vertices in the local visitation order. When it arrives at the vertex of some task t, it waits until $c(t)$ robots (including itself) are present.

V. Experimental Evaluation

We implemented SAGL in Java using the JGraphT library [18] and compiled it using the Open Java Development Kit 8. The authors of [3] and [4] provided their ARF implementation in C to us. We compiled it using gcc 4.9.2 with the "-O3" option. We ran our experiments on a GNU/Linux workstation with an Intel Xeon Processor E3-1240 v3 (8MB Cache, 3.4GHz) and 16GB of RAM.

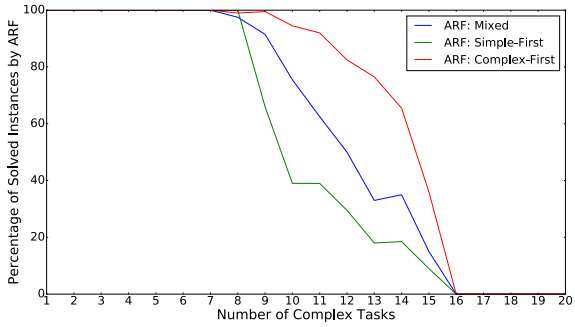


Fig. 3: Shows the percentages of CRP instances that were solved by the "Mixed," "Simple-First" and "Complex-First" variants of ARF [4] for each number of complex tasks within a time limit of two minutes. SAGL solved all CRP instances within 0.4 seconds and is thus not shown in the figure.

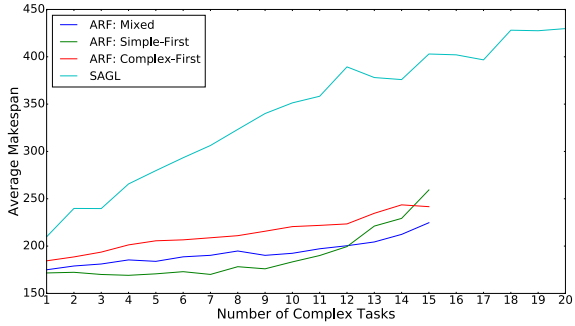
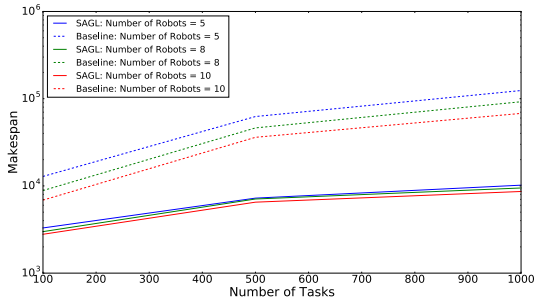


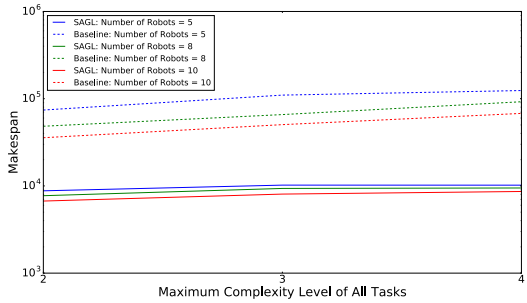
Fig. 4: Shows the makespans for each number of complex tasks, averaged over all CRP instances that were solved by SAGL and the "Mixed," "Simple-First" and "Complex-First" variants of ARF [4], respectively, within a time limit of two minutes. The variants of ARF did not solve any CRP instances with more than 15 complex tasks, which is why their graphs end early.

A. Experiment 1

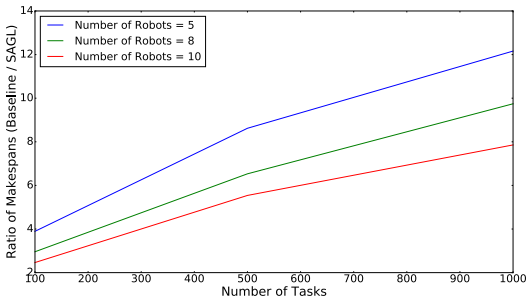
For Experiment 1, we used a discrete 51×51 four-neighbor grid map with square cells that are either blocked or un-



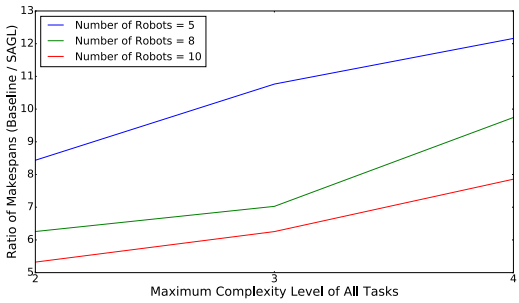
(a) Average makespans (in log scale) for each number of tasks for maximum complexity level four



(b) Average makespans (in log scale) for each maximum complexity level for 1,000 tasks



(c) Ratios of the average makespans (baseline/SAGL) for each number of tasks for maximum complexity level four



(d) Ratios of the average makespans (baseline/SAGL) for each maximum complexity level for 1,000 tasks

Fig. 5: Shows the makespans for SAGL and the baseline algorithm. Each line color represents a different number of robots.

blocked, as used before in [4]. The grid map models an office environment with walls and doors [19]. For each number of complex tasks from one to twenty, we generated 200 CRP instances with random vertices for 10 robots, 80 simple tasks and the given number of complex tasks. Figure 3 shows that SAGL scales better than ARF in the number of complex tasks, and Figure 4 shows that the makespans of SAGL are between 1.5 and 2.0 times larger than the ones of ARF.

B. Experiment 2

For Experiment 2, we used larger CRP instances than ARF can solve within a time limit of two minutes. We used an obstacle-free 300×300 continuous square. For 5, 8 and 10 robots, we generated 15 CRP instances with random vertices for the given number of robots and 100, 500 and 1,000 tasks, where the complexity level of each task is randomly chosen between 1 and a maximum complexity level of 2, 3 and 4. SAGL solved all CRP instances within the time limit. We compared it against the following baseline algorithm:

1) Assign the vertex of each task t to its nearest $c(t)$ initial robot vertices using the distances on G.
2) Generate a random global visitation order of all complex task vertices to prevent deadlocks (for the reason given in the context of Step 3 of SAGL).
3) Run the metric TSP solver for each robot on only its initial vertex and its assigned simple task vertices (that is, on the subgraph of G induced by these vertices), remove the edge incident on the initial robot vertex with the largest cost from the resulting cycle and use the resulting path (starting with the initial robot vertex) for its visitation order of its assigned simple task vertices.
4) Create two task queues for each robot, one for its assigned complex task vertices in the visitation order produced in Step 2 and one for its assigned simple task vertices in the visitation order produced in Step 3. The robot then randomly chooses one of the two queues, dequeues the next task vertex from it and visits that task vertex, until both queues are empty.

Figure 5 shows that the makespan of SAGL gets smaller relative to the one of the baseline algorithm as the number of tasks and the maximum complexity level increase.

VI. Conclusions and Future Work

We studied the Complex Routing Problem (CRP) and presented a new centralized algorithm, called SAGL, for solving it heuristically. SAGL is inspired by the application of linear programming duality to the Steiner Forest Problem. It makes less restrictive assumptions than the state-of-the-art distributed Approach with Reaction Functions and scales better in both the number of complex tasks and their complexity levels, although it results in somewhat larger makespans.

Directions for future work include relaxing the restrictions that we imposed on the CRP, developing a distributed version of SAGL and extending it to heterogeneous robots and tasks (for example, where tasks require robots with specific capabilities) and tasks with flexible complexity levels, where the

time needed to accomplish a task depends on the number of robots present.

VII. Acknowledgments

We thank Xiaoming Zheng for making his implementation of ARF available to us and Anagha Kulkarni for implementing a prototype of SAGL. Our research was supported by NSF under grant numbers 1409987 and 1319966 and a MURI under grant number N00014-09-1-1031. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies or the U.S. government.

References

[1] M. Charikar, R. Motwani, P. Raghavan, and C. Silverstein, "Constrained TSP and low-power computing," in *Algorithms and Data Structures*. Springer, 1997, pp. 104–115.

[2] A. Viguria, I. Maza, and A. Ollero, "S+T: An algorithm for distributed multirobot task allocation based on services for improving robot cooperation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2008, pp. 3163–3168.

[3] X. Zheng and S. Koenig, "Reaction functions for task allocation to cooperative agents," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2008, pp. 559–566.

[4] ——, "Generalized reaction functions for solving complex-task allocation problems," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2011, pp. 478–483.

[5] D. Ye, Q. Bai, M. Zhang, K. T. Win, and Z. Shen, "An efficient task allocation protocol for P2P multi-agent systems," in *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing with Applications*, 2009, pp. 11–18.

[6] S. D. Ramchurn, A. Farinelli, K. S. Macarthur, and N. R. Jennings, "Decentralized coordination in RoboCup rescue," *The Computer Journal*, vol. 53, no. 9, pp. 1447–1461, 2010.

[7] Y. Altshuler, V. Yanovski, I. A. Wagner, and A. M. Bruckstein, "Multi-agent cooperative cleaning of expanding domains," *The International Journal of Robotics Research*, vol. 30, no. 8, pp. 1037–1071, 2010.

[8] Y. Kong, M. Zhang, D. Ye, and X. Luo, "A negotiation method for task allocation with time constraints in open grid environments," in *Next Frontier in Agent-based Complex Automated Negotiation*, ser. Studies in Computational Intelligence. Springer, 2015, vol. 596, pp. 19–36.

[9] M. B. Dias and A. Stentz, "Opportunistic optimization for market-based multirobot control," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, 2002, pp. 2714–2720.

[10] V. D. Dang and N. R. Jennings, "Generating coalition structures with finite bound from the optimal guarantees," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2004, pp. 564–571.

[11] J. P. Kahan and A. Rapoport, *Theories of coalition formation*. Psychology Press, 1984.

[12] S. Kraus, O. Shehory, and G. Taase, "Coalition formation with uncertain heterogeneous information," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2003, pp. 1–8.

[13] T. W. Sandholm and V. R. Lesser, "Coalitions among computationally bounded agents," *Artificial Intelligence*, vol. 94, no. 1-2, pp. 99–137, 1997.

[14] M. X. Goemans and Y.-S. Myung, "A catalog of Steiner tree formulations," *Networks*, vol. 23, no. 1, pp. 19–28, 1993.

[15] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, 1956.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.

[17] A. Bachrach, K. Chen, C. Harrelson, R. Mihaescu, S. Rao, and A. Shah, "Lower bounds for maximum parsimony with gene order data," in *Comparative Genomics*, ser. Lecture Notes in Bioinformatics. Springer, 2005, vol. 3678, pp. 1–10.

[18] B. Naveh and Contributors, "JGraphT—a free Java graph library," 2015.

[19] S. Koenig, C. Tovey, X. Zheng, and I. Sungur, "Sequential bundle-bid single-sale auction algorithms for decentralized control," in *Proceedings of the International Joint Conference on Artifical Intelligence*, 2007, pp. 1359–1365.